23rd International Conference on Database Theory

ICDT 2020, March 30–April 2, 2020, Copenhagen, Denmark

Edited by

Carsten Lutz
Jean Christoph Jung

LIPICS

Editors

Carsten Lutz
University of Bremen, Germany
clu@uni-bremen.de

Jean Christoph Jung
University of Bremen, Germany
jeanjung@uni-bremen.de

ACM Classification 2012
Information systems → Data management systems; Information systems → Database design and models; Information systems → Database query processing; Information systems → Query languages; Information systems → Relational database model; Information systems → Parallel and distributed DBMSs; Information systems → Information integration; Information systems → Stream management; Theory of computation → Incomplete, inconsistent, and uncertain databases; Theory of computation → Complexity theory and logic; Theory of computation → Database theory

ISBN 978-3-95977-139-9

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-139-9.

Publication date
March, 2020

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICDT.2020.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

Preface

The 23. International Conference on Database Theory (ICDT 2020) was held in Copenhagen, Denmark, from March 30 to April 2, 2020. The Program Committee has selected 22 research papers out of 69 submissions for publication at the conference. It has further decided to give the best paper award to *A Dichotomy for Homomorphism-Closed Queries on Probabilistic Graphs* by Antoine Amarilli and İsmail İlkan Ceylan. We congratulate the winners! Apart from the 22 regular papers, these proceedings include abstracts for the invited (shared) EDBT/ICDT keynotes by Benny Kimelfeld (Technion, Israel) and by Juan L. Reutter (PUC Chile) and the invited paper associated with the ICDT invited talk by Jerzy Marcinkowski (University of Wrocław, Poland).

A committee formed by Frank Neven, Andreas Pieris, and Jorge Pérez has decided to give the Test of Time Award for ICDT 2020 to the ICDT 2010 paper *Foundations of SPARQL query optimizations* by Michael Schmidt, Michael Meier, and Georg Lausen. We congratulate also the winners of this award!

We would like to thank all people who contributed to the success of ICDT 2020, including the authors of all submitted papers, keynote and invited talk speakers, and, of course, all members of the Program Committee as well as the external reviewers, for the very substantial work that they have invested over the two submission cycles of ICDT 2020. Their commitment and sagacity were crucial to ensure that the final program of the conference satisfies the highest standards. We would also like to thank the ICDT Council members for their support on a wide variety of matters, the local organizers of the EDBT/ICDT 2020 conference, led by General Chairs Yongluan Zhou and Marcos Antonio Vaz Salles, for the great job they did in organizing the conference and co-located events. Finally, we wish to acknowledge Dagstuhl Publishing for their support with the publication of the proceedings in the LIPIcs (Leibniz International Proceedings in Informatics) series.

Carsten Lutz and Jean Christoph Jung
March 2020

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

General Chairs

Yongluan Zhou (University of Copenhagen)

Marcos Antonio Vaz Salles (University of Copenhagen)

Program Chair

Carsten Lutz (University of Bremen)

Proceedings Chair

Jean Christoph Jung (University of Bremen)

Program Committee

Marcelo Arenas (PUC, Santiago de Chile)

Michael Benedikt (University of Oxford)

Christoph Berkholz (HU Berlin)

Angela Bonifati (University Lyon 1)

Pierre Bourhis (University of Lille)

James Cheney (University of Edinburgh)

Graham Cormode (University of Warwick)

Victor Dalmau (UPF, Barcelona)

Claire David (University Paris-Est)

Floris Geerts (University of Antwerp)

Bas Ketsman (EPFL, Lausanne)

Daniel Kifer (Penn State University)

Leonid Libkin (University of Edinburgh)

Sebatian Maneth (University of Bremen)

Filip Murlak (University of Warsaw)

Reinhard Pichler (TU Vienna)

Andreas Pieris (University of Edinburgh)

Sebastian Rudolph (TU Dresden)

Thomas Schwentick (University of Dortmund)

Uri Stemmer (Ben-Gurion University, Negev)

Domagoj Vrgoc (PUC, Santiago de Chile)

Frank Wolter (University of Liverpool)

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Diego Arroyuelo

Jan Van den Bussche

Nadime Francis

Alan Fekete

Victor Marsault

Stefan Mengel

Matthias Niewerth

Liat Peterfreund

Ling Ren

Juan L. Reutter

Cristian Riveros

Dimitris Sacharidis

Oskar Skibski

Rossano Venturini

Nils Vortmeier

Marcin Waniek

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contributing Authors

Heba Aamer

Antoine Amarilli

Pablo Barceló

Leopoldo Bertossi

Bart Bogaerts

Jan Van den Bussche

Yu Chen

Johannes Doleschal

Diego Figueira

Henrik Forssell

Dominik D. Freydenberger

Gaetano Geck

Floris Geerts

Alejandro Grez

Martin Grohe

Jelle Hellings

Nelson Higuera

Ismail Ilkan Ceylan

Batya Kenig

Bas Ketsman

Evgeny Kharlamov

Benny Kimelfeld

Christoph Koch

Peter Lindner

Ester Livshits

Wim Martens

Gonzalo Navarro

Frank Neven

Rasmus Pagh

Jorge Pérez

Liat Peterfreund

Juan L. Reutter

Cristian Riveros

Javiel Rojas-Ledesma

Mohammad Sadoghi

Jorge Salas

Thomas Schwentick

Moshe Sebag

Michael Simpson

Johan Sivertsen

Venkatesh Srinivasan

Bernardo Subercaseux

Dan Suciu

Dimitri Surinx

Yufei Tao

Evgenia Ternovska

Alex Thomo

Samuel M. Thompson

Evgenij Thorstensen

Martin Ugarte

Stijn Vansummeren

Ke Yi

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICDT 2020 Test of Time Award

In 2013, the International Conference on Database Theory (ICDT) began awarding the ICDT test-of-time (ToT) award, with the goal of recognizing one paper, or a small number of papers, presented at ICDT a decade earlier that have best met the "test of time". In 2020, the award recognizes a paper from the ICDT 2010 proceedings that has had the most impact in terms of research, methodology, conceptual contribution, or transfer to practice over the past decade. The award was presented during the EDBT/ICDT 2020 Joint Conference, March 30-April 2, 2020, in Copenhagen, Denmark.

The 2020 ToT Committee consists of Frank Neven (chair), Andreas Pieris and Jorge Pérez. After careful consideration and soliciting external assessments, the committee has chosen the following recipient of the 2020 ICDT Test of Time Award:

Foundations of SPARQL query optimization
Michael Schmidt, Michael Meier, Georg Lausen

This paper is one of the stepping stones that placed Semantic Web query languages on the radar of Database Theory. The paper focuses on SPARQL, the standard language for querying the graph-based model underlying Semantic Web data. It presents an elegant complexity analysis of SPARQL pinpointing the impact of every single operator of the language. It also derives an impressive set of optimization rules highlighting the similarities as well as the important differences between SPARQL and more classical languages such as relational algebra and SQL.

The paper has had a substantial impact counting more than 300 citations. It has influenced the theoretical development of SPARQL and its extensions, the design and construction of Benchmarks for comparing implementations, and also the now ubiquitous research on knowledge-graph data and queries.

Frank Neven	Andreas Pieris	Jorge Pérez
Hasselt University	University of Edinburgh	Universidad de Chile

The ICDT Test-of-Time Award Committee for 2020

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Facets of Probabilistic Databases

Benny Kimelfeld
Technion – Israel Institute of Technology, Haifa, Israel
bennyk@cs.technion.ac.il

Abstract

Probabilistic databases are commonly known in the form of the tuple-independent model, where the validity of every tuple is an independent random event. Conceptually, the notion is more general, as a probabilistic database refers to any probability distribution over ordinary databases. A central computational problem is that of marginal inference for database queries: what is the probability that a given tuple is a query answer? In this talk, I will discuss recent developments in several research directions that, collectively, position probabilistic databases as the common and natural foundation of various challenges at the core of data analytics. Examples include reasoning about uncertain preferences from conventional distributions such as the Mallows model, data cleaning and repairing in probabilistic paradigms such as the HoloClean system, and the explanation of query answers through concepts from cooperative game theory such as the Shapley value and the Banzhaf Power Index. While these challenges manifest different facets of probabilistic databases, I will show how they interrelate and, moreover, how they relate to the basic theory of inference over tuple-independent databases.

2012 ACM Subject Classification Theory of computation → Incomplete, inconsistent, and uncertain databases; Mathematics of computing → Probabilistic representations; Information systems → Data model extensions

Keywords and phrases Probabilistic databases, data cleaning, preference models, Shapley value

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.1

Category Invited Talk

Funding This work was supported by the Israel Science Foundation (ISF) Grants 1295/15 and 768/19, the U.S.-Israel Binational Science Foundation (BSF) Grant 2017753, and the Deutsche Forschungsgemeinschaft (DFG) Project 412400621 (DIP program).

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 1; pp. 1:1–1:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

What Makes a Variant of Query Determinacy (Un)Decidable?

Jerzy Marcinkowski
Institute of Computer Science, University of Wrocław, Poland

Abstract

This paper was written as the companion paper of the ICDT 2020 invited tutorial. Query determinacy is a broad topic, with literally hundreds of papers published since late 1980s. This paper is not going to be a "survey" but rather a personal perspective of a person somehow involved in the recent developments in the area.

First I explain how, in the last 30+ years, the question of determinacy was formalized. There are many parameters here: obviously one needs to choose the query language of the available views and the query language of the query itself. But – surprisingly – there is also some choice regarding what the word "to compute" actually means in this context.

Then I concentrate on certain variants of the decision problem of determinacy (for each choice of parameters there is one such problem) and explain how I understand the mechanisms rendering such variants of determinacy decidable or undecidable. This is on a rather informal level. No really new theorems are presented, but I show some improvements of existing theorems and also simplified proofs of some of the earlier results.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases database theory, query, view, determinacy

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.2

Category Invited Talk

Funding *Jerzy Marcinkowski*: Supported by the Polish National Science Centre (NCN) grant 2016/23/B/ST6/01438.

Acknowledgements The author would not have been in a position to write this invited paper had he not been first blessed with the opportunity to work with extraordinary students: Tomasz Gogacz, Grzegorz Głuch and Piotr Ostropolski-Nalewaja.

1 Introduction (1)

"Assume that a set of derived relations is available in a stored form. Given a query, can it be computed from the derived relations and, if so, how?" is the first sentence of [10], the oldest paper I know addressing the Query Determinacy Problem (QDP). On the very informal level this first sentence still does very good job explaining the idea of QDP. But in order to be really able to work on it, to formulate theorems and to try to prove them, we need to be a bit more precise.

And, as it turns out, there is a huge number of ways in which one can be more precise, each way leading to one variant of QDP. One can choose (1) between various query languages defining the stored views and (2) defining the given query, (3) between information-theoretic notion of determinacy (this paper) and various rewriting languages (not this paper), (4) between considering only finite database instances (QDP^f), or any relational structures, finite or infinite (QDP^∞), and finally (5) between *exact* (QDP_e) and *sound* (QDP_s) semantics. Let us first stick to the exact semantics, which is (I think) simpler to understand (sound semantics will be briefly discussed in Section 7).

Each variant of QDP is a decision problem. The instance is always a set of queries $\mathcal{V} = \{V_1, \ldots V_k\}$ in query language $\mathcal{L}_V$ and another query Q in some query language $\mathcal{L}_Q$. We are asked whether it is true that *for each database instance* D the query $Q(D)$ can be computed by a *device* from some set $\mathcal{C}$, from the set of views $V_1(D)$, $V_2(D) \ldots$, $V_k(D)$.

We are going to use notation $QDP_e^{-}(\mathcal{L}_V, \mathcal{L}_Q, \_)$ to denote the variant of QDP, under exact semantics, with arguments reflecting our choice of parameters. So, for example, $QDP_e^f(UCQ, UCQ, FO)$ will be the version of QDP where views are defined by queries $V_1, \ldots V_k$ being unions of conjunctive queries, query Q is also a union of conjunctive queries, only finite instances are allowed, and we ask whether there is a First Order rewriting of a given query Q, computing, regardless of D, the query $Q(D)$, when applied to the views $V_1(D)$, $V_2(D) \ldots$, $V_k(D)$. As another example $QDP_e^\infty(CQ, CQ)$ will denote the problem of deciding, for a set of conjunctive queries $V_1, \ldots V_k$, and another conjunctive query Q, whether $Q(D)$ can **in any way**[1] be computed from $V_1(D)$, $V_2(D) \ldots$, $V_k(D)$ for any (possibly infinite) structure D. Notice that we do not assume that it needs to be computed by an algorithm, or a Turing machine[2]. The only thing that matters is that the complete information about $Q(D)$ is already in $V_1(D)$, $V_2(D) \ldots$, $V_k(D)$. In other words, and more precisely:

▶ **Definition 1.** *We say that a set $\mathcal{V} = \{V_1,\ V_2 \ldots, V_k\}$ of queries, written in some query language $\mathcal{L}_V$ (finitely)* ***determine*** *query Q, written in some query language $\mathcal{L}_Q$, denoted as $[\mathcal{V}, Q] \in QDP_e^\infty(\mathcal{L}_V, \mathcal{L}_Q)$ (resp. $QDP_e^f(\mathcal{L}_V, \mathcal{L}_Q)$) if for each two database instances (resp. finite database instances) D, D' such that for each $1 \leq i \leq k$ there is $V_i(D) = V_i(D')$ there is also $Q(D) = Q(D')$.*

A pair of structures D, D' as above will be sometimes called a counterexample for determinacy.

In this paper we will mainly be interested in complexity (or rather decidability) of problems of the form $QDP_e^{-}(\_, \_)$ which we think of as of proper determinacy problems, rather than “rewriting” problems (which require different methods). Among them, we mainly concentrate on problems of the form $QDP_e^\infty(\_, \_)$. But, at least when talking about the negative results, this is only for convenience – analysis for the unrestricted case is simpler and more intuitive than in the finite case, but does not require, as far as we know, significantly different tools than the respective $QDP_e^f(\_, \_)$ variants[3].

2 Preliminaries

We mainly try to use standard notions and notations of relational database theory. In particular, for a query Ψ, with k free variables, and for a database instance D, the notation $\Psi(D)$ denotes the k-ary relation resulting from applying the query Ψ to D. For a conjunctive

[1] The distinction between query rewritability and query determinacy in the information-theoretic sense, as considered in this paper, was not really fully realized before the end of 1990s. The earliest paper I know which makes a clear distinction is [3]. Let me quote it here: *Unfortunately, many of* [the previous papers on view-based query answering] *do not distinguish between view-based query answering and view-based query rewriting, and give raise to a sort of confusion between the two notions. [...] So, in spite of the large amount of work on the subject, the relationship between view-based query rewriting and view-based query answering is not completely clarified yet.*

[2] There are only two arguments now – since there is no rewriting now there is also no need to specify the rewriting language.

[3] In order to translate an undecidability proof for some $QDP_e^\infty(\mathcal{L}_V, \mathcal{L}_Q)$ to a proof of the same result for $QDP_e^f(\mathcal{L}_V, \mathcal{L}_Q)$ one usually needs to recall what recursively inseparable sets are.

query ψ by the frozen body of ψ, denoted as $[\psi]$, we mean the relational structure[4] (unique up to isomorphism) whose elements[5] are the variables of ψ and whose relational atoms are the atomic formulas of ψ.

A tuple generating dependency (TGD) is a formula of the form: $\forall \bar{x}, \bar{y}\ (\phi(\bar{x}, \bar{y}) \Rightarrow \exists \bar{z}\ \psi(\bar{x}, \bar{z}))$, where ϕ and ψ are conjunctions of atomic formulas (which means that our TGDs are "multi-head"). Formula ϕ is called the *body* of the TGD and ψ is its *head*. The universal quantifier in front of the formula is always omitted and implicit.

Another notion we often use is the one of disjunctive TGD. They are like TGDs except that ψ is now of the form $\psi = \bigvee_{1 \leq i \leq k} \psi_i(\bar{x}, \bar{z}_i)$, for some k, where each ψ_i is a conjunction of atomic formulas and each $\bar{z}_i$ is a subtuple of $\bar{z}$ (in this way we are trying to say that there is the same set $\bar{x}$ of free variables in each of the disjuncts of ψ, but the sets of quantified variables may differ). An analogous condition holds for ϕ.

2.1 The zoo of query languages

Let us now define the classes of languages which occur as $\mathcal{L}_Q$ or as $\mathcal{L}_V$.

We have two kinds of them. First kind are the fundamental languages from the relational databases tradition. CQ is the language of conjunctive queries, being conjunctions of atomic formulas preceded by some existential quantifiers. UCQ are unions of conjunctive queries, which means they are formulas of the form $\psi = \bigvee_{1 \leq i \leq k} \exists \bar{z}_i\ \psi_i(\bar{x}, \bar{z}_i)$, for some k, where each ψ_i is a conjunction of atomic formulas (notice that we again assume that free variables are the same in each disjunct).

We will also consider languages of unary (monadic) CQs and unary (monadic) $UCQs$ (notations mCQ and $mUCQ$ will be used). These are queries with only one free variable (so that $\Psi(D)$ is a unary relation, that is a set of elements of D).

DL queries are ones defined by Datalog programs. We only consider boolean DL queries, which means that there is always one special arity zero predicate *goal* in each program, and, for a program P and a database instance D we think that $P(D)$ is true if P proves *goal* in D. Monadic Datalog (mDL) are DL queries defined by a program with all the IDB predicates (this means, the predicates that occur in a head of any rule) being unary.

Then there is the second sort of query languages, coming from the tradition of graph databases. Path Queries (PQ) are conjunctive queries of the form:

$$\exists z_1, z_2, z_{m-1}\ E_1(x, z_1) \wedge E_2(z_1, z_2) \wedge \ldots E_m(z_{m-1}, y)$$

A Path Query always has two free variables. It says, about two elements of the structure, that there is a path between them, labelled with the particular sequence (word) of labels (predicate names). We identify such query with this word, for example the above PQ is usually denoted as simply $E_1 E_2 \ldots E_m$.

A union of Path Queries (UPQ) is a UCQ whose CQs are path queries. Hence it also has two free variables, and it says, about x and y, for some finite set of words W, that there is a path, from x to y, labelled with a $\mathbf{w} \in W$.

Finally, a Regular Path Query (RPQ) is like a union of Path Queries, but the set W no longer is assumed to be finite: it can be any regular set over the language of labels. In a sense RPQ is, for the graph databases tradition, what DL is for the relational databases one.

[4] The terms "relational structure" and "database instance" mean the same thing for us.

[5] Relational structures have elements (vertices). We never call them "constants" or "nulls": the term "constant" is reserved (according to the tradition of mathematical logic) for the constants of the language.

2.2 Unfoldings of Datalog programs

We often want to build a minimal structure satisfying some query Ψ. The word "minimal" means minimal from the point of view of positive information. For such minimal structure M and for any other structure D with $D \models \Psi$ we would like to be sure that a homomorphism exists from M to D. This is easy for CQs – the frozen body of Ψ is always such a minimal structure. If Ψ is a UCQ we no longer have **the** minimal M, but we have a finite number $M_1, M_2 \ldots M_k$ of structures (frozen bodies of the disjuncts of Ψ) such that if $D \models \Psi$ then we are sure that there exists a homomorphism from one of the M_i to D.

But how about Datalog queries? Here we no longer have a finite set of such minimal structures (unless the program in question is bounded). The minimal structures are now the unfoldings of the program – all the possible ways of proving *goal*. It will be helpful at some point to see that:

▶ **Exercise 2.** *For each Datalog program P there exists a constant c such that all the unfoldings of P have tree width bounded by c.*

3 Introduction (2)

3.1 Examples

Let us see an example or two.

Suppose Q is a Path Query $BACA$ (see Section 2.1 for explanation) and:

$$\mathcal{V}_1 = \{BAC, ACA, AC\} \qquad \mathcal{V}_2 = \{BAC, ACA, CA\}$$

Then $[\mathcal{V}_1, Q] \in QDP_e^\infty(PQ, PQ)$ while $[\mathcal{V}_2, Q]$ is a negative instance of $QDP_e^\infty(PQ, PQ)$ – there is no determinacy.

In order to prove the second claim (the one about $[\mathcal{V}_2, Q]$) it is enough to take:

$$D_1 = \{B(a,b), A(b,c), C(c,d), A(d,e)\}$$

$$D_2 = \{B(a,b'), A(b',c'), C(c',d), A(b,c), C(c,d'), A(d',e)\}$$

Then $\mathcal{V}_2(D_1) = \mathcal{V}_2(D_2)$ (exercise!) and $D_1 \models Q$ but $D_2 \not\models Q$. This was easy. But how could we possibly prove the first claim?

3.2 What is this paper about?

Definition 1 is precise and simple. But it does not usually mean that for given $\mathcal{L}_V, \mathcal{L}_Q$ one can easily figure out if the problem $QDP_e^\infty(\mathcal{L}_V, \mathcal{L}_Q)$ (or $QDP_e^f(\mathcal{L}_V, \mathcal{L}_Q)$) is decidable or not.

The first obstacle is that it is not obvious at all how to work with this definition. How can we make sure that something is true for "each two database instances"?

To deal with this problem we ([8]) invented the notion of Green-Red Chase. The idea to explain determinacy in terms of Chase was not totally absent in some of the previous papers, including [11]. The Green-Red Chase is just a very little step forward. But, as we are going to explain, due to this little step we suddenly can see things we were unable to notice before.

Green-Red Chase will be introduced in Section 4. Then, in Section 5 we are going to show how the insight provided by Green-Red Chase can be very easily used to prove some positive results.

It was shown in [11] that $QDP_e^\infty(mCQ, CQ)$ is decidable. In Section 5.1 the Reader is going to see how this result can be significantly strengthened, and without using the complicated argument from [11]. We will show that, for example, $QDP_e^\infty(mUCQ, CQ)$ is decidable. And everything will be very simple, almost trivial.

Another known positive result is the one from [1] (later slightly improved by [12]). It is shown, in [A11], that $QDP_e^\infty(PQ, PQ)$ is decidable. In Section 5.2 we are going to build on the top of the technique from [1] to show a slightly stronger result, that $QDP_e^\infty(PQ, RPQ)$ is also decidable.

Both the techniques we present in Section 5 rely, apart from the insight given by the Green-Red Chase, on simple automata-theoretic arguments.

Then we are going to move to the negative results. It was proved in the paper [13] that $QDP_e^\infty(UCQ, UCQ)$ is undecidable. Then, in [6] we have shown undecidability of $QDP_e^\infty(RPQ, RPQ)$ (solving a problem left open in the series of papers on "loselessness" of Regular Path Queries, including [4]). In [7] we refined the technique from [6] showing that $QDP_e^\infty(UPQ, UPQ)$ is also undecidable (notice that this result is also strictly stronger than the aforementioned negative result from [13] about $QDP_e^\infty(UCQ, UCQ)$). As it turns out[6], the technique in [6] and [7] relies mainly on, as I call it here, the Cold/Hot Trick. Thanks to this trick one can easily encode a Turing machine computation inside the Green-Red Chase. All that is needed is disjunction, both in $\mathcal{L}_V$ and in $\mathcal{L}_Q$. In Section 6 we explain the trick and show how it can be used to prove undecidability of $QDP_e^\infty(UCQ, UCQ)$.

But how about query languages without disjunction, where the Cold/Hot Trick does not work? It turns out that proving lower bounds for variants of the form $QDP_e^\infty(CQ, \_)$ is quite hard. Indeed, the only negative result for a variant of this form known before 2015 was the one from [5] where undecidability of $QDP_e^\infty(CQ, DL)$ is proved. And, as we explain in Section 7, this result does not really, on the technical level, have much to do with the phenomenon of determinacy. It is just a simple consequence of undecidability of Datalog programs containment.

In Section 4.3 we reveal the reason behind this hardness, which is the Curse of Pâte Feuilletée[7]. Then, in Section 8 we show how to break the Curse of Pâte Feuilletée. This is the most complicated part of this paper and we are only able to explain some ideas of our proof technique from [8] and [9], where undecidability of $QDP_e^\infty(CQ, CQ)$ and of $QDP_e^f(CQ, CQ)$ is proven.

In the meantime, in Section 7, we report some work in progress [2] regarding the sound semantics, which means variants of the form $QDP_s^\infty(\_, \_)$. We explain how Green-Red Chase is also relevant in this case, and also how the simple automata-theoretic techniques from Section 5.1 can be helpful there. We also remark that the Cold/Hot Trick can be useful for the variants of $QDP_s^\infty(\_, \_)$ where disjunction is available, and that the Curse of Pâte Feuilletée seems to be in force for at least one variant of $QDP_s^\infty(\_, \_)$ (for which decidability of determinacy remains open).

4 The Green-Red Chase

The notion of Chase is one of the ubiquitous notions of database theory. Suppose we have a set $\mathcal{T}$ of TGDs, a database instance D and a conjunctive query Q. And, for some reason, we want to know whether $\mathcal{T}, D \models Q$, which means that Q is true in all superstructures D^+ of

[6] From the perspective, it seems to me, that when writing [6] and [7] we did not understood the technique well enough yet, and we did a poor job explaining what constitutes the core of the technique there and what is the implementation.

[7] This is how we called it with Tomek Gogacz, who was my student at that time, when we struggled, in the years 2012-13, to solve the problem of the decidability status of $QDP_e^\infty(CQ, CQ)$.

D satisfying all the TGDs in $\mathcal{T}$. Then we can construct $Chase(\mathcal{T}, D)$ and check whether $Chase(\mathcal{T}, D) \models Q$. If Q is satisfied in $Chase(\mathcal{T}, D)$ then it is satisfied in all structures D^+ as above[8]. Due to this property $Chase(\mathcal{T}, D)$ is called *a universal structure.*

How is this universal structure built? We start from[9] $Chase_0 = D$. Suppose that $Chase_i$ is defined. Then $Chase_{i+1}$ is defined by applying the following step, in parallel, to all possible TGDs $T \in \mathcal{T}$ and all tuples $\bar{c}$ of elements of $Chase_i$: suppose the body $\phi(\bar{c}, \bar{l})$ of T is satisfied in $Chase_i$ for some tuple $\bar{l}$ and that T postulates that there exists some tuple of elements $\bar{k}$ which, together with the tuple $\bar{c}$ of elements of $Chase_i$, satisfy the head $\psi(\bar{c}, \bar{k})$. Then we simply invent a tuple $\bar{k}$ of new elements, and add them to $Chase_i$ together with the atoms which occur in ψ. And – importantly – we do it **in the minimal way, from the point of view of the amount of positive knowledge**: the elements that are being added are all new, and they are never equal unless the TGD T explicitly requires them to be equal.

Finally, $Chase(\mathcal{T}, D) = \bigcup_{i \in \mathbb{N}} Chase_i$ turns out to be the universal structure.

We use the same idea in the context of Definition 1. Imagine we have an instance of the problem $QDP_e^\infty(\mathcal{L}_V, \mathcal{L}_Q)$. Such an instance, to recall, will be a pair $\mathcal{I}$ consisting of a set of $\mathcal{V}$ of queries and a query Q. Following the idea of Chase, in order to check whether there is determinacy, we should try to construct a "universal counterexample" – a pair of database instances" D_G (as *green*) and D_R (as *red*) such that ($\clubsuit_1$) $V(D_G) = V(D_R)$ for each $V \in \mathcal{V}$, that[10] ($\clubsuit_2$) $D_G \models Q$, and that the two conditions:

$$(\spadesuit_1)\ D_R \models Q \qquad \text{and} \qquad (\spadesuit_2)\ \mathcal{I} \in QDP_e^\infty(\mathcal{L}_V, \mathcal{L}_Q)$$

are equivalent, which means that if there exists any counterexample for determinacy for this instance then D_G and D_R are such an example.

4.1 Green and red structures and queries

We prefer however (and this is exactly the idea of the Green-Red Chase from [GM15]) to, instead of constructing two structures D_G and D_R, over some signature Σ, construct a single structure over a new signature $\Sigma_G \cup \Sigma_R$ consisting of colored (green and red) versions of the predicates from Σ.

To speak about objects over this new signature it will be convenient to have two operators: G and R, painting any object over Σ green or red. So, for example, for a query Φ over Σ we will have its red version $R(\Phi)$, over Σ_R. Another operator we will sometimes need is daltonization (denoted $dalt()$). It takes green or red objects (over the signature $\Sigma_G \cup \Sigma_R$) and returns the same objects with colors removed (over Σ).

Now, instead of producing two structures over the signature Σ of $\mathcal{I}$ we will construct one structure D_{GR} over $\Sigma_G \cup \Sigma_R$ such that for each $V \in \mathcal{V}$ there will be ($\clubsuit_1$) $R(V)(D_{\mathrm{GR}}) = G(V)(D_{\mathrm{GR}})$, such that ($\clubsuit_2$) $D_{\mathrm{GR}} \models G(Q)$ and that the conditions:

$$(\spadesuit_1)\ D_{\mathrm{GR}} \models R(Q) \qquad \text{and} \qquad (\spadesuit_2)\ \mathcal{I} \in QDP_e^\infty(\mathcal{L}_V, \mathcal{L}_Q)$$

are equivalent.

[8] The "only if" direction is of course trivially true.

[9] From now on we will skip the arguments of Chase: we will write simply $Chase_n$ instead of $Chase_n(\mathcal{T}, D)$. Unless we think this can lead to any confusion.

[10] As it turns out, one can restrict the attention to Q being a boolean query, by which we mean a query without free variables.

4.2 The Green-Red chase. The CQ case.

Now, to begin with, imagine the simplest case, that $\mathcal{L}_V = \mathcal{L}_Q = CQ$.

We want to construct D_{GR} satisfying $(\clubsuit_1)$ $R(V)(D_{GR}) = G(V)(D_{\text{GR}})$ for each $V \in \mathcal{V}$. Let V be $\exists \bar{x}\ \phi(\bar{x}, \bar{y})$, where ϕ is a conjunction of atoms. Then $(\clubsuit_1)$ is equivalent to the conjunction of two TGDs:

$$G(\phi)(\bar{x}, \bar{y}) \Rightarrow \exists \bar{x}'\ R(\phi)(\bar{x}', \bar{y}) \qquad R(\phi)(\bar{x}, \bar{y}) \Rightarrow \exists \bar{x}'\ G(\phi)(\bar{x}', \bar{y})$$

Let $\mathcal{T}_{\mathcal{V}}$ be the set of all TGDs generated in this way from the queries in $\mathcal{V}$.

Since we also want to have $D_{\text{GR}} \models G(Q)$, first take a *minimal, from the point of view of the amount of positive knowledge* structure which satisfies $G(Q)$, and see it as $Chase_0$. In the CQ case, which we now consider, there is just one such minimal structure: the frozen body of $G(Q)$. So $Chase_0 = G([Q])$.

And now it follows from all we know about the Chase, that $D_{\text{GR}} = Chase(\mathcal{T}_{\mathcal{V}}, G([Q]))$ **is indeed the universal structure** we were looking for.

4.3 Discussion and the Curse of Pâte Feuilletée

Let us go back to the instance $[\mathcal{V}_1, Q]$ from Section 3.1. One of the views from $\mathcal{V}_1$, is the conjunctive query BAC, or $\exists z_1, z_2\ B(x, z_1), A(z_1, z_2), C(z_2, y)$. Then the two Green-Red TGDs generated by BAC are:

$$(\heartsuit_{gr})\ G(B)(x, z_1), G(A)(z_1, z_2), G(C)(z_2, y) \Rightarrow \exists z_1', z_2'\ R(B)(x, z_1'), R(A)(z_1', z_2'), R(C)(z_2', y)$$

and:

$$(\heartsuit_{rg})\ R(B)(x, z_1), R(A)(z_1, z_2), R(C)(z_2, y) \Rightarrow \exists z_1', z_2'\ G(B)(x, z_1'), G(A)(z_1', z_2'), G(C)(z_2', y)$$

Imagine how $(\heartsuit_{gr})$ is applied. Suppose we have some $Chase_{2k}$ already constructed[11] and there are some a_1, a_2, a_3 and a_4 there, which form a green BAC-path in this $Chase_{2k}$ (or, in other words, we have a green copy of the frozen body of BAC). The TGD $(\heartsuit_{gr})$ tells us that there also should also be such red BAC-path, from a_1 to a_4. If there is no such path in $Chase_{2k}$ then a new copy $R(B)(a_1', a_2'), R(A)(a_2', a_3'), R(C)(a_3', a_4')$ of the frozen body of $R(BAC)$ is created, it is added to $Chase_{2k}$, with a_1' identified with a_1 in the new structure and a_4' identified with a_4. The vertices a_2' and a_3' are not identified with anything in the old structure – they are new in $Chase_{2k+1}$.

An important take away from the example is that when constructing $Chase_{n+1}$ from $Chase_n$ we produce many copies of the (colored versions) of the frozen bodies of queries in $\mathcal{V}$ and join them with $Chase_n$ by identifying the elements that relate to the free variables of the respective V (like the x and y in the example) with elements of $Chase_n$. The elements that relate to the quantified variables of the respective V are the "new" elements of $Chase_{n+1}$.

Back to the $[\mathcal{V}_1, Q]$ from Section 3.1. Now we can prove that indeed $[\mathcal{V}_1, Q] \in QDP_e^\infty(PQ, PQ)$. Let us run the Green-Red chase. There will be[12] $Chase_0 = \{G(B)(a, b), G(A)(b, c), G(C)(c, d), G(A)(d, e)\}$ for some elements a, b, c, d, e. Then, $Chase_1 = Chase_0 \cup \{R(B)(a, b_1), R(A)(b_1, c_1), R(C)(c_1, d), R(A)(b, c_2), R(C)(c_2, d), R(A)(b, c_3), R(C)(c_3, d_3)\}$

There will be also, among some other atoms: $G(A)(b_1, c_4), G(C)(c_4, d)$ in $Chase_2$ and $R(A)(b_1, c_5), R(C)(c_5, d_5)$ and $R(A)(d_5, e)$ in $Chase_3$. But the last three atoms, together with $R(B)(a, b_1)$ form $R([Q])$ and, by universality of the Green-Red chase we get that $[\mathcal{V}_1, Q] \in QDP_e^\infty(PQ, PQ)$.

[11] Why $2k$? Wait, Observation 3 is coming.
[12] The Reader is invited to run the chase herself, to make sure that what I write here makes sense.

▶ **Observation 3.** *When constructing $Chase_{2k}$ only green atoms are added. When constructing $Chase_{2k+1}$ only red atoms are added.*

Proof. Induction. Clearly, since there is nothing red in $Chase_0$ only red atoms will be added when constructing $Chase_1$. For the induction step, since nothing red (resp. green) was added while constructing $Chase_{2k}$ (resp. $Chase_{2k+1}$), all the TGDs of the form $(\heartsuit_{rg})$ (resp. $(\heartsuit_{gr})$) TGDs are satisfied in $Chase_{2k}$ (resp. $Chase_{2k+1}$). ◀

For the following Observation recall that we still assume that $\mathcal{L}_V = CQ$.

▶ **Observation 4.** *For each $n \in \mathbb{N}$ there exists a homomorphism h_n from $dalt(Chase_n)$ to $dalt(Chase_0)$, with $h_n \subseteq h_{n+1}$.*

Similar observation, in the context of the chase with two separate structures rather than one green-red structure can be found in [13].

Proof. Induction. Clearly, h_0 is the identity. For the induction step, in order to keep the notations light, we will use the above example. Suppose some elements of $Chase_{n+1}$ were added by an application of the $(\spadesuit_{gr})$, as in the example. Then define $h_{n+1}(a'_2) = h_n(a_2)$ and $h_{n+1}(a'_3) = h_n(a_3)$. For any element a of $Chase_n$ define $h_{n+1}(a) = h_n(a)$. ◀

It immediately follows from the observation that $\bigcup_{n\in\mathbb{N}} h_n$ is a homomorphism from $\bigcup_{n\in\mathbb{N}} dalt(Chase_n)$ to $dalt(Chase_0)$.

The Curse of Pâte Feuilletée. When trying to prove a lower bound for a problem of the form $QDP_e^{\infty}(CQ, \_)$ one soon realizes that Observation 4 is the main obstacle. How can we possibly encode anything in the structure of $Chase$ if all we get there is basically $Chase_0$ repeated infinitely many times? Each time something new is added to the structure it is merely a (re-colored) version of something that already was there. What we get is a pâte feuilletée, with infinite number of almost identical layers, each of them being a copy of $G([Q])$ and nothing but air between them.

4.4 The Green-Red Chase. The non-CQ case.

Now let $\mathcal{L}_V = \mathcal{L}_Q = UCQ$. Imagine we have an instance $[\mathcal{V}, Q]$ of $QDP_e^{\infty}(UCQ, UCQ)$ and recall that we assume that for each $V \in \mathcal{V}$ all the disjuncts of V have the same free variables.

Again, we want to have $D_{\mathrm{GR}} \models G(Q)$. But no longer **the** *minimal, from the point of view of the amount of positive knowledge* structure which satisfies $G(Q)$ exists. There are several such minimal structures, namely the (green versions) of the frozen bodies of the CQs being the disjuncts of Q.

We need to choose one of these disjuncts, call it Q', and put $Chase_0 = G([Q'])$.

But who is this "we" here? There was no need to ask this question in Section 4.2 as the Chase there was a normal, deterministic chase, leading to the same result regardless of who performs it. But now there is a choice, so who makes it and with what goal on mind?

In [6] and [7] we do not use the word "Chase" at all in this context. Instead, we consider a game, with a single player, called the Fugitive, and we call this version of Chase "the game of Escape". The goal of this player is to show that the QDP instance in question is a negative one – there is no determinacy. In the process of his Escape the Fugitive tries to construct a green-red structure D_{GR} for which it holds that $(\clubsuit_1)$ $G(\mathcal{V})(D_{\mathrm{GR}}) = R(\mathcal{V})(D_{\mathrm{GR}})$ and $(\clubsuit_2)$ $D_{\mathrm{GR}} \models G(Q)$ but $D_{\mathrm{GR}} \not\models R(Q)$.

So it is the Fugitive who begins the game choosing some disjunct Q' of Q, and defining $Chase_0$ as $G([Q'])$. At this point he is already sure that ($\clubsuit_2$) is satisfied. But how about ($\clubsuit_1$)?

Like in Section 4.2 we need to ask what it means, for a structure D_{GR}, that ($\clubsuit_1^V$) $R(V)(D_{\text{GR}}) = G(V)(D_{\text{GR}})$ for a $V \in \mathcal{V}$ (which is a UCQ now). It is easy to see that now ($\clubsuit_1^V$) is equivalent to the conjunction of two disjunctive TGDs:

$(\heartsuit_{gr})$ $\bigvee_{0\leq i<k} G(\phi_i)(\bar{x}'_i, \bar{y}) \Rightarrow \bigvee_{0\leq i<k} \exists \bar{x}'_i\, R(\phi_i)(\bar{x}_i, \bar{y})$

' $(\heartsuit_{rg})$ $\bigvee_{0\leq i<k} R(\phi_i)(\bar{x}'_i, \bar{y}) \Rightarrow \bigvee_{0\leq i<k} \exists \bar{x}'_i\, G(\phi_i)(\bar{x}_i, \bar{y})$

where k is the number of disjuncts in V, the formula ϕ_i is the quantifier-free part of the i-th disjunct, and $\bar{y}$ are the free variables of (each conjunct of) V.

Now suppose the body of $(\heartsuit_{gr})$ is satisfied in some[13] $Chase_{2n}$. Then $Chase_{2n} \models G(\phi_i)(\bar{a}, \bar{b})$ for some ϕ_i and some tuple $\bar{a}, \bar{b}$ of elements of $Chase_{2n}$. An application of our disjunctive TGD will produce a tuple $\bar{a}'$ in $Chase_{2n+1}$ such that $Chase_{2n+1} \models R(\phi_{i'})(\bar{a}', \bar{b})$. But there is no reason for i' to equal i, and it is the job of the Fugitive to choose the i' which suits him best (remember, his goal is to reach the fixpoint without satisfying $R(Q)$).

Now, a lemma in [6] and [7] is that such a game indeed characterizes determinacy: the Fugitive has a winning strategy if and only if $[\mathcal{V}, Q] \not\in QDP_e^\infty(UCQ, UCQ)$. The proof of the lemma goes in the footsteps of the standard proof of universality of Chase, which means that "induction" and "homomorphism" are the keywords.

5 Some applications on the positive side

5.1 Unary queries

It is proven in [11] (it is not a long proof but the argument is not so easy to understand) that $QDP_e^\infty(mCQ, CQ, CQ)$ equals to $QDP_e^\infty(mCQ, CQ)$. In other words, if a set of unary conjunctive queries determines a CQ, then there exists a rewriting, which itself is a conjunctive query. Then a corollary follows in [11] that $QDP_e^\infty(mCQ, CQ)$ is decidable.

But it seems to me that a stronger decidability result, not provable in any obvious way by the aforementioned rewriting argument, can be easily proved using the insight given by the Green-Red Chase[14]. Let $\mathcal{L}_V$ be any query language, consisting of unary queries, and such that (*) all the minimal bodies of queries are of bounded tree width (so, for example $\mathcal{L}_V$ may contain unions of unary conjunctive queries and/or Monadic DL queries). And let also $\mathcal{L}_Q$ be any[15] query language satisfying (*). Then $QDP_e^\infty(\mathcal{L}_V, \mathcal{L}_Q)$ is decidable.

Let us now explain the proof idea using the example of $QDP_e^\infty(mUCQ, CQ)$. Suppose that we have given a set $\mathcal{V}$ of mUCQs and a CQ Q. Then (and this is the sentence which summarizes the whole proof) any structure which the Fugitive can construct as his $Chase(\mathcal{T}_\mathcal{V}, G([Q]))$ is a structure having the tree width bounded by some k: indeed, the bags of the tree decomposition in question are the (red or green versions of) the frozen bodies of the conjunctive queries being the disjuncts of the queries from $\mathcal{V}$. This is because such

[13] There are many structures now which can be built by the Fugitive as his $Chase_m$, and as his final $Chase$. But – hoping this will not lead to additional confusion – we call each of them $Chase_m$ or $Chase$.

[14] While reading this subsection the Reader may notice that the idea of having, instead of two structures over Σ, a single structure over a green-red signature is of critical importance here.

[15] If it was a regular paper we should be slightly more formal here: condition (*) should also require that the set of minimal bodies of queries is *regular*. This is of course satisfied for mUCQs and for mDL.

a frozen body, when being added to *Chase* at some point, only connects to the current structure via the elements being substituted for its free variables, which in this case is a single element[16].

Now one can construct an automaton A_1 which, for a given structure $\mathcal{A}$, of tree width bounded by k, decides whether $\mathcal{A} \models \mathcal{T}_\mathcal{V}$ and $\mathcal{A} \models G([Q])$. It will accept any structure that can be built as $Chase(\mathcal{T}_\mathcal{V}, G([Q]))$ and probably also many other structures which, while satisfying $\mathcal{T}_\mathcal{V}$ and containing $G([Q])$, cannot be produced by a Green-Red Chase procedure.

One can also construct another automaton A_2 which, on any structure with the tree width bounded by k, decides whether $R([Q])$ is contained in this structure. Then we apply the standard automata-theoretic procedure to check whether there exists a structure accepted by A_1 but not by A_2.

5.2 Slightly beyond Path Queries

It is proved in [1] that $QDP_e^\infty(PQ, PQ)$ is decidable. And the decision algorithm, while quite simple, gives a really very nice insight into PQ determinacy. In our language the algorithm can be expressed as follows.

Given a path query Q and a set $\mathcal{V}$ of path queries first construct $Chase_0$, which, as we know, will be $G([Q])$. This structure is a green path. Let s be the starting point of this path and let t be its endpoint.

Then construct $Chase_1$. This means that a set of new red paths connecting some pairs of elements of $Chase_0$ will be added. Consider now the graph being the red part of $Chase_1$ (that is all the edges that are in $Chase_1$ but not in $Chase_0$). It is not terribly hard to prove (and is left for the Reader as a rather non-trivial exercise) that $[\mathcal{V}, Q] \in QDP_e^\infty(PQ, PQ)$ if and only if s and t are in the same connected component[17] of this graph.

Notice that this really works for the $[\mathcal{V}_1, Q]$ from Section 3.1: the a and e there indeed are connected via the red edges of $Chase_1$ (one needs to make 3 steps going “forward”, then two steps “backward” and then again 3 steps “forward”).

Using the above “connectivity” criterion one[18] can show that:

▶ **Theorem 5.** *$QDP_e^\infty(PQ, RPQ)$ is decidable.*

For the proof of the theorem suppose Q and $\mathcal{V}$ are given. Q is a query defined as a union of some regular set $\mathcal{R}$ of path queries, and each of the queries in $\mathcal{V}$ is a path query, so there is no disjunction there. This means that the only choice the Fugitive has here is when he constructs $Chase_0$. He can take, as $Chase_0$, any green path $G([\mathbf{w}])$, from some s to t, for some $\mathbf{w} \in \mathcal{R}$

He is deemed to lose if for each such choice of $\mathbf{w}$ the vertices s and t will be in the same connected component of the red part of $Chase_1(\mathcal{T}_\mathcal{V}, G([\mathbf{w}]))$. He wins (and so $[\mathcal{V}, Q] \notin QDP_e^\infty(PQ, RPQ)$) if he can find a $\mathbf{w}$ for which s and t will be in two different connected components.

[16] As I learned from a discussion with Sebastian Rudolph, this argument holds true even for non-unary queries if all the free variables always occur in a single atom of each disjunct of V. The general picture is that what we actually do here is deciding query entailment for the theory $\mathcal{T}_\mathcal{V}$, and this is decidable for sets of TGDs which are frontier guarded. One can also notice here that for positive results for the respective $QDP_e^f(\mathcal{L}_V, \mathcal{L}_Q)$ variants known theorems regarding Finite Controllability for certain sets of TGDs can be applied.

[17] We think of an undirected connected component of a directed graph here.

[18] Theorem 5 comes from the unpublished master’s thesis by my student Grzegorz Głuch.

How can we decide whether such $\mathbf{w}$ exists? First of all recall how $Chase_1(\mathcal{T}_\mathcal{V}, G([\mathbf{w}]))$ is constructed: for two vertices a, b of $G([w])$ connected with some word $G(\mathbf{v})$ (which is a sub-word of $\mathbf{w}$) a new red path from a to b, labelled with $R(\mathbf{v})$ is created in $Chase_1$ if and only if $\mathbf{v} \in \mathcal{V}$. We imagine we have a green straight path from s to t, with $\mathbf{w}$ being the sequence of the labels of its edges (this is $Chase_0$), and red arcs over this green straight path joining each two vertices which are connected with a path being some green $\mathbf{v} \in \mathcal{V}$.

It is now easy to construct a two-way non-deterministic finite automaton which will accept $\mathbf{w}$ if and only if s and t are in the same connected component of the red part of $Chase_1$: the automaton starts its run in the head in s, and then, at each stage of the run[19] it first guesses whether it should now walk down some red arc (towards the t) or up (back, towards s), and which $\mathbf{v} \in \mathcal{V}$ is the label of the arc it now takes. And then it just walks down or up $G([\mathbf{v}])$ imagining it moves down or up a red arc (and checking whether it indeed is labelled with $\mathbf{v}$). It accepts when t is reached after completing some number of stages.

Notice that in the case of of the instance $[\mathcal{V}_1, Q]$ from Section 3.1 our two-way automaton will first go towards t (the $\mathbf{v}$ will be BAC), then back towards s (the $\mathbf{v}$ will be AC), and finally it will reach t after the third stage where $\mathbf{v}$ will be ACA.

Now, this two-way nondeterministic finite automaton can be translated into a normal DFA $A_\mathcal{V}$. We also have a DFA $A_\mathcal{R}$, deciding the language $\mathcal{R}$. All we need to do to see whether the Fugitive is deemed to lose is to decide, using handbook methods, the containment of languages for $A_\mathcal{R}$ and $A_\mathcal{V}$.

▶ **Exercise 6.** *Why doesn't it prove that also $QDP_e^\infty(RPQ, RPQ)$ is decidable? Exactly the same automata trick would work in this case.*

Notice that, instead of building this two-way automaton we could just use the fact that (since the red arcs are short and local) the structure $Chase_1$ is of bounded tree width (and the bound does not depend on $\mathbf{w}$).

6 How disjunction leads to undecidability. $QDP_e^\infty(UCQ, UCQ)$.

If disjunction is available in $\mathcal{L}_Q$ and $\mathcal{L}_V$ then the Curse of Pâte Feuilletée is not in force: the Fugitive **can**, while executing the Green-Red Chase, add to the structure something that is not just a copy of $Chase_0$. For example, suppose that some query $V \in \mathcal{V}$ equals $\exists \bar{x}\ \phi(y, z, \bar{x}) \wedge \phi'(y, z, \bar{x})$, for some CQs ϕ and ϕ', and that $R(V)$ is satisfied in some $Chase_i$, because there is $Chase_i \models R(\phi)(a, b)$ for some elements a and b. Suppose however that $G(V)(a, b)$ is not satisfied in $Chase_i$. Then the Fugitive must satisfy $G(V)$ in $Chase_{i+1}$ and he can do it by adding to $Chase_{i+1}$ a new copy of $G([\phi])$, connected to $Chase_i$ via a and b ("a re-colored copy of something we already saw") or a new copy of $G([\phi'])$ (connected in the same way). So he **can** produce something new. But can we force him to? This is what the Cold/Hot trick is about, which we are going to present in this Section.

The example we are going use is $QDP_e^\infty(UCQ, UCQ)$. We will show that the problem is undecidable. The result comes from [11], but the proof we present here is based on the ideas from [6], where we employed the cold/hot trick to show undecidability of $QDP_e^\infty(RPQ, RPQ)$ (and of $QDP_e^f(RPQ, RPQ)$) and from [7] where analogous results were shown for UPQ as both $\mathcal{L}_V$ and $\mathcal{L}_Q$. Clearly, both RPQ and UPQ support disjunction, and disjunction is all we need for the Cold/Hot trick to work.

[19] The run will consist of an unbounded number of stages, each of them comprising a bounded number of steps.

6.1 The Cold/Hot Trick

Imagine that our signature $\Sigma = \Sigma_C \dot{\cup} \Sigma_H$, so it is a disjoint union of *cold* and *hot* relation symbols[20]. This means that there can possibly be four kinds of atoms in the Green-Red Chase: red-warm, red-cold, green-warm and green-cold. The idea is to construct[21] Q and $\mathcal{V}$ in such a way that if the Fugitive chooses anything green-hot or red-cold then he loses the game immediately. In other words he will need to always make sure that the green part of the Chase is entirely cold and the red part is entirely hot.

Let *Lukewarm* be the set of all conjunctive queries of the form[22] $C(x,x') \wedge H(y,y')$ where – unsurprisingly – C is cold and H is hot. Our UCQ Q is:

$$\exists x, x', y, y' \quad \alpha_C(x,y) \vee \omega_H(x,y) \ \ \vee \bigvee_{\phi \in Lukewarm} \phi(x,x',y,y')$$

where α_C is a certain cold relation symbol and ω_H is a certain hot one. Our set of UCQs $\mathcal{V}$ is the disjoint union of $\mathcal{V}_{good}$ and $\mathcal{V}_{bad}$.

Each UCQ $V \in \mathcal{V}_{good}$ is of the form $V^H \vee V^C$ where V^H is a CQ being a conjunction of hot atoms and the V^C is a CQ being a conjunction of cold ones. We assume that $\alpha_C(x,y) \wedge \beta_H(x,y)$ is one of the queries in $\mathcal{V}_{good}$, for some hot β_H, and that this is the only place where α_C occurs in $\mathcal{V}_{good}$ The set of queries $\mathcal{V}_{good}$ is where the instance of some undecidable problem is going to be encoded. But we do not need to think of the details now.

At this point $\mathcal{V}_{bad}$ is more interesting, which is defined as the set containing all the queries from *Lukewarm* and the query $\omega_H(x,y)$. All the queries in $\mathcal{V}_{bad}$ are CQs.

Let, as always, $\mathcal{T}_{\mathcal{V}}$ be the set of all green-red disjunctive TGDs generated by $\mathcal{V}$.

Now let us analyze what the Fugitive choices for $Chase_0$ are:

▶ **Observation 7.** *The Fugitive must pick $G([\alpha_C])$ as $Chase_0$ or he will lose immediately.*

Proof. See what his other choices of minimal structures satisfying Q are. One is to pick $G([\omega_H])$. But notice that $G(\omega_H)(x,y) \Rightarrow R(\omega_H)(x,y)$ is one of the TGDs in $\mathcal{T}_{\mathcal{V}}$. Its body would be satisfied in $Chase_0$ so its head would need to be satisfied in $Chase_1$. So, it would be that $Chase_1 \models R(\omega_H)$ and hence $Chase_1 \models R(Q)$ and the game is over for the Fugitive.

The other choice would be to pick, as $Chase_0$, the structure $G([\phi])$ for some lukewarm query ϕ. But then he loses again, since in this case $G(\phi) \Rightarrow R(\phi)$ is one of the TGDs in $\mathcal{T}_{\mathcal{V}}$. ◀

Once we know that $Chase_0 = \{G(\alpha_C)(s,t)\}$ for some s, t, let the Fugitive build $Chase_1$:

▶ **Observation 8.** *$Chase_1 = Chase_0 \cup \{R(\beta_H)(s,t)\}$ or the Fugitive loses immediately.*

Proof. Recall that $\alpha_C(x,y) \wedge \beta_H(x,y)$ is one of the queries in $\mathcal{V}_{good}$. Hence the TGD:

$$G(\alpha_C)(x,y) \vee G(\beta_H)(x,y) \Rightarrow R(\alpha_C)(x,y) \vee R(\beta_H)(x,y)$$

is in $\mathcal{T}_{\mathcal{V}}$. So there either must be $R(\alpha_C)(s,t)$ in $Chase_1$ or $R(\beta_H)(s,t)$. But having $R(\alpha_C)(s,t)$ means that $Chase_1 \models R(Q)$ and loses the game for the Fugitive. ◀

[20] Let us also assume that all the relations in Σ are binary.

[21] This is an undecidability proof, so we construct Q and $\mathcal{V}$, depending on the instance of our favourite undecidable problem.

[22] Queries from the set *Lukewarm*, as defined here, are UCQs, but they are not UPQs (or RPQs). When implementing the idea of the Cold/Hot Trick in order to prove undecidability of $QDP^{\infty}_{e}(UPQ, UPQ)$ (or $QDP^{\infty}_{e}(RPQ, RPQ)$) one needs to invent something that would play the same role as *Lukewarm* but would also be expressible as path queries. This is easy in the RPQ case [6] but a bit complicated in the case of UPQ [7].

We are now sure that (unless the Fugitive is suicidal) there must be one green cold atom (namely, $G(\alpha_C)(s,t)$) and one red hot atom (namely, $G(\beta_H)(s,t)$) in $Chase_1$. Finally:

▶ **Observation 9.** *The Fugitive loses immediately if he ever produces a red cold atom or a green hot atom.*

Proof. Clearly, if a red cold atom was ever produced then it would satisfy, together with $R(\beta_H)(s,t)$, some $R(\phi)$ for a lukewarm query ϕ. And red lukewarm queries are forbidden (by Q) if the Fugitive wants to win[23]. Now notice that if a green hot atom was ever produced then it would satisfy, together with $G(\alpha_C)(s,t)$, some $G(\phi)$ for a lukewarm query ϕ. In this case, in the next step of Chase, $R(\phi)$ would also be satisfied. ◀

6.2 Now undecidability follows easily

To explain the remaining part of the proof let us use an example. Imagine $\mathcal{V}_{good}$ consists, apart from $\alpha_C(x,y) \wedge \beta_H(x,y)$, of the queries:

(i) $\beta_H(x,y) \vee C_\beta^{EF}(x,y)$
(ii) $C_\beta^{EF}(x,z) \vee \exists y\ E(x,y) \vee F(y,z)$
(iii) $F(x,y) \vee C_F^{EF}(x,y)$
(iv) $C_F^{EF}(x,z) \vee \exists y\ E(x,y) \vee F(y,z)$

Where E and F are hot and C_β^{EF} and $C_F^{EF}(x,y)$ are cold.

Now, let us recall that if the Fugitive wants to win, there must be $R(\beta)(s,t)$ in $Chase_1$. Then the red-green TGD generated by (i) forces the Fugitive to have, in $Chase_2$, either $G(\beta_H)(s,t)$ or $G(C_\beta^{EF})(s,t)$. But, by Observation 9, $G(\beta_H)(s,t)$ is forbidden, so there will be $G(C_\beta^{EF})(s,t)$ in $Chase_2$. Then, by analogous reasoning, there will be a new element s_1 in $Chase_3$, such that $Chase_3 \models R(E)(s,s_1), R(F)(s_1,t)$.

▶ **Exercise 10.** *There will be $Chase_5 \models R(E)(s,s_1), R(E)(s_1,s_2), R(F)(s_2,t)$ or the Fugitive will lose.*

In this way, using queries like (i)-(iv) we can easily encode the word problem for finitely represented semigroups: for each word $\mathbf{w}$ which is, in the given semigroup, equivalent to the word β_H we will finally get a red path in Chase, from s to t, labelled with the symbols of $\mathbf{w}$. Like in our example, where the semigroup is represented by $\beta_H = EF$ and $F = EF$ we soon forced the Fugitive to produce the path EEF. The Fugitive of course loses if at some point he is forced to produce atom ω_H. But it is a very well known undecidable problem whether, in a given finitely represented semigroup, there is any word $\mathbf{w}$ which contains ω_H and is equivalent to the word β_H.

7 Aside: determinacy under sound semantics

A variant of Query Determinacy Problem, studied in a number of papers in 1990s and early 2000s, and enjoying some new interest recently [2] is determinacy under sound semantics. It combines determinacy as formalized by Definition 1 with the observation that one never can be sure whether a queried database represents all the facts about some phenomenon,

[23] Rev. 3:16 (ESV) “So, because you are lukewarm, and neither hot nor cold, I will spit you out of my mouth.”

and thus all the views should be seen as correct but potentially incomplete. The precise definition is complicated[24], and we decided not to copy it here. And we actually do not need to copy it here, because we have another one, which happens to be equivalent[25]:

▶ **Definition 11.** *$QDP^\infty_s(\mathcal{L}_V, \mathcal{L}_Q)$ is the set of such instances $[\mathcal{V}, Q]$, with $Q \in \mathcal{L}_Q$, $\mathcal{V} \subseteq \mathcal{L}_V$ that regardless of the strategy of the Fugitive it holds that $Chase_1 \models R(Q)$.*

Notice that it is almost like our characterization of QDP^∞_e with the only difference, that for determinacy under "exact semantics" to hold, the Fugitive must be deemed to satisfy $R(Q)$ anywhere at any point of the Green-Red Chase, while for sound semantics this must happen already in $Chase_1$. This in particular means that QDP^∞_s is a (much) stronger notion than QDP^∞_e for the same parameters. It also follows directly from the definition that if languages $\mathcal{L}_Q$ and $\mathcal{L}_V$ have the property that each query has only finitely many "minimal structures" satisfying this query, then there are only finitely many possible structures $Chase_1$ and $QDP^\infty_s(\mathcal{L}_Q, \mathcal{L}_V)$ is trivially decidable (so, for example $QDP^\infty_s(UCQ, UCQ)$ is decidable).

▶ **Exercise 12** (CGLV02)**.** *Show that $QDP^\infty_s(RPQ, RPQ)$ is decidable.*

Hint: Like in Section 5.2 (compare to Exercise 6). But easier: a one-way finite automaton is sufficient here.

On the other hand, it is very easy to see that:

▶ **Observation 13.** *$QDP^\infty_s(CQ, DL)$ is undecidable, even if the CQs defining the views are projection-free.*

Proof. For the proof of the observation first recall that the containment of Datalog programs is undecidable. In other words it is undecidable whether, for two given programs ϕ and ψ, it holds that for each database instance D if $goal \in \phi(D)$ then $goal \in \psi(D)$. One can of course assume here that the sets of IDB predicates of ϕ and of ψ are disjoint (except of course for the arity zero predicate *goal* which is an IDB both in ϕ and in ψ). We also assume that both the programs are over some set Σ_0 of EDB predicates.

Now, let tr (like "trigger") be a new arity zero EDB predicate, and let ϕ' be a new Datalog program, which is exactly like ϕ but with the additional atom tr in the body of each rule. Which means that ϕ' behaves exactly like ϕ on the instances where tr is true, and does nothing at all on the instances where tr is false.

Let now our Q be the union of ψ and ϕ' and let our $\mathcal{V}$ contain a query $E(\bar{x})$ for each[26] predicate E in Σ_0. We claim that $[\mathcal{V}, Q] \in QDP^\infty_s(CQ, DL)$ if and only if ψ contains ϕ.

To see why the claim is true first suppose that $Chase_0$ is somehow constructed and notice that, due to the way $\mathcal{V}$ is defined, each atom of $Chase_1$ is either some $G(A)$ which was already in $Chase_0$ or $R(A)$. Or, in other words, $Chase_1$ is a union of $Chase_0$ and a red version of $Chase_0$. Except for tr: it may happen that $Chase_0 \models G(tr)$, but there is no way to have $R(tr)$ in $Chase_1$. This in particular means that no rule of $R(\phi')$ can ever be applied in $Chase_1$ and the only way to have $Chase_1 \models R(goal)$ is to prove the $R(goal)$ using $R(\psi)$.

Now let us go back one step and think of the ways in which the Fugitive can pick $Chase_0$. It is any minimal structure in which the program $G(Q)$ proves $G(goal)$, in other words it is an "unfolding" of the Datalog program $G(Q)$. There are infinitely many possible choices

[24] The notion of certain answers plays a role there.

[25] This equivalence, I understand, is proved in [2] (I didn't have the opportunity to see the paper yet). The main difficulty here is to understand the original definition of QDP_s (see for example [4]).

[26] Recall that $tr \notin \Sigma_0$.

for the Fugitive (if there is any recursion in Q). But the main choice he has, from the point of view of this argument, is whether he wants $G(Q)$ to be proven by means of the program $G(\psi)$ or $G(\phi')$.

In the first case, when $Chase_0$ is an unfolding of $G(\psi)$, it follows from the previous paragraph, that the red part of $Chase_1$ is an unfolding of $R(\psi)$, so $R(goal)$ can be proved there and the Fugitive loses. So the only way for him is to take, as $Chase_0$, an unfolding of $G(\phi')$.

Notice that in such case the red part of $Chase_1$ is any (chosen by the Fugitive) unfolding of $R(\phi)$ (not $R(\phi')$ but $R(\phi)$, as there is no $R(tr)$ in $Chase_1$!). Now, ψ contains ϕ if and only if in every such unfolding $R(goal)$ will be provable by $R(\psi)$. ◀

The variants of $QDP_s^\infty(\mathcal{L}_V, \mathcal{L}_Q)$ studied in [2] are ones with $\mathcal{L}_Q$ being Datalog programs syntactically restricted in such a way that containment is decidable, and the above very simple argument cannot be applied. One of the theorems they prove is that:

▶ **Theorem 14.** *$QDP_s^\infty(UCQ, mDL)$ is undecidable.*

The proof is by a clever application of the cold/hot trick.

On the positive side [2] shows that:

▶ **Theorem 15.** *$QDP_s^\infty(CQ, mDL)$ is decidable.*

I did not have the opportunity to see the proof from [2] so far, but I understand that it uses a tree automata argument, basically following the ideas presented in Section 5.1. Any unfolding of a monadic datalog program (and thus also every possible $Chase_0$) is a tree of bounded tree width. Then, for a Pâte Feuilletée reason nothing really new is added, only elements which were close to each other in $Chase_0$ can be close to each other in $Chase_1$, and the red part of $Chase_1$ is of bounded tree width too. And then, since the Datalog program in question is monadic, it can be decided by an automaton whether $R(goal)$ can be proven on such a bounded tree width $Chase_1$.

I am summarizing the proof of Theorem 15 in order to remark that things are a bit subtle here. For example, one could ask, why isn't $Chase_1$ a structure of bounded tree width even in the $QDP_s^\infty(UCQ, mDL)$ case? The answer is in a query:

$$(A(x) \wedge B(y)) \vee \exists z\ (E(x,z) \wedge E(y,z))$$

With such query in $\mathcal{V}$ two remote elements a, b of $Chase_0$, such that $Chase_0 \models G(A(a))$ and $Chase_0 \models G(B(b))$ can be connected[27], in $Chase_1$, via a new element e such that $Chase_1 \models R(E(a,e)), R(E(b,e))$.

Another subtlety regards the situation where we allow constants in the language. Normally, one would think, adding constants to the signature of a Datalog program should not change much: we can always replace $S(x,y,c)$ (where S is a relation symbol and c is a constant of the language) with an atom of a new predicate $S_{3=c}(x,y)$. Since there are finitely many constants, such an operation could possibly cost us in the terms of complexity of problems, but should not impact their decidability.

But imagine that for every variable x occurring somewhere in the body of any rule in Q, there is an atom $E(x,c)$ in this body. Which means that every element of $Chase_0$ will be connected, by $G(E)$, to c. It looks innocent. But imagine also that the query

[27] This works because one of the CQs in our UCQ is connected and the other is not. It seems to me that the version of $QDP_s^\infty(UCQ, mDL)$ where only such UCQs are allowed which have each of their CQs connected, is decidable, by the same proof which works for $QDP_s^\infty(CQ, mDL)$.

$\exists z\ E(x,z), E(y,z)$ is in $\mathcal{V}$. Then, by the rules of the Green-Red Chase, for each pair of elements a, b of $Chase_0$ there will be a new element s in $Chase_1$ such that $Chase_1 \models R(E)(a,s), R(E)(b,s)$.

In consequence $Chase_1$ will not have bounded tree width. Decidability of the version of $QDP_s^{\infty}(mDL, CQ)$ where constants are allowed in Datalog is, to the best of my knowledge, left open in [2].

8 Encoding. Spiders live here.

In Section 6 we explained how the Green-Red Chase can be used to simulate some computing device (which, in this case, was the word problem for semigroups). The mechanism we constructed there crucially required disjunction in the views from $\mathcal{V}$ (and in Q). Can anything similar be done without disjunction? In [8] and [9] Spiders are the answer.

8.1 Spiders and spider queries

Let $K \in \mathbb{N}$ be a fixed natural number. Full Spider is any structure $\mathcal{S}$ isomorphic to $\{H(a)\} \cup \{T_i(a,b_i), C_i(b_i,c_i) : 1 \le i \le K\} \cup \{2 \text{ more atoms to come}\}$. There can be of course Full Red Spider, which is $R(\mathcal{S})$, and Full Green Spider, $G(\mathcal{S})$.

But the workhorses of our construction are Lame Spiders. An i-Lame Red Spider $\mathcal{S}_R^i$, for $1 \le i \le K$, is a Full Red Spider with the atom $R(C_i)(b_i,c_i)$ replaced with $G(C_i)(b_i,c_i)$. An i-Lame Green Spider $\mathcal{S}_G^i$ is defined in an analogous way.

So a (Green or Red) Full Spider is a creature, either entirely red or entirely green, with a head (where the predicate H is) and K legs of length two. Legs are distinguishable and each of them comprises a thigh (predicate T) and a calf (predicate C). A Lame Spider, from the daltonized point of view, looks like a Full Spider, but has one calf of the opposite color.

To operate on Lame Spiders we define Spider Queries. For $1 \le i,j \le K$ we define the spider query $\Psi_{i,j}$ as a CQ (with variables of the form z_n as free variables):

$$\exists x, \bar{y} \quad H(x) \wedge T_i(x,z_i) \wedge T_j(x,z_j) \wedge \bigwedge_{k \neq i,j} (T_k(x,y_k) \wedge C_k(y_k,z_k) \wedge \ 2 \text{ more atoms to come}$$

So each Spider Query looks like a colorless Full Spider, but with two calves missing.

A pivotal example now. Let $K = 4, i = 2, j = 3$ and imagine there is an i-Lame Red Spider $\mathbb{S}$ somewhere[28] in $Chase_l$ (with the nodes $a, b_1, \ldots b_4, c_1 \ldots c_4$). Suppose also that $\Psi_{i,j}$ is in $\mathcal{V}$, meaning that the Red-Green TGD θ generated by $\Psi_{i,j}$ must be satisfied in Chase.

Let us convince ourselves that the body of θ matches with $\mathbb{S}$: there is an atom $R(H)(x)$ in the body of θ and there indeed is $R(H)(a)$ in $\mathbb{S}$. There is $R(T_1)(x,y_1) \wedge R(C_1)(y_1,z_1)$ in the body of θ and there indeed are $R(C_1)(a,b_1)$ and $R(C_1)(b_1,c_1)$ in $\mathbb{S}$ (and same if we took 4 instead of 1). Finally, there are $R(T_2)(x,z_2)$ and $R(T_3)(x,z_3)$ in the body of θ and $R(T_2)(a,b_2)$ and $R(T_3)(a,b_3)$ in $\mathbb{S}$.

▶ **Exercise 16** (Important in order to understand the idea). *Notice that there would be no such match if we considered the same $\mathbb{S}$ but $\Psi_{1,3}$ instead of $\Psi_{2,3}$.*

Back to our example. We have already noticed that the body of the TGD θ matches with $\mathbb{S}$. Now suppose the head of θ is not satisfied in $Chase_l$ (for this match). Let us see what will be produced when the red-green TGD generated by θ is applied. We know that new

[28] $\mathbb{S}$ is a copy, in some $Chase_l$, of $\mathcal{S}_R^i$

elements will be added in $Chase_{l+1}$ for all existentially quantified variables in θ: new a' will be produced, with $G(H)(a')$ and with $G(T_2)(a', b_2)$ and $G(T_3)(a', b_3)$. And also new b'_1 and b'_4 will be created, with $G(T_1)(a', b'_1)$, $G(C_1)(b'_1, c_1)$ and $G(T_4)(a', b'_4)$, $G(C_4)(b'_4, c_4)$.

Now see, the newly produced atoms, together with atoms $R(C_2)(b_2, c_2)$ and $G(C_1)(b_1, c_1)$ of $\mathbb{S}$ form a j-Lame Green Spider! We assumed that an i-Lame Red Spider is in $Chase_l$ and that $\Psi_{i,j}$ is in $\mathcal{V}$ and we proved that in this case there must be a j-Lame Green Spider in $Chase_{l+1}$! And of course the same holds true for the colors swapped. We proved (by example):

▶ **Observation 17** (The law of Spider Algebra). *If there is S^i_G somewhere in Chase, and $\Psi_{i,j}$ is in $\mathcal{V}$ then S^j_R is also in the same Chase (and the same for the colors swapped).*

And we almost[29] proved:

▶ **Observation 18.** *Let $\mathcal{G} = \langle\{1, 2, \ldots K\}, E\rangle$ be an undirected graph. Let $\mathcal{V}$ consist of all the queries $\Psi_{i,j}$ such that $[i, j] \in E$. Let $1 \leq k, k' \leq K$. Then the following two conditions are equivalent:*

- *k and k' are in the same connected component of $\mathcal{G}$;*
- *some (red or green) k-Lame Spider is in Chase if and only if some k'-Lame Spider is.*

Observation 18 shows how the mechanism of spiders (together with spider queries) can do some computing for us. But of course proving that determinacy is at least as difficult as graph reachability is not a big deal. And even if it was, the proof is not yet complete. The input/output procedures still remain to be a small problem: we need to have $G([Q])$ as $Chase_0$, which is entirely green. So where can we get our k-Lame Spider from? And we know that determinacy holds once $R([Q])$ occurs somewhere in $Chase$, but $R([Q])$ is entirely red, so it is not our k'-Lame Spider either. This small problem is solved in [8] by a minor modification of the definition of spider query. But let us skip it here.

8.2 High level view of spiders

In the proof of undecidability of $QDP^{\infty}_{e}(UCQ, UCQ)$, as presented in Section 6, we constructed our example $\mathcal{V}_{good} \subseteq \mathcal{V}$ in such a way, that (for example) whenever there were two vertices a, b in $Chase$ such that $R(EF)(a, b)$ was true in $Chase$ (since it was true in some $Chase_i$) then also $R(EEF)(a, b)$ was true in $Chase$ (since it was true in some $Chase_{i+2}$). This example was meant to convince the Reader that any instance of the word problem for finitely represented semigroups can be encoded.

If we wanted to be more precise we would probably have said that for a proof of undecidability **two tricks** are needed: **first**, we need to be able to ensure that whenever there are two vertices a, b in $Chase$ such that if $Chase \models R(AZ)(a, b)$ then also $Chase \models R(Z'A')(a, b)$ (this reflects a single operation of a Turing machine: think of Z as the machine head, in certain state). **Second**, we need to always be able to give this Turing machine more space, so we need to be able to ensure that whenever there are two vertices a, b in $Chase$ such that $Chase \models R(B)(a, b)$ then also $Chase \models R(BB)(a, b)$.

In [8] we show how to employ spiders for the two tricks. Two disjoint ideas are needed for that, and here we only have room to try to explain one of them – the first one.

It is now time to reveal what the *two more atoms* in the definition of Spider are. They are $An(a, an)$ and $Ta(ta, a)$. The elements an and ta are called the Spider's antenna and tail (a is , as it was earlier, its head). We also have respective two atoms $An(x, x_a) \wedge Ta(x_t, x)$ as the *two more atoms* in the definition of Spider Query.

[29] Clearly, the ⇐ implication is missing.

So far nothing has changed, apart from things being slightly more complicated. Observation 17 still holds true.

But imagine now a complicated structure, full of Spiders (Lame and Full, Red and Green), possibly some of them sharing some body parts (like they do in the *Chase* in Observation 18). And imagine there are two kinds of vertices of this structure: major and minor ones. The major elements serve only as antennas and tails of some Spiders (and each tail or antenna is a major element). Minor elements are spiders' heads and the elements of the legs. Now imagine you are taking your reading glasses off and now you only see the major vertices – the tails and antennas. And you also see the Spiders between them, but only as abstract objects, without being able to notice the details. What you now get is a graph, whose vertices are the major vertices of the old structure, and whose edges are labelled with labels from the set $\{\mathcal{S}_R^i, \mathcal{S}_G^i : 1 \leq i \leq K\} \cup \{\mathcal{S}_R, \mathcal{S}_G\}$ – there is an edge labelled with $\mathcal{S}_R^i$ from a vertex a to b in the "abstract" graph if there are major vertices a and b and a Spider $\mathcal{S}_R^i$ in the original structure, with a being the tail of this Spider and b being its antenna. Nobody is going to be surprised that we will think of such Spider as of an atom $\mathcal{S}_R^i(a, b)$.

8.3 Two-spider queries

Let now $\Psi_{i,j}$ be a Spider Query, as defined in Section 8.1, with the two additional atoms that came in Section 8.2. Let us write $\Psi_{i,j}$ as $\psi_{i,j} \wedge An(x, x_a) \wedge Ta(x_t, x)$. Let also $\Psi'_{i',j'}$ be a new spider query, like $\Psi_{i',j'}$, but with a fresh set of variables – every variable that occurs in $\Psi_{i',j'}$ is primed[30] in $\Psi_{i',j'}$. Let $\Psi'_{i',j'} = \psi'_{i',j'} \wedge An(x', x'_a) \wedge Ta(x'_t, x')$.

Now define $\Phi_{i,j}^{i',j'}$ as the query[31]:

$$\exists x_t \;\; \psi_{i,j} \wedge \psi'_{i',j'} \wedge An(x, x_a) \wedge Ta(x_t, x) \wedge An(x', x'_a) \wedge Ta(x'_t, x') \wedge \; x_t = x'_a$$

The query $\Psi_{i,j}^{i',j'}$, like any other query, generates two Green-Red TGDs, call them $\heartsuit_{gr}$ and $\heartsuit_{rg}$. Let us try to understand when the body of $\heartsuit_{gr}$ is satisfied in some $Chase_l$. The no-glasses view is that there need to be three major vertices in the structure, a, $b = a'$ and b' and it must hold that $Chase_l \models \mathcal{S}_G^i(a, b)$ (suppose this is the case) or $Chase_l \models \mathcal{S}_G^j(a, b)$ and that $Chase_l \models \mathcal{S}_G^{i'}(a', b')$ (suppose this is the case) or $Chase_l \models \mathcal{S}_G^{j'}(a', b')$. Then the TGD $\heartsuit_{gr}$ will be applied, creating a new node c (matching with the existentially quantified variable $x_t = x'_a$) , and – according to the Law of Spider Algebra – new spiders/edges between the major vertices: $\mathcal{S}_R^j(a, c)$ and $\mathcal{S}_R^{j'}(c, b')$.

Notice that our $\Psi_{i,j}^{i',j'}$ (or, rather, the TGDs it generates) does exactly the "first trick" from Section 8.2.

8.4 An easy hill to climb: undecidability of $QDP_e^\infty(CQ, MDL)$

Without the second trick (as we called it in Section 8.2) we of course will not be able to present the proof here, of the result from [8], that $QDP_e^\infty(CQ, CQ)$ is undecidable. But at least we can briefly explain how one can use the "first trick" from Section 8.3 to prove:

▶ **Theorem 19.** *$QDP_e^\infty(CQ, mDL)$ is undecidable.*

[30] Of course, i' and j' are not variables. They are some fixed natural numbers, which may, or may not, be different from i and j.

[31] We use equality in this CQ, which is of course only needed to keep the notation reasonably simple.

It is shown in [5] that $QDP_e^\infty(CQ, DL)$ is undecidable. The proof is essentially what the Reader could see in Section 7, and relies on undecidability of Datalog programs containment. Notice that Theorem 19 is already beyond the reach of the technique from [5], as containment of monadic Datalog programs is decidable.

To cook a proof of Theorem 19 we will need three ingredients. First is a trivial lemma:

▶ **Lemma 20.** *The following problem is undecidable:*

Given a set of word equations[32] *of the form* $\mathcal{S}_G^i \mathcal{S}_G^{i'} = \mathcal{S}_R^j \mathcal{S}_R^{j'}$*, and three numbers*[33] $1 \leq k, k', k'' \leq K$*. Is it true that, for each* $m \in \mathbb{N}$*, the above equations imply that:*

$$\mathcal{S}_G^{k'} (\mathcal{S}_G^k)^m \mathcal{S}_G^{k''} = \mathcal{S}_R^{k'} (\mathcal{S}_R^k)^m \mathcal{S}_R^{k''} \qquad ?$$

Our second ingredient, which we are not going to discuss in details here, is the the input/output procedure. We can write a query, quite similar to a Spider Query, which will add[34], to the Chase, the edge $\mathcal{S}_R^k(a, b)$ everywhere where it was $\mathcal{S}_G(a, b)$ and will add the edge $\mathcal{S}_R(a, b)$ everywhere where it was $\mathcal{S}_G^k(a, b)$. And also, we can write two more similar queries, one of them will add $\mathcal{S}_R^{k'}(a, b)$ everywhere where it was $\mathcal{S}_G(a, b) \wedge \alpha(a)$, where α is a new unary predicate, and the other one will add $\mathcal{S}_R^{k''}(a, b)$ everywhere where it was $\mathcal{S}_G(a, b) \wedge \omega(b)$, with ω being another new unary predicate. This may sound cryptic, but only until you read about the third ingredient, which is our monadic Datalog program Q. Let Φ be the conjunctive query whose frozen body is the Full Spider $\mathcal{S}$, as defined at the beginning of Section 8.1, with variables x_a and x_t matching with the spider's antenna and tail. The program Q will consist of 3 rules:

$$\begin{aligned} \Phi(x_t, x_a), \alpha(x_t) &\Rightarrow M(x_a) \\ M(x_t), \Phi(x_t, x_a) &\Rightarrow M(x_a) \\ M(x_t), \Phi(x_t, x_a), \omega(x_a) &\Rightarrow goal \end{aligned}$$

Now imagine a Green-Red chase for Q and $T_{\mathcal{V}}$, which are the Green-Red TGDs representing our given set of word equations. The elements of $\mathcal{V}$ are two-spider queries, which are CQs, so there will be no room for the Fugitive for any maneuver there. But he can choose $Chase_0$, as it is going to be the green version of some unfolding of Q. Unfoldings of Q are chains of Full Spiders, with the antenna of a predecessor always being the tail of its successor, and with the first tail marked with α and last antenna marked with ω. The input/output rules will produce another chain, of Lame Red Spiders, with the same antennas and tails, and with $\mathcal{S}_R^{k'}$ as the first Spider, $\mathcal{S}_R^{k''}$ as the last one, and with $\mathcal{S}_R^k$ everywhere in between. Then, in the process of Chase, all the words equal to $\mathcal{S}_R^{k'} (\mathcal{S}_R^k)^m \mathcal{S}_R^{k''}$ (modulo the equations enforced by $T_{\mathcal{V}}$) will be created. Which means that if $\mathcal{S}_G^{k'} (\mathcal{S}_G^k)^m \mathcal{S}_G^{k''} = \mathcal{S}_R^{k'} (\mathcal{S}_R^k)^m \mathcal{S}_R^{k''}$ then the word $\mathcal{S}_G^{k'} (\mathcal{S}_G^k)^m \mathcal{S}_G^{k''}$ will be created in *Chase* at some stage. Then, in the next step of Chase, due to the input/output queries, a red version of an unfolding of Q will be produced, causing the Fugitive to lose the game. Of course many details remain unexplained here, including the "only if" direction, but all the important ideas have been presented here.

[32] We consider a semigroup here, whose generators are (names of) Green and Red Lame Spiders.
[33] We already met these, or at least similar, k and k', in Observation 18.
[34] More precisely, a red-green TGD generated by this query will add.

References

1 Foto N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, 412(11):1005–1021, 2011. doi:10.1016/j.tcs.2010.12.031.

2 Michael Benedikt, Stanislav Kikot, Piotr Ostropolski-Nalewaja, and Miguel Romero Orth. Unpublished manuscript, 2019.

3 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. What is View-Based Query Rewriting? In *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB 2000), Berlin, Germany, August 21, 2000*, pages 17–27, 2000. URL: http://ceur-ws.org/Vol-29/02-cdlv.ps.

4 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Lossless Regular Views. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 247–258, New York, NY, USA, 2002. ACM. doi:10.1145/543613.543646.

5 Wenfei Fan, Floris Geerts, and Lixiao Zheng. View determinacy for preserving selected information in data transformations. *Inf. Syst.*, 37:1–12, 2012.

6 Grzegorz Gluch, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. Can One Escape Red Chains?: Regular Path Queries Determinacy is Undecidable. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 492–501, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209120.

7 Grzegorz Gluch, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. The First Order Truth Behind Undecidability of Regular Path Queries Determinacy. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 15:1–15:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.15.

8 Tomasz Gogacz and Jerzy Marcinkowski. The Hunt for a Red Spider: Conjunctive Query Determinacy Is Undecidable. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, pages 281–292, Washington, DC, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.35.

9 Tomasz Gogacz and Jerzy Marcinkowski. Red Spider Meets a Rainworm: Conjunctive Query Finite Determinacy Is Undecidable. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 121–134, New York, NY, USA, 2016. ACM. doi:10.1145/2902251.2902288.

10 Per-Åke Larson and H. Z. Yang. Computing Queries from Derived Relations. In *Proceedings of the 11th International Conference on Very Large Data Bases - Volume 11*, VLDB '85, pages 259–269. VLDB Endowment, 1985. URL: http://dl.acm.org/citation.cfm?id=1286760.1286784.

11 Alan Nash, Luc Segoufin, and Victor Vianu. Determinacy and Rewriting of Conjunctive Queries Using Views: A Progress Report. In Thomas Schwentick and Dan Suciu, editors, *Database Theory – ICDT 2007*, pages 59–73, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

12 Daniel Pasailă. Conjunctive Queries Determinacy and Rewriting. In *Proceedings of the 14th International Conference on Database Theory*, ICDT '11, pages 220–231, New York, NY, USA, 2011. ACM. doi:10.1145/1938551.1938580.

13 Luc Segoufin and Victor Vianu. Views and queries: determinacy and rewriting. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, pages 49–60, 2005. doi:10.1145/1065167.1065174.

Current Challenges in Graph Databases

Juan L. Reutter
School of Engineering, Pontificia Universidad Catolica, Santiago, Chile
IMFD, Santiago, Chile

Abstract

As graph databases grow in popularity, decades of work in graph query languages and models are materialising in industry standards and in the construction of new graph database systems. However, this surge in graph systems has in turn opened up a series of new, interesting research problems related to graph databases.

Our first set of problems has to do with more efficient ways of computing the answers of graph queries, specifically graph patterns, path queries, and combinations between them. Traditionally, researchers in graph databases have pointed out that relational systems are ill-equipped to process these types of queries, and if one looks at the performance of native graph database systems, there is clearly a lot of room for improvement. The talk focuses on two possible directions for improving the state of the art in graph query processing. The first is implementing worst-case optimal algorithms for processing graph patterns that traduce in relational queries with several joins. Some advances are already in development (see e.g. Nguyen, Dung, et al. "Join processing for graph patterns: An old dog with new tricks." GRADES'15. or Hogan, Aidan, et al. "A Worst-Case Optimal Join Algorithm for SPARQL." ISWC'19.), but we are still far from a full fledged solution: most algorithms require complex data structures, or need further support in terms of heuristics to select an order in which joins are processed. Second, we need to understand what is the best way of evaluating path queries (that is, finding all pairs of nodes connected by a path), in such a way that these results can be further integrated with other query results in a graph system pipeline. We already have complexity results regarding path computation and enumeration for different semantics of path queries (see e.g. Martens, Wim, and Tina Trautner. "Evaluation and enumeration problems for regular path queries." ICDT'18. or Bagan, Guillaume, Angela Bonifati, and Benoit Groz. "A trichotomy for regular simple path queries on graphs." PODS'13.), but still very little is known in terms of optimal processing of path queries when inside a tractable fragment.

Our second set of problems is related to graph analytics, one of the current selling points of graph databases. Systems should be able to run more complex analytical queries involving tasks such as more complex path finding, centrality or clustering. It is also important to be able to run these algorithms not over native graphs, but perhaps over a certain set of nodes or edges previously selected by a graph query, and one may also want to pose further queries over the result of the analytics task. Finally, all of this should be done in an efficient way, specially in the prospect that graph databases may contain a huge amount of nodes. In this talk I will discuss possible approaches to perform these operations, covering aspects from the design of languages for graph analytics to efficient ways of processing them, and also comparing the expressive power of graph analytics solutions with other forms of graph computation.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Graph databases, Join algorithms, path queries, graph analytics

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.3

Category Invited Talk

Funding This work was partially funded by Millennium Institute for Foundational Research on Data (IMFD).

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 3; pp. 3:1–3:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Executable First-Order Queries in the Logic of Information Flows

Heba Aamer
Universiteit Hasselt, Belgium

Bart Bogaerts
Vrije Universiteit Brussel, Belgium

Dimitri Surinx
Universiteit Hasselt, Belgium

Eugenia Ternovska
Simon Fraser University, Canada

Jan Van den Bussche
Universiteit Hasselt, Belgium

Abstract

The logic of information flows (LIF) has recently been proposed as a general framework in the field of knowledge representation. In this framework, tasks of a procedural nature can still be modeled in a declarative, logic-based fashion. In this paper, we focus on the task of query processing under limited access patterns, a well-studied problem in the database literature. We show that LIF is well-suited for modeling this task. Toward this goal, we introduce a variant of LIF called "forward" LIF, in a first-order setting. We define FLIF$^{\text{io}}$, a syntactical fragment of forward LIF, and show that it corresponds exactly to the "executable" fragment of first-order logic defined by Nash and Ludäscher. The definition of FLIF$^{\text{io}}$ involves a classification of the free variables of an expression into "input" and "output" variables. Our result hinges on inertia and determinacy laws for forward LIF expressions, which are interesting in their own right. These laws are formulated in terms of the input and output variables.

2012 ACM Subject Classification Information systems → Query languages; Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Logic of Information Flows, Limited access pattern, Executable first-order logic

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.4

Funding Jan Van den Bussche is partially supported by the National Natural Science Foundation of China (61972455). This research was partially supported by FWO project G0D9616N and by the Flanders AI Research Program.

Acknowledgements We thank the ICDT reviewers for pointing out the connection to related work [10, 12].

1 Introduction

An information source is said to have a limited access pattern if it can only be accessed by providing values for a specified subset of the attributes; the source will then respond with tuples giving values for the remaining attributes. A typical example is a restricted telephone directory $D(\text{name}; \text{tel})$ that will show the phone numbers for a given name, but not the other way around.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 4; pp. 4:1–4:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The querying of information sources with limited access patterns has been quite intensively investigated. The research is motivated by diverse considerations, such as query processing using indices, or information integration on the Web. We refer to the review given by Benedikt et al. [5, Chapter 3.12]. We also cite the work by Calì and collaborators [7, 8, 9].

In this paper, we offer a fresh perspective on querying with limited access patterns, based on the Logic of Information Flows (LIF). This framework has been recently introduced in the field of knowledge representation [18, 19]. The general aim of LIF is to model how information propagates in complex systems. LIF allows machine-independent characterizations of computation; in particular, it allows tasks of a procedural nature to be modeled in a declarative fashion.

In the full setting, LIF is a rich family of logics with higher-order features. The present paper is self-contained, however, and we will work in a lightweight, first-order fragment, which we call *forward* LIF (FLIF). Specifically, we will define a well-behaved, syntactic fragment of FLIF, called *io-disjoint* FLIF. Our main result then is to establish an equivalence between io-disjoint FLIF and *executable first-order logic* (executable FO).

Executable FO [15] is a syntactic fragment of FO in which formulas can be evaluated over information sources in such a way that the limited access patterns are respected. Furthermore, the syntactical restrictions are not very severe and become looser the more free variables are declared as input.

The standard way of formalizing query processing with limited access patterns is by a form of relational algebra programs, called plans [5]. In such plans, database relations can only be accessed by joining them on their input attributes with a relation that is either given as input or has already been computed. Apart from that, plans can use the usual relational algebra operations. Plans can be expressed by executable FO formulas. The strong result [6] is known that every (boolean) FO formula with the semantic property of being *access-determined* can be evaluated by a plan. We will not need this result further on, but it provides a strong justification for working with executable FO formulas.

Our language, FLIF, provides a new, *navigational* perspective on query processing with limited access patterns. In our approach, we formalize the database as a *graph* of variable bindings. Directed edges are labeled with the names of source relations (we are simplifying a bit here). A directed edge $\nu_1 \xrightarrow{R} \nu_2$ indicates that, if we access R with input values given by ν_1, then the output values in ν_2 are a possible result. In a manner very similar to navigational or XPath-like graph query languages [16, 14, 4, 11, 17, 3], FLIF expressions represent paths in the graph.

The io-disjoint fragment of FLIF is defined in terms of *input* and *output* variables that are inferred for expressions. We establish *inertia* and *input-determinacy* properties for FLIF expressions which are instrumental in proving our equivalence between io-disjoint expressions and executable FO, but are also interesting in their own right. Apart from the intuitive navigational nature, another advantage of io-disjoint FLIF is that it is very obvious how expressions in this language can be evaluated by plans. As we will show, the structure of the evaluation plan closely follows the shape of the expression, and all joins can be taken to be natural joins; no attribute renamings are needed.

This paper is further organized as follows. Section 2 recalls the basic setting of executable FO on databases with limited access patterns. Section 3 introduces the language FLIF. Section 4 gives translations between executable FO and io-disjoint FLIF, showing that the evaluation problems for the two languages can be naturally reduced to each other. Section 5 discusses evaluation plans. We conclude in Section 6.

2 Executable FO

Relational database schemas are commonly formalized as finite relational vocabularies, i.e., finite collections of relation names, each name with an associated arity (a natural number). To model limited access patterns, we additionally specify an *input arity* for each name. For example, if R has arity five and input arity two, this means that we can only access R by giving input values, say a_1 and a_2, for the first two arguments; R will then respond with all tuples $(x_1, x_2, x_3, x_4, x_5)$ in R where $x_1 = a_1$ and $x_2 = a_2$.

Thus, formally, we define a *database schema* as a triple $\mathcal{S} = (\mathit{Names}, \mathrm{ar}, \mathrm{iar})$, where *Names* is a set of relation names; ar assigns a natural number $\mathrm{ar}(R)$ to each name R in *Names*, called the arity of R; and iar similarly assigns an input arity to each R, such that $\mathrm{iar}(R) \leq \mathrm{ar}(R)$.

▶ Remark 1. In the literature, a more general notion of schema is often used, allowing, for each relation name, several possible sets of input arguments; each such set is called an access method. In this paper, we stick to the simplest setting where there is only one access method per relation, consisting of the first k arguments, where k is set by the input arity. All subtleties and difficulties already show up in this setting. Nevertheless, our definitions and results can be easily generalized to the setting with multiple access methods per relation. ◀

The notion of database instance remains the standard one. Formally, we fix a countably infinite universe **dom** of atomic data elements, also called *constants*. Now an *instance* D of a schema $\mathcal{S}$ assigns to each relation name R an $\mathrm{ar}(R)$-ary relation $D(R)$ on **dom**. We say that D is *finite* if every relation $D(R)$ is finite. The *active domain* of D, denoted by $\mathrm{adom}(D)$, is the set of all constants appearing in the relations of D.

The syntax and semantics of first-order logic (FO, relational calculus) over $\mathcal{S}$ is well known [2]. In formulas, we allow constants only in equalities of the form $x = c$, where x is a variable and c is a constant. Also, in writing relation atoms, we find it clearer to separate input arguments from output arguments by a semicolon. Thus, we write relation atoms in the form $R(\bar{x}; \bar{y})$, where $\bar{x}$ and $\bar{y}$ are tuples of variables such that the length of $\bar{x}$ is $\mathrm{iar}(R)$ and the length of $\bar{y}$ is $\mathrm{ar}(R) - \mathrm{iar}(R)$. The set of free variables of a formula φ is denoted by $\mathrm{FV}(\varphi)$.

We use the "natural" semantics [2] and let variables in formulas range over the whole of **dom**. Formally, a *valuation* on a set X of variables is a mapping $\nu : X \to \mathbf{dom}$. Given an instance D of $\mathcal{S}$, an FO formula φ over $\mathcal{S}$, and a valuation ν defined on $\mathrm{FV}(\varphi)$, the definition of when φ is satisfied by D and ν, denoted by $D, \nu \models \varphi$, is standard.

A well-known problem with the natural semantics for general FO formulas is that φ may be satisfied by infinitely many valuations on $\mathrm{FV}(\varphi)$, even if D is finite. However, as motivated in the Introduction, we will focus on *executable* formulas, formally defined in this section. These formulas can safely be used under the natural semantics.

The notion of when a formula is executable is defined relative to a set of variables $\mathcal{V}$, which specifies the variables for which input values are already given. We first give a few examples.

▶ **Example 2.**

- Let φ be the formula $R(x; y)$. As mentioned above, this notation makes clear that the input arity of R is one. If we provide an input value for x, then the database will give us all y such that $R(x, y)$ holds. Indeed, φ will turn out to be $\{x\}$-executable. Giving a value for the first argument of R is mandatory, so φ is neither $\emptyset$-executable nor $\{y\}$-executable. However, it is certainly allowed to provide input values for both x and y; in that case we are merely testing if $R(x, y)$ holds for the given pair (x, y). Thus, φ is also $\{x, y\}$-executable. In general, a $\mathcal{V}$-executable formula will also be $\mathcal{V}'$-executable for any $\mathcal{V}' \supseteq \mathcal{V}$.

- Also the formula $\exists y\, R(x; y)$ is $\{x\}$-executable. In contrast, the formula $\exists x\, R(x; y)$ is not, because even if a value for x is given as input, it will be ignored due to the existential quantification. In fact, the latter formula is not $\mathcal{V}$-executable for any $\mathcal{V}$.
- The formula $R(x; y) \wedge S(y; z)$ is $\{x\}$-executable, intuitively because each y returned by the formula $R(x; y)$ can be fed into the formula $S(y; z)$, which is $\{y\}$-executable in itself.
- The formula $R(x; y) \vee S(x; z)$ is not $\{x\}$-executable, because any y returned by $R(x; y)$ would already satisfy the formula, leaving variable z unconstrained. This would lead to an infinite number of satisfying valuations. The formula is neither $\{x, z\}$-executable; if $S(x, z)$ holds for the given values for x and z, then y is left unconstrained. Of course, the formula is $\{x, y, z\}$-executable.
- For a similar reason, $\neg R(x; y)$ is only $\mathcal{V}$-executable for $\mathcal{V}$ containing x and y. ◀

Formally, for any set of variables $\mathcal{V}$, the $\mathcal{V}$-executable formulas are defined as follows.

- An equality $x = y$, for variables x and y, is $\mathcal{V}$-executable if at least one of x and y belongs to $\mathcal{V}$.
- An equality $x = c$, for a variable x and a constant c, is always $\mathcal{V}$-executable.
- A relation atom $R(\bar{x}; \bar{y})$ is $\mathcal{V}$-executable if $X \subseteq \mathcal{V}$, where X is the set of variables from $\bar{x}$.
- A negation $\neg\varphi$ is $\mathcal{V}$-executable if φ is, and moreover $\mathrm{FV}(\varphi) \subseteq \mathcal{V}$.
- A conjunction $\varphi \wedge \psi$ is $\mathcal{V}$-executable if φ is, and moreover ψ is $\mathcal{V} \cup \mathrm{FV}(\varphi)$-executable.
- A disjunction $\varphi \vee \psi$ is $\mathcal{V}$-executable if both φ and ψ are, and moreover $\mathrm{FV}(\varphi) \triangle \mathrm{FV}(\psi) \subseteq \mathcal{V}$. Here, $\triangle$ denotes symmetric difference.
- An existential quantification $\exists x\, \varphi$ is $\mathcal{V}$-executable if φ is $\mathcal{V} - \{x\}$-executable.

Note that universal quantification is not part of the syntax of executable FO.

▶ Remark 3. The naturalness of the above definition may be attested by its reinvention in the context of a different application, namely, inferring bounds on result sizes of FO queries. Indeed, the notion of "controlled" formula that was introduced for this purpose, strikingly conforms to that of executable formula [10]. In the setting of controlled formulas, the input arity k of an n-ary relation R is interpreted as an integrity constraint. An instance D satisfies the constraint if for each k-tuple $\bar{a}$ of constants, the number of $n - k$-tuples $\bar{b}$ such that $\bar{a} \cdot \bar{b} \in D(R)$ stays below a fixed upper bound. ◀

Given an FO formula φ and a finite set of variables $\mathcal{V}$ such that φ is $\mathcal{V}$-executable, we describe the following task:

Problem: The evaluation problem $\mathit{Eval}_{\varphi,\mathcal{V}}(D, \nu_{\mathrm{in}})$ for φ with input variables $\mathcal{V}$.
Input: A database instance D and a valuation ν_{in} on $\mathcal{V}$.
Output: The set of all valuations ν on $\mathcal{V} \cup \mathrm{FV}(\varphi)$ such that $\nu_{\mathrm{in}} \subseteq \nu$ and $D, \nu \models \varphi$.

As mentioned in the Introduction, this problem is known to be solvable by a relational algebra plan respecting the access patterns. In particular, if D is finite, the output is always finite: each valuation ν in the output can be shown to take only values in $\mathrm{adom}(D) \cup \nu_{\mathrm{in}}(\mathcal{V})$.[1]

[1] Actually, a stronger property can be shown: only values that are "accessible" from ν_{in} in D can be taken [5], and if this accessible set is finite, the output of the evaluation problem is finite. This will also follow immediately from our equivalence between executable FO and FLIF$^{\mathrm{io}}$.

3 Forward LIF, inputs, and outputs

In this section, we introduce the language FLIF.[2] It will be notationally convenient here to work under the following proviso:

▶ **Proviso 4.** *When we write "valuation" without specifying on which variables it is defined, we assume it is defined on all variables. (Formally, we assume a countably infinite universe of variables.)*

Importantly, we will define the semantics of an FLIF expression in such a way that it depends only on the value of the valuations on the free variables of the expression. This situation is comparable to the classical way in which the semantics of first-order logic is often defined.

The central idea is to view a database as a graph. The nodes of the graph are all possible valuations (hence the graph is infinite.) The edges in the graph are labeled with *atomic FLIF expressions*. Over a schema $\mathcal{S}$, there are five kinds of atomic expressions τ, given by the following grammar:

$$\tau ::= R(\bar{x}; \bar{y}) \mid (x = y) \mid (x = c) \mid (x := y) \mid (x := c)$$

Here, $R(\bar{x}; \bar{y})$ is a relation atom over $\mathcal{S}$ as in first-order logic, x and y are variables, and c is a constant.

Given an instance D of $\mathcal{S}$, and an atomic expression τ, we define the set of τ-labeled edges in the graph representation of D as a set $[\![\tau]\!]_D$ of ordered pairs of valuations, as follows.

1. $[\![R(\bar{x}; \bar{y})]\!]_D$ is the set of all pairs (ν_1, ν_2) of valuations such that the concatenation $\nu_1(\bar{x}) \cdot \nu_2(\bar{y})$ belongs to $D(R)$, and ν_1 and ν_2 agree outside the variables in $\bar{y}$.
2. $[\![(x := y)]\!]_D$ is the set of all pairs (ν_1, ν_2) of valuations such that $\nu_2 = \nu_1[x := \nu_1(y)]$. Thus, $\nu_2(x) = \nu_1(y)$ and ν_2 agrees with ν_1 on all other variables.
3. Similarly, $[\![(x := c)]\!]_D$ is the set of all pairs (ν_1, ν_2) of valuations such that $\nu_2 = \nu_1[x := c]$.
4. $[\![(x = y)]\!]_D$ is the set of all identical pairs (ν, ν) such that $\nu(x) = \nu(y)$.
5. Likewise, $[\![(x = c)]\!]_D$ is the set of all identical pairs (ν, ν) such that $\nu(x) = c$.

The syntax of all FLIF expressions α is now given by the following grammar:

$$\alpha ::= \tau \mid \alpha \,;\, \alpha \mid \alpha \cup \alpha \mid \alpha \cap \alpha \mid \alpha - \alpha$$

Here, τ ranges over atomic expressions as defined above. The semantics of ';' is composition, defined as follows:

$$[\![\alpha_1 \,;\, \alpha_2]\!]_D = \{(\nu_1, \nu_3) \mid \exists \nu_2 : (\nu_1, \nu_2) \in [\![\alpha_1]\!]_D \text{ and } (\nu_2, \nu_3) \in [\![\alpha_2]\!]_D\}$$

The semantics of the set operations are standard union, intersection and set difference.

We see that FLIF expressions describe paths in the graph, in the form of source–target pairs. Composition is used to navigate through the graph, and to conjoin paths. Paths can be branched using union, merged using intersection, and excluded using set difference.

▶ Remark 5. The way the smantics of FLIF is defined is in line with first-order dynamic logic or dynamic predicate logic (DPL) [13, 12]. DPL gives a dynamic interpretation to existential quantification and interprets conjunction as composition. For example, the FLIF expression $R(x; y) \,;\, S(y; z)$ would be expressed in DPL as $\exists y\, R(x, y) \wedge \exists z\, S(y, z)$. On the other hand, disjunction in DPL is always interpreted as a test. Because of this, FLIF expressions such as $R(x; y) \cup S(u; v)$ seem inexpressible in DPL.

[2] Pronounced as "eff-lif".

▶ **Example 6.** Consider a simple Facebook abstraction with a single binary relation F of input arity one. When given a person as input, F returns all their friends. We assume that this relation is symmetric. Suppose, for an input person x (say, a famous person), we want to find all people who are friends with at least two friends of x. Formally, we want to navigate from a valuation ν_1 giving a value for x, to all valuations ν_2 giving values to variables y_1, y_2, and z, such that

- $\nu_1(x)$ is friends with both $\nu_2(y_1)$ and $\nu_2(y_2)$;
- $\nu_2(y_1)$ and $\nu_2(y_2)$ are both friends with $\nu_2(z)$; and
- $\nu_2(y_1) \neq \nu_2(y_2)$.

This can be done by the expression $\alpha - (\alpha \mathbin{;} (y_1 = y_2))$, where α is the expression

$$(F(x; y_1) \mathbin{;} F(y_1; z)) \cap (F(x; y_2) \mathbin{;} F(y_2; z)).$$

▶ Remark 7. In the above example, it would be more efficient to simply write $\alpha \mathbin{;} (y_1 \neq y_2)$. For simplicity, we have not added nonequality tests in FLIF as they are formally redundant in the presence of set difference, but they can easily be added in practice. ◀

In every expression we can identify the *input* and the *output* variables. Intuitively, the output variables are those that can change value along the execution path; the input variables are those whose value at the beginning of the path is needed in order to know the possible values for the output variables. These intuitions will be formalized below. We first give some examples.

▶ **Example 8.**

- In the expression α from Example 6, the only input variable is x, and the other variables are output variables.
- FLIF, in general, allows expressions where a variable is both input and output. For example, assume **dom** contains the natural numbers and consider a binary relation Inc of input arity one that holds pairs of natural numbers $(n, n+1)$. Then it is reasonable to use an expression $\mathit{Inc}(x; x)$ to increment the value x. Formally, this expression defines all pairs of valuations (ν_1, ν_2) such that $\nu_2(x) = \nu_1(x) + 1$ (and ν_2 agrees with ν_1 on all other variables).
- Consider the expression $R(x; y_1) \cap S(x; y_2)$. Then not only x, but also y_1 and y_2 are input variables. Indeed, the expression $R(x; y_1)$ will pair an input valuation ν_1 with an output valuation ν_2 that sets y_1 such that $R(\nu_1(x), \nu_2(y_1))$ holds, but ν_2 will have the same value as ν_1 on any other variable. In particular, $\nu_2(y_2) = \nu_1(y_2)$. The expression $S(x; y_2)$ has a similar behavior, but with y_1 and y_2 interchanged. Thus, the intersection expression checks two conditions on the input valuation; formally, it defines all identical pairs (ν, ν) for which $R(\nu(x), \nu(y_1))$ and $S(\nu(x), \nu(y_2))$ hold. Since the expression only tests conditions, it has no output variables.
- On the other hand, for the expression $R(x; y_1) \cup S(x; y_2)$, the output variables are y_1 and y_2. Indeed, consider an input valuation ν_1 with $\nu_1(x) = a$. The expression pairs ν_1 either with a valuation giving a new value for y_1, or with a valuation giving a new value for y_2. However, y_1 and y_2 are also input variables (together with x). Indeed, when pairing ν_1 with a valuation ν_2 that sets y_2 to some b for which $S(a, b)$ holds, we must know the value of $\nu_1(y_1)$ so as to preserve it in ν_2. A similar argument holds for y_2. ◀

Table 1 now formally defines, for any expression α, the sets $I(\alpha)$ and $O(\alpha)$ of input and output variables. We denote the union of $I(\alpha)$ and $O(\alpha)$ by $\mathrm{FV}(\alpha)$. We refer to this set as the *free variables* of α, but note that it actually equals the set of all variables occurring in the expression.

Table 1 Input and output variables of FLIF expressions. In the case of $R(\bar{x};\bar{y})$, the set X is the set of variables in $\bar{x}$, and the set Y is the set of variables in $\bar{y}$. Recall that $\triangle$ is symmetric difference.

α	$I(\alpha)$	$O(\alpha)$
$R(\bar{x};\bar{y})$	X	Y
$(x = y)$	$\{x, y\}$	$\emptyset$
$(x := y)$	$\{y\}$	$\{x\}$
$(x = c)$	$\{x\}$	$\emptyset$
$(x := c)$	$\emptyset$	$\{x\}$
$\alpha_1;\alpha_2$	$I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1))$	$O(\alpha_1) \cup O(\alpha_2)$
$\alpha_1 \cup \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1) \cup O(\alpha_2)$
$\alpha_1 \cap \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1) \cap O(\alpha_2)$
$\alpha_1 - \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1)$

We next establish three propositions that show that our definition of inputs and outputs, which is purely syntactic, reflects actual properties of the semantics. The first proposition confirms an intuitive property and can be straightforwardly verified by induction.

▶ **Proposition 9** (Law of inertia). *If $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ then ν_2 agrees with ν_1 outside $O(\alpha)$.*

The second proposition confirms, as announced earlier, that the semantics of expressions depends only on the free variables; outside $\mathrm{FV}(\alpha)$, the binary relation $[\![\alpha]\!]_D$ is cylindrical. The proof for difference expressions is not immediate, and uses the law of inertia.

▶ **Proposition 10** (Free variable property). *Let $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ and let ν_1' and ν_2' be valuations such that*

- *ν_1' agrees with ν_1 on $\mathrm{FV}(\alpha)$, and*
- *ν_2' agrees with ν_2 on $\mathrm{FV}(\alpha)$, and agrees with ν_1' outside $\mathrm{FV}(\alpha)$.*

Then also $(\nu_1', \nu_2') \in [\![\alpha]\!]_D$.

The third proposition is the most important one, and is proven using the previous two. It confirms that the values for the input variables determine the values for the output variables.

▶ **Proposition 11** (Input determinacy). *Let $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ and let ν_1' be a valuation that agrees with ν_1 on $I(\alpha)$. Then there exists a valuation ν_2' that agrees with ν_2 on $O(\alpha)$, such that $(\nu_1', \nu_2') \in [\![\alpha]\!]_D$.*

By the law of inertia, the valuation ν_2' given by the above proposition is unique.

We are now in a position to formulate the FLIF evaluation problem. Given an expression α, we consider the following task:[3]

Problem: The evaluation problem $\mathit{Eval}_\alpha(D, \nu_{\text{in}})$ for α.
Input: A database instance D and a valuation ν_{in} on $I(\alpha)$.
Output: The set $\{\nu_{\text{out}}|_{\mathrm{FV}(\alpha)} \mid \exists \nu_{\text{in}}' : \nu_{\text{in}} \subseteq \nu_{\text{in}}' \text{ and } (\nu_{\text{in}}', \nu_{\text{out}}) \in [\![\alpha]\!]_D\}$.

[3] For a valuation ν on a set of variables X (possibly all variables), and a subset Y of X, we use $\nu|_Y$ to denote the restriction of ν to X.

By inertia and input determinacy, the choice of ν'_{in} in the definition of the output does not matter. Moreover, if D is finite, the output is finite as well. As was the case for executable FO, the above problem can be solved by a relational algebra plan respecting the access patterns. Unfortunately, since the sets of input and output variables of general FLIF expressions need not be disjoint, the plan is a bit intricate; we have to work with relations that have two copies for every variable, to keep track of how assignments are paired up.

For this reason, in the next section, we introduce a well-behaved fragment called *io-disjoint* FLIF. Plans for expressions in this fragment can be generated in a very transparent manner, as is shown in Section 5.

4 Executable FO and io-disjoint FLIF

Consider an FLIF expression α for which the set $O(\alpha)$ is disjoint from $I(\alpha)$. Then any pair $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ satisfies that ν_1 and ν_2 are equal on $I(\alpha)$. Put differently, every $\nu_{\text{out}} \in \mathit{Eval}_\alpha(D, \nu_{\text{in}})$ is equal to ν_{in} on $I(\alpha)$; all that the evaluation does is expand the input valuation with output values for the new output variables. This makes the evaluation process for expressions α where $I(\beta) \cap O(\beta) = \emptyset$, for every subexpression β of α (including α itself), very transparent. We call such expressions *io-disjoint.*

The following proposition makes it easier to check if an expression is io-disjoint:

▶ **Proposition 12.** *The following alternative definition of io-disjointness is equivalent to the definition given above:*

- *An atomic expression $R(\bar{x}; \bar{y})$ is io-disjoint if $X \cap Y = \emptyset$, where X is the set of variables in $\bar{x}$, and Y is the set of variables in $\bar{y}$.*
- *Atomic expressions of the form $(x = y)$, $(x = c)$, $(x := y)$ or $(x := c)$ are io-disjoint.*
- *A composition $\alpha_1 \mathbin{;} \alpha_2$ is io-disjoint if α_1 and α_2 are, and moreover $I(\alpha_1) \cap O(\alpha_2) = \emptyset$.*
- *A union $\alpha_1 \cup \alpha_2$ is io-disjoint if α_1 and α_2 are, and moreover $O(\alpha_1) = O(\alpha_2)$.*
- *An intersection $\alpha_1 \cap \alpha_2$ is io-disjoint if α_1 and α_2 are.*
- *A difference $\alpha_1 - \alpha_2$ is io-disjoint if α_1 and α_2 are, and moreover $O(\alpha_1) \subseteq O(\alpha_2)$.*

The fragment of io-disjoint expressions is denoted by FLIF^{io}. We are going to show that FLIF^{io} is expressive enough, in the sense that executable FO can be translated into FLIF^{io}. The converse translation is also possible, so, FLIF^{io} exactly matches executable FO in expressive power.

Recall the evaluation problem for executable FO, as defined at the end of Section 2, and the evaluation problem for α, as defined at the end of the previous section. We can now formulate the translation result from executable FO to FLIF^{io} as follows.

▶ **Theorem 13.** *Let φ be a $\mathcal{V}$-executable formula over schema $\mathcal{S}$. There exists an* FLIF^{io} *expression α over $\mathcal{S}$ with the following properties:*

1. $I(\alpha) = \mathcal{V}$.
2. $O(\alpha) \supseteq \text{FV}(\varphi) - \mathcal{V}$.
3. *For every D and ν_{in}, we have $\mathit{Eval}_{\varphi,\mathcal{V}}(D, \nu_{\text{in}}) = \pi_{\text{FV}(\varphi)\cup\mathcal{V}}(\mathit{Eval}_\alpha(D, \nu_{\text{in}}))$.*

The length of α is polynomial in the length of φ and the cardinality of $\mathcal{V}$.

The above projection operator π restricts each valuation in $\mathit{Eval}_\alpha(D, \nu_{\text{in}})$ to $\text{FV}(\varphi) \cup \mathcal{V}$. It is imposed because we allow $O(\alpha)$ to have auxiliary variables not in $\text{FV}(\varphi)$.

▶ **Example 14.** Before giving the proof, we give a few examples.

- Suppose φ is $R(x; y)$ with input variable x. Then, as expected, α can be taken to be $R(x; y)$.

- However, now consider $T(x;x,y)$, again with input variable x. Intuitively, the formula asks for outputs (u,y) where u equals x. Hence, a suitable io-disjoint translation is $T(x;u,y)\,;(u=x)$.
- If φ is $R(x;y)\wedge S(y;z)$, still with input variable x, we can take $R(x;y)\,;S(y;z)$ for α. The same expression also serves for the formula $\exists y\,\varphi$. However, if φ is $\exists y\,R(x;y)$ with $\mathcal{V}=\{x,y\}$, we must use a fresh variable and use $R(x;u)\,;(y=y)$ for α. The test $(y=y)$ may seem spurious but is needed to ensure that $I(\alpha)=\mathcal{V}$.
- Suppose φ is $R(x;x)\vee S(y;)$ with $\mathcal{V}=\{x,y\}$. For this $\mathcal{V}$, we translate $R(x;x)$ to $R(x;u)\,;(x=u)\,;(y=y)$. Similarly, $S(y;)$ is translated to $S(y;)\,;(x=x)$. Unfortunately the union of these two expressions is not io-disjoint. We can formally solve this by composing the second expression with a dummy assignment to u. So the final α can be taken to be $R(x;u)\,;(x=u)\,;(y=y)\cup S(y;)\,;(x=x)\,;(u:=42)$. Since the output valuations will be projected on $\{x,y\}$, the choice of the constant assigned to u is irrelevant.
- A similar trick can be used for negation. For example, if φ is $\neg R(x;y)$ with $\mathcal{V}=\{x,y\}$, then α can be taken to be $(u:=42)\,-\,R(x;u)\,;(u=y)\,;(u:=42)$.

Proof. We only describe the translation; its correctness, which hinges on the law of inertia and input determinacy, also involves verifying that io-disjointness holds.

If φ is a relation atom $R(\bar{x};\bar{y})$, then α is $R(\bar{x};\bar{z})\,;\xi\,;\xi'$, where $\bar{z}$ is obtained from $\bar{y}$ by replacing each variable from $\mathcal{V}$ by a fresh variable. The expression ξ consists of the composition of all equalities $(y_i=z_i)$ where y_i is a variable from $\bar{y}$ that is in $\mathcal{V}$ and z_i is the corresponding fresh variable. The expression ξ' consists of the composition of all equalities $(u=u)$ with u a variable in $\mathcal{V}$ not mentioned in φ.

If φ is $x=y$, then α is

$$\begin{cases}(x=y)\,;\xi & \text{if } x,y\in\mathcal{V}\\ (x:=y)\,;\xi & \text{if } x\notin\mathcal{V}\\ (y:=x)\,;\xi & \text{if } y\notin\mathcal{V},\end{cases}$$

where ξ is the composition of all equalities $(u=u)$ with u a variable in $\mathcal{V}$ not mentioned in φ.

If φ is $x=c$, then α is

$$\begin{cases}(x=c)\,;\xi & \text{if } x\in\mathcal{V}\\ (x:=c)\,;\xi & \text{otherwise,}\end{cases}$$

with ξ as in the previous case.

If φ is $\varphi_1\wedge\varphi_2$, then by induction we have an expression α_1 for φ_1 and $\mathcal{V}$, and an expression α_2 for φ_2 and $\mathcal{V}\cup\mathrm{FV}(\varphi_1)$. Now α can be taken to be $\alpha_1\,;\alpha_2$.

If φ is $\exists x\,\varphi_1$, then without loss of generality we may assume that $x\notin V$. By induction we have an expression α_1 for φ_1 and $\mathcal{V}$. This expression also works for φ.

If φ is $\varphi_1\vee\varphi_2$, then by induction we have an expression α_i for φ_i and $\mathcal{V}$, for $i=1,2$. Fix an arbitrary constant c, and let ξ_1 be the composition of all expressions $(z:=c)$ for $z\in O(\alpha_2)-O(\alpha_1)$; let ξ_2 be defined symmetrically. Now α can be taken to be $\alpha_1\,;\xi_1\cup\alpha_2\,;\xi_2$.

Finally, if φ is $\neg\varphi_1$, then by induction we have an expression α_1 for φ_1 and $\mathcal{V}$. Fix an arbitrary constant c, and let ξ be the composition of all expressions $(z:=c)$ for $z\in O(\alpha_1)$. (If $O(\alpha_1)$ is empty, we add an additional fresh variable.) Then α can be taken to be $\xi\,-\,\alpha_1;\xi$. ◀

We next turn to the converse translation. Here, a sharper equivalence is possible, since executable FO has an explicit quantification operation which is lacking in FLIF.

Table 2 Translation showing how FLIF$^{\text{io}}$ embeds in executable FO. In the table, φ_i abbreviates φ_{α_i} for $i = 1, 2$.

α	φ_α
$R(\bar{x}; \bar{y})$	$R(\bar{x}; \bar{y})$
$(x = y)$	$x = y$
$(x := y)$	$x = y$
$x = c$	$x = c$
$x := c$	$x = c$
$\alpha_1 ; \alpha_2$	$(\exists x_1 \dots \exists x_k\, \varphi_1) \wedge \varphi_2$ where $\{x_1, \dots, x_k\} = O(\alpha_1) \cap O(\alpha_2)$
$\alpha_1 \cup \alpha_2$	$\varphi_1 \vee \varphi_2$
$\alpha_1 \cap \alpha_2$	$\varphi_1 \wedge \varphi_2$
$\alpha_1 - \alpha_2$	$\varphi_1 \wedge \neg\varphi_2$

▶ **Theorem 15.** *Let α be an* FLIF$^{\text{io}}$ *expression over schema $\mathcal{S}$. There exists an $I(\alpha)$-executable FO formula φ_α over $\mathcal{S}$, with* $\text{FV}(\varphi_\alpha) = \text{FV}(\alpha)$*, such that for every D and ν_{in}, we have $Eval_\alpha(D, \nu_{\text{in}}) = Eval_{\varphi_\alpha, I(\alpha)}(D, \nu_{\text{in}})$. The length of φ_α is linear in the length of α.*

▶ **Example 16.** To illustrate the proof, consider the expression $R(x; y, u) ; S(x; z, u)$. Procedurally, this expression first retrieves a (y, u)-binding from R for the given x. It proceeds to retrieve a (z, u)-binding from S for the given x, effectively overwriting the previous binding for u. Thus, a correct translation into executable FO is $(\exists u\, R(x; y, u)) \wedge S(x; z, u)$.

For another example, consider the assignment $(x := y)$. This translates to $x = y$ considered as a $\{y\}$-executable formula. The equality test $(x = y)$ also translates to $x = y$, but considered as an $\{x, y\}$-executable formula.

Proof. Table 2 shows the translation, which is almost an isomorphic embedding, except for the case of composition. The correctness of the translation for composition again hinges on inertia and input determinacy. ◀

Notably, in the proof of Theorem 13, we do not need the intersection operation. Hence, by translating FLIF$^{\text{io}}$ to executable FO and then back to FLIF$^{\text{io}}$, we obtain that intersection is redundant in FLIF$^{\text{io}}$, in the following sense:

▶ **Corollary 17.** *For every* FLIF$^{\text{io}}$ *expression α there exists a* FLIF$^{\text{io}}$ *expression α' with the following properties:*

1. *α' does not use the intersection operation.*
2. *$I(\alpha') = I(\alpha)$.*
3. *$O(\alpha') \supseteq O(\alpha)$.*
4. *For every D and ν_{in}, we have $Eval_\alpha(D, \nu_{\text{in}}) = \pi_{\text{FV}(\alpha)}(Eval_{\alpha'}(D, \nu_{\text{in}}))$.*

▶ Remark 18. One may wonder whether the above corollary directly follows from the equivalence between $\alpha_1 \cap \alpha_2$ and $\alpha_1 - (\alpha_1 - \alpha_2)$. While these two expressions are semantically equivalent and have the same input variables, they do not have the same output variables, so a simple inductive proof eliminating intersection while preserving the guarantees of the above corollary does not work. Moreover, the corollary continues to hold for the positive fragment of FLIF$^{\text{io}}$ (without the difference operation). Indeed, positive FLIF$^{\text{io}}$ can be translated into executable FO without negation, which can then be translated into positive FLIF$^{\text{io}}$ without intersection.

5 Relational algebra plans for io-disjoint FLIF

In this section we show how the evaluation problem for $\mathrm{FLIF}^{\mathrm{io}}$ expressions can be solved in a very direct manner, using a translation into a particularly simple form of relational algebra plans.

We generalize the evaluation problem so that it can take a set of valuations as input, rather than just a single valuation. Formally, for an $\mathrm{FLIF}^{\mathrm{io}}$ expression α over database schema $\mathcal{S}$, an instance D of $\mathcal{S}$, and a set N of valuations on $I(\alpha)$, we want to compute $\mathit{Eval}_\alpha(D,N) := \bigcup\{\mathit{Eval}_\alpha(D,\nu_{\mathrm{in}}) \mid \nu_{\mathrm{in}} \in N\}$.

Viewing variables as attributes, we can view a set of valuations on a finite set of variables Z, like the set N above, as a relation with relation schema Z. Consequently, it is convenient to use the named perspective of the relational algebra [2], where every expression has an output relation schema (a finite set of attributes; variables in our case). We briefly review the well-known operators of the relational algebra and their behavior on the relation schema level:

- Union and difference are allowed only on relations with the same relation schema.
- Natural join ($\bowtie$) can be applied on two relations with relation schemas Z_1 and Z_2, and produces a relation with relation schema $Z_1 \cup Z_2$.
- Projection (π) produces a relation with a relation schema that is a subset of the input relation schema.
- Selection (σ) does not change the schema.
- Renaming will not be needed. Instead, however, to accommodate the assignment expressions present in FLIF, we will need the generalized projection operator that adds a new attribute with the same value as an existing attribute, or a constant. Let N be a relation with relation schema Z, let $y \in Z$, and let x be a variable not in Z. Then

$$\pi_{Z,x:=y}(N) = \{\nu[x := \nu(y)] \mid \nu \in N\}$$
$$\pi_{Z,x:=c}(N) = \{\nu[x := c] \mid \nu \in N\}$$

Plans are based on *access methods*, which have the following syntax and semantics. Let $R(\bar{x};\bar{y})$ be an atomic $\mathrm{FLIF}^{\mathrm{io}}$-expression. Let X be the set of variables in $\bar{x}$ and let Y be the set of variables in $\bar{y}$ (in particular, X and Y are disjoint). Let N be a relation with a relation schema Z that contains X but is disjoint from Y. Let D be a database instance. We define the result of the *access join* of N with $R(\bar{x};\bar{y})$, evaluated on D, to be the following relation with relation schema $Z \cup Y$:

$$N \overset{\text{access}}{\bowtie} R(\bar{x};\bar{y}) := \{\nu \text{ valuation on } Z \cup Y \mid \nu|_Z \in N \text{ and } \nu(\bar{x}) \cdot \nu(\bar{y}) \in D(R)\}$$

This result relation can clearly be computed respecting the limited access pattern on R. Indeed, we iterate through the valuations in N, feed their X-values to the source R, and extend the valuations with the obtained Y-values.

Formally, over any database schema $\mathcal{S}$ and for any finite set of variables I, we define a *plan over $\mathcal{S}$ with input variables I* as an expression that can be built up as follows:

- The special relation name *In*, with relation schema I, is a plan.
- If $R(\bar{x};\bar{y})$ is an atomic $\mathrm{FLIF}^{\mathrm{io}}$ expression over $\mathcal{S}$, with sets of variables X and Y as above, and E is a plan with output relation schema Z as above, then also $E \overset{\text{access}}{\bowtie} R(\bar{x};\bar{y})$ is a plan, with output relation schema $Z \cup Y$.
- Plans are closed under union, difference, natural join, and projection.

Given a database instance D, a set N of valuations on I, and a plan E with input variables I, we can instantiate the relation name *In* by N and evaluate E on (D, N) in the obvious manner. We denote the result by $E(D, N)$.

We establish:

▶ **Theorem 19.** *For every* $\mathrm{FLIF}^{\mathrm{io}}$ *expression α over database schema $\mathcal{S}$ there exists a plan E_α over $\mathcal{S}$ with input variables $I(\alpha)$, such that $Eval_\alpha(D, N) = E_\alpha(D, N)$, for every instance D of $\mathcal{S}$ and set N of valuations on $I(\alpha)$.*

▶ **Example 20.**

- A plan for $R(x; y) \,;\, S(y; z)$ is $(In \overset{\text{access}}{\bowtie} R(x; y)) \overset{\text{access}}{\bowtie} S(y; z)$.
- A plan for $R(x_1; y, u) \,;\, S(x_2, y; z, u)$ is

$$\pi_{x_1,x_2,y}(In \overset{\text{access}}{\bowtie} R(x_1; y, u)) \overset{\text{access}}{\bowtie} S(x_2, y; z, u).$$

- Recall the expression $R(x; y_1) \cap S(x; y_2)$ from Example 8, which has input variables $\{x, y_1, y_2\}$ and no output variables. A plan for this expression is

$$(\pi_{x,y_2}(In) \overset{\text{access}}{\bowtie} R(x; y_1)) \bowtie In \;\cap\; (\pi_{x,y_1}(In) \overset{\text{access}}{\bowtie} S(x; y_2)) \bowtie In.$$

  The joins with *In* ensure that the produced output values are equal to the given input values.

Proof. To prove the theorem we need a stronger induction hypothesis, where we allow N to have a larger relation schema $Z \supseteq I(\alpha)$, while still being disjoint with $O(\alpha)$. The claim then is that

$$E_\alpha(D, N) = \{\nu \text{ on } Z \cup O(\alpha) \mid \nu|_{\mathrm{FV}(\alpha)} \in Eval_\alpha(D, \nu|_{I(\alpha)})\}.$$

The base cases are clear. If α is $R(\bar{x}; \bar{y})$, then E_α is $In \overset{\text{access}}{\bowtie} R(\bar{x}; \bar{y})$ for E_α. If α is $(x = y)$, then E_α is the selection $\sigma_{x=y}(In)$. If α is $(x := y)$, then E_α is the generalized projection $\pi_{y,x:=y}(In)$.

In what follows we use the following notation. Let P and Q be plans. By $Q(P)$ we mean the plan obtained from Q by substituting P for *In*.

Suppose α is $\alpha_1 \,;\, \alpha_2$. Plan E_{α_1}, obtained by induction, assumes an input relation schema that contains $I(\alpha_1)$ and is disjoint from $O(\alpha_1)$. Since $I(\alpha) = I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1))$, $I(\alpha_1) \cap O(\alpha_1) = \emptyset$, and Z is disjoint from $O(\alpha) = O(\alpha_1) \cup O(\alpha_2)$, we can apply E_{α_1} with input relation schema Z. Let P_1 be the plan $\pi_{Z - O(\alpha_2)}(E_{\alpha_1})$. Then E_α is the plan $E_{\alpha_2}(P_1)$. (One can again verify that this is a legal plan.)

Next, suppose α is $\alpha_1 \cup \alpha_2$. Then $I(\alpha) = I(\alpha_1) \cup I(\alpha_2)$, which is disjoint from $O(\alpha_1) = O(\alpha_2)$ (compare Proposition 12). Hence for E_α we can simply take the plan $E_{\alpha_1} \cup E_{\alpha_2}$.

Next, suppose α is $\alpha_1 \cap \alpha_2$. Note that $I(\alpha) = I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$. Now E_α is

$$E_{\alpha_1}(\pi_{I(\alpha)-O(\alpha_1)}(In)) \bowtie In \;\cap\; E_{\alpha_2}(\pi_{I(\alpha)-O(\alpha_2)}(In)) \bowtie In.$$

Finally, suppose α is $\alpha_1 - \alpha_2$. Then E_α is

$$E_{\alpha_1} - (E_{\alpha_2}(\pi_{I(\alpha)-O(\alpha_2)}(In)) \bowtie In.$$

In general, in the above translations, we follow the principle that the result of a subplan E_{α_i} must be joined with *In* whenever $O(\alpha_i)$ may intersect with $I(\alpha)$. ◀

▶ Remark 21. When we extend plans with assignment statements such that common expressions can be given a name [5], the translation given in the above proof leads to a plan E_α of size linear of the length of α. Each time we do a substitution of a subexpression for *In* in the proof, we first assign a name to the subexpression and only substitute the name.

6 Conclusion

Nash and Ludäscher [15] deserve credit for having come up with executable FO as a beautiful declarative query language that strikes a perfect balance between first-order logic expressiveness and the limitations imposed by the access patterns on the information sources. On the other hand, relational algebra plans are more operational and rather low-level. We think of FLIF as an intermediate language between the two levels. FLIF is still declarative, as it is still a logic, be it an algebraic one. On the other hand FLIF is also operational, in view of its dynamic semantics akin to dynamic logics [13] and navigational graph query languages. For us, the main novelty of FLIF lies in the mechanism of input and output variables, and the law of inertia.

The book by Benedikt et al. [5] stands as an authoritative reference on the topic of querying under limited access patterns. Remarkably, Benedikt et al. do not follow Nash and Ludäscher's proposal, but use their own, quite different notion of executable first-order query. This notion involves a two-step process where, first, an executable UCQ (union of conjunctive queries) retrieves a set of tuples from the sources, which is then filtered by a first-order condition that is "executable for membership". The filter condition must be expressed in a range-restricted version of first-order logic. In a result similar to our Theorem 19, Benedikt et al. then proceed to show [5, Theorem 3.4] that their executable FO queries are equivalent in expressive power to plans. We feel that our work makes a contribution, enabled by the LIF perspective, by providing a more declarative formalism, a simpler format of plans, and more streamlined translations between the languages.

On the other hand we should stress that the main strength of the work by Benedikt et al. lies elsewhere, namely, in matching semantic properties to syntactic restrictions, for a variety of settings and languages. In this respect, we recall the result [5, Theorem 3.9] already mentioned in the Introduction, to the effect that every "access-determined" boolean first-order query has a plan. This result, proven using model-theoretic interpolation, assumes access-determinacy over unrestricted structures (not necessarily finite). It is open whether a similar result holds in restriction to finite structures.

Our three results (Theorems 13, 15 and 19) exploit the good properties enjoyed by io-disjointness of FLIF expressions. However, as far as expressive power is concerned, io-disjointness may not be a real restriction. Indeed, we conjecture that that every FLIF expression is equivalent, modulo variable renaming, to a FLIF$^{\text{io}}$ expression that can use more variables.

Another topic for further research concerns our definition of inputs and outputs of FLIF expressions (Table 1). While guaranteeing the properties of inertia and input determinacy, this definition cannot be complete in this respect, as said properties are undecidable. Yet, the definition may be "locally" optimal in some sense analogous to an optimality result obtained for the notion of controlled formula [10, Proposition 4.3].

Finally, it would be interesting to look more closely into the practical aspects of the plans generated for FLIF$^{\text{io}}$ expressions. We have shown that these plans have linear size, do not need renaming, and the only joins are natural joins. Does this lead to more efficiency or better optimizability?

In closing, we note that querying under limited access patterns has applicability beyond traditional data or information sources. For instance in the context of distributed data, when performing tasks involving the composition of external services, functions, or modules, limited access patterns are a way for service providers to protect parts of their data, while still allowing their services to be integrated seamlessly in other applications. Limited access patterns also have applications in active databases, where we like to think of FLIF as an analogue of Active XML [1] for the relational data model.

References

1 S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *The VLDB Journal*, 17(5):1019–1040, 2008.

2 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

3 R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017.

4 R. Angles, P. Barceló, and G. Rios. A practical query language for graph DBs. In L. Bravo and M. Lenzerini, editors, *Proceedings 7th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 1087 of *CEUR Workshop Proceedings*, 2013.

5 M. Benedikt, J. Leblay, B. ten Cate, and E. Tsamoura. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation.* Morgan & Claypool, 2016.

6 M. Benedikt, B. ten Cate, and E. Tsamoura. Generating plans from proofs, 2016.

7 A. Calì, D. Calvanese, and D. Martinenghi. Dynamic query optimization under access limitations and dependencies. *Journal of Universal Computer Science*, 15(1):33–62, 2009.

8 A. Calì, D. Martinenghi, I. Razon, and M. Ugarte. Querying the deep web: Back to the foundations. In J.L. Reutter and D. Srivastava, editors, *Proceedings 11th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 1912 of *CEUR Workshop Proceedings*, 2017.

9 A. Calì and M. Ugarte. On the complexity of query answering under access limitations: A computational formalism. In D. Olteanu and B. Poblete, editors, *Proceedings 12th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 2100 of *CEUR Workshop Proceedings*, 2018.

10 W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *Proceedings 33th ACM Symposium on Principles of Database Systems*, pages 51–62, 2014.

11 G.H.L. Fletcher, M. Gyssens, D. Leinders, D. Surinx, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. *Information Sciences*, 298:390–406, 2015.

12 J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.

13 D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.

14 L. Libkin, W. Martens, and D. Vrgoč. Quering graph databases with XPath. In *Proceedings 16th International Conference on Database Theory.* ACM, 2013.

15 A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *Proceedings 23th ACM Symposium on Principles of Database Systems*, pages 307–318, 2004.

16 J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.

17 D. Surinx, G.H.L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs using transitive closure. *Logic Journal of the IGPL*, 23(5):759–788, 2015.

18 E. Ternovska. Recent progress on the algebra of modular systems. In J.L. Reutter and D. Srivastava, editors, *Proceedings 11th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 1912 of *CEUR Workshop Proceedings*, 2017.

19 E. Ternovska. An algebra of modular systems: static and dynamic perspectives. In A. Herzig and A. Popescu, editors, *Frontiers of Combining Systems: Proceedings 12th FroCos*, volume 11715 of *Lecture Notes in Artificial Intelligence*, pages 94–111. Springer, 2019.

A Dichotomy for Homomorphism-Closed Queries on Probabilistic Graphs

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

İsmail İlkan Ceylan
University of Oxford, United Kingdom

Abstract

We study the problem of probabilistic query evaluation (PQE) over probabilistic graphs, namely, tuple-independent probabilistic databases (TIDs) on signatures of arity two. Our focus is the class of queries that is closed under homomorphisms, or equivalently, the *infinite* unions of conjunctive queries, denoted UCQ^∞. Our main result states that all *unbounded* queries in UCQ^∞ are #P-hard for PQE. As *bounded* queries in UCQ^∞ are already classified by the dichotomy of Dalvi and Suciu [17], our results and theirs imply a complete dichotomy on PQE for UCQ^∞ queries over probabilistic graphs. This dichotomy covers in particular all fragments in UCQ^∞ such as *negation-free (disjunctive) Datalog*, *regular path queries*, and a large class of *ontology-mediated queries* on arity-two signatures. Our result is shown by reducing from counting the valuations of positive partitioned 2-DNF formulae (#PP2DNF) for some queries, or from the source-to-target reliability problem in an undirected graph (#U-ST-CON) for other queries, depending on properties of minimal models.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Tuple-independent database, #P-hardness, recursive queries, homomorphism-closed queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.5

Related Version A full version of the paper containing all missing proofs is available at [3], `https://arxiv.org/abs/1910.02048`.

Funding This work was supported by the UK EPSRC grant EP/R013667/1.

1 Introduction

The management of *uncertain and probabilistic data* is an important problem in many applications, e.g., automated knowledge base construction [19, 25, 28], data integration from diverse sources, predictive and stochastic modeling, applications based on (error-prone) sensor readings, etc. To represent probabilistic data, the most basic model is that of tuple-independent *probabilistic databases (TIDs)* [33]. In TIDs, every fact of the database is viewed as an independent random variable, and is either kept or discarded according to some probability. Hence, a TID induces a probability distribution over all *possible worlds*, that is, all possible subsets of the database. The central inference task for TIDs is then *probabilistic query evaluation* (PQE): Given a query Q, compute the probability of Q relative to a TID $\mathcal{I}$, i.e., the total probability of the possible worlds where Q is satisfied.

Dalvi and Suciu [17] obtained a dichotomy for PQE on *unions of conjunctive queries (UCQs)*, measured in *data complexity*, i.e., as a function of the input TID and with the query being fixed. They have shown that PQE can be solved in polynomial time for some UCQs (called *safe*), and that it is #P-hard for all other UCQs (called *unsafe*). Their result was the foundation of many other studies of the complexity of PQE [2, 14, 21, 27, 29, 30, 32].

Despite this extensive research on TIDs, there is little known about PQE for monotone query languages beyond UCQs. In particular, only few results are known for languages featuring *recursion*, which is an essential ingredient in many applications: it is unknown if PQE admits a dichotomy for Datalog, or for ontology-mediated queries [13].

In this work, we focus on a large class of queries beyond first-order: we study the queries that are *closed under homomorphisms*. We denote the class of such queries by UCQ^∞ as they are equivalent to *infinite* unions of conjunctive queries. We distinguish between *bounded* UCQ^∞ queries, which are logically equivalent to a UCQ, and *unbounded* UCQ^∞ queries, which cannot be expressed as a UCQ. Notably, UCQ^∞ captures (negation-free) disjunctive Datalog, regular path queries (RPQs) and a large class of ontology-mediated queries.

We study the framework of *probabilistic graphs*, i.e., probabilistic databases where all relations have *arity two.* Arity-two relations are the formalism used in description logics, and in works on knowledge graphs and information extraction such as NELL [28], Yago [25], and Google's Knowledge Vault [19]. In these contexts, we wish to evaluate unbounded queries on the data, e.g., RPQs or other UCQ^∞ queries, while taking into account its uncertainty. Therefore, we study the complexity of query evaluation on probabilistic graphs, and ask if PQE for the class UCQ^∞ admits a data complexity dichotomy in this case.

The main result of this paper is to show that PQE is #P-hard for *any unbounded* UCQ^∞ *query* over probabilistic graphs. Our result thus implies a dichotomy on PQE for UCQ^∞ over such graphs: as *bounded* UCQ^∞ queries are equivalent to UCQs, they are already classified by Dalvi and Suciu, and we show that all other UCQ^∞ queries are unsafe, i.e., the PQE problem is #P-hard for them. Of course, it is not surprising that *some* unbounded queries in UCQ^∞ are unsafe for similar reasons as unsafe UCQs, but the challenge is to show hardness for *every* unbounded UCQ^∞ query. We conjecture that the same result holds also on arbitrary arity signatures, but we leave this question open, as we explain in the Conclusion.

The proof has two main parts: First, we study UCQ^∞ queries with a model featuring a so-called *non-iterable edge.* For all such queries, we reduce from the problem of counting the valuations of positive partitioned 2-DNF formulae (#PP2DNF). Second, for other unbounded queries in UCQ^∞, we reduce from the source-to-target reliability problem in an undirected graph (#U-ST-CON): this second step is harder and relies on a study of minimal models.

Related work. Research on probabilistic databases is a well-established field; see e.g. [33]. The first dichotomy for queries on such databases was shown by Dalvi and Suciu [16]: a self-join-free conjunctive query is safe if it is *hierarchical*, and #P-hard otherwise. They then extended this result to a dichotomy on all UCQs [17]. Beyond UCQs, partial dichotomy results are known for some queries with negation [21], with disequality ($\neq$) joins in the queries [29], or with inequality ($<$) joins [30]. Some results are known for extended models, e.g., the dichotomy of Dalvi and Suciu has been lifted from TIDs to *open-world probabilistic databases* [14]. However, we are not aware of dichotomies in the probabilistic database literature that apply to Boolean queries beyond first-order logic, or to queries with fixpoints.

Query evaluation on probabilistic graphs has also been studied in the context of *ontology-mediated queries* (OMQs) [27, 9, 10]. An OMQ is a composite query that typically consists of a UCQ and an *ontology*, i.e., a logical theory on an arity-two signature. The only classification result on PQE for OMQs beyond FO-rewritable languages is given for the description logic $\mathcal{ELI}$ [27]. This result applies to a class of queries that go beyond first-order logic. Our work generalizes this result (Theorem 6 of [27]) by showing hardness for any unbounded UCQ^∞, not just the ones expressible as OMQs based on $\mathcal{ELI}$. Part of our techniques (Section 4) are related to theirs, but the bulk of our proof (Sections 5 and 6) uses new techniques, the need for which had in fact been overlooked[1] in [26, 27]. Our proof thus completes the proof of Theorem 6 in [27], and generalizes it to all unbounded UCQ^{∞}.

Paper structure. We introduce preliminaries in Section 2, and formally state our result in Section 3. We prove the result in the rest of the paper. We first deal in Section 4 with the case of queries having a model with a non-iterable edge (reducing from #PP2DNF), then argue in Section 5 that unbounded queries must have a model with a minimal tight edge, before explaining in Section 6 how to use this (when the edge is iterable) to reduce from #U-ST-CON. We then conclude in Section 7. Detailed proofs can be found in the full version [3].

2 Preliminaries

Vocabulary. We consider a *relational signature* σ which is a set of *predicates*. In this work, the signature is required to be *arity-two*, i.e., consist *only* of predicates of arity two. Our results can easily be extended to signatures with relations having predicates of arity one and two (see the full version [3]), as is more common in some contexts such as description logics.

A *σ-fact* is an expression of the form $F = R(a, b)$ where R is a predicate and a, b are constants. By a slight abuse of terminology, we call F a *unary* fact if $a = b$, and a *non-unary fact* otherwise. A *σ-atom* is defined in the same way with variables instead of constants. For brevity, we will often talk about a *fact* or an *atom* when σ is clear from context. We also speak of *R-facts* or *R-atoms* to specifically refer to facts or atoms that use the predicate R.

It will be convenient to write $\sigma^{\leftrightarrow}$ the arity-two signature consisting of the relations of σ and of the relations R^- for $R \in \sigma$, with a semantics that we define below.

Database instances. A *database instance over* σ, or a *σ-instance*, is a set of facts over σ. All instances considered in this paper are finite. The *domain* of a fact F, denoted $\mathrm{dom}(F)$, is the set of constants that appear in F, and the *domain* of an instance I, denoted $\mathrm{dom}(I)$, is the set of constants that appear in I, i.e., the union of the domains of its facts.

Every σ-instance I can be seen as a $\sigma^{\leftrightarrow}$-instance consisting of all the σ-facts in I, and all the facts $R^-(b, a)$ for each fact $R(a, b)$ of I. Thus, whenever we consider a σ-instance I, choose some $a \in \mathrm{dom}(I)$, and say, e.g., that we consider all $\sigma^{\leftrightarrow}$-facts of the form $F = R(a, b)$ in I, we mean all unary facts $S(a, a)$ with some $S \in \sigma$, all facts $S(a, b)$ of I with some $S \in \sigma$ and $b \in \mathrm{dom}(I)$, and also all facts $S^-(a, b)$ of I for some $S \in \sigma$ and $b \in \mathrm{dom}(I)$, that is, facts of the form $S(b, a)$. If we say that, for one such fact $F_0 = R(a, b_0)$, we create the fact $R(a', b_0)$ for some $a' \in \mathrm{dom}(I)$, it means that we create $S(a', b_0)$ if $F_0 = S(a, b_0)$ with $S \in \sigma$, and $S(b_0, a')$ if $F_0 = S^-(a, b_0)$ with $S \in \sigma$.

The *Gaifman graph* of an instance I is the undirected graph having $\mathrm{dom}(I)$ as vertex set, and having an edge $\{u, v\}$ between any two $u \neq v$ in $\mathrm{dom}(I)$ that co-occur in some fact of I. An instance is *connected* if its Gaifman graph is connected. We then call $\{u, v\}$ an (undirected) *edge* of I, and the facts that *realize* the undirected edge e are the σ-facts of I whose domain is a subset of $\{u, v\}$. Note that a fact of the form $R(u, u)$ realizes all

[1] Specifically, we identified a gap in the proofs of Theorem 6 of [27] and Theorem 5.31 of [26] concerning a subtle issue of "back-and-forth" matches. We have communicated this with the authors of [26, 27], which they kindly confirmed. The problem is related to the use of inverse roles of $\mathcal{ELI}$, so we believe that it does not occur for the description logic $\mathcal{EL}$.

edges involving u. Slightly abusing notation, we say that an *ordered* pair $e = (u, v)$ is a (directed) *edge* of I if $\{u, v\}$ is an edge of the Gaifman graph. We then talk about the facts that *realize* the directed edge e as the $\sigma^{\leftrightarrow}$-facts of I defined as follows: all unary σ-facts of the form $R(u, u)$ and $R(v, v)$, all σ-facts $S(u, v)$ of I with $S \in \sigma$, and one fact $S^{-}(u, v)$ for every σ-fact $S(v, u)$ of I with $S \in \sigma$. Note that the facts that *realize* the directed edge (v, u) would correspond to the same σ-facts of I, but they are not the same $\sigma^{\leftrightarrow}$ facts: specifically, each relation $S \in \sigma$ has been exchanged with its reverse relation S^{-} in non-unary facts.

In the course of our proofs, we will often modify instances in a specific way, which we call *copying* an edge. Let I be an instance, let (u, v) be a directed edge of I, and let u', v' be any elements of I. If we say that we *copy* the edge e on (u', v'), it means that we modify I to add a copy of each fact realizing the edge e, but using u' and v' instead of u and v. Specifically, we create $S(u', v')$ for all σ-facts of the form $S(u, v)$ in I, we create $S(v', u')$ for all σ-facts of the form $S(v, u)$ in I, and we create $S(u', u')$ and $S(v', v')$ for all σ-facts respectively of the form $S(u, u)$ and $S(v, v)$ in I. Of course, if some of these facts already exist, they are not created again. Note that (u', v') is an edge of I after this process.

An instance I is a *subinstance* of another instance I' if $I \subseteq I'$, and I is a *proper subinstance* of I' if $I \subset I'$. Given a set $S \subseteq \mathrm{dom}(I)$ of domain elements, the subinstance of I *induced* by S is the instance formed of all the facts $F \in I$ such that $\mathrm{dom}(F) \subseteq S$.

A *homomorphism* from an instance I to an instance I' is a function h from $\mathrm{dom}(I)$ to $\mathrm{dom}(I')$ such that, for every fact $R(a, b)$ of I, the fact $R(h(a), h(b))$ is a fact of I'. In particular, whenever $I \subseteq I'$ then I has a homomorphism to I'. An *isomorphism* is a bijective homomorphism whose inverse is also a homomorphism.

Query languages. Throughout this work, we focus on Boolean queries. A (Boolean) *query* over a signature σ is a function from σ-instances to Booleans. An instance I *satisfies* a query Q (or Q *holds* on I, or I is a *model* of Q), written $I \models Q$, if Q returns true when applied to I; otherwise, I *violates* Q. We say that two queries Q_1 and Q_2 are *equivalent* if for any instance I, we have $I \models Q_1$ iff $I \models Q_2$. In this work, we study the class UCQ^{∞} of queries that are *closed under homomorphisms* (also called *homomorphism-closed*), i.e., if I satisfies the query and I has a homomorphism to I' then I' also satisfies the query. Note that queries closed under homomorphisms are in particular *monotone*, i.e., if I satisfies the query and $I \subseteq I'$, then I' also satisfies the query.

One well-known subclass of UCQ^{∞} is *bounded* UCQ^{∞}: every bounded query in UCQ^{∞} is logically equivalent to a *union of conjunctive query* (UCQ), without negation or inequalities. Recall that a *conjunctive query* (CQ) is an existentially quantified conjunctions of atoms, and a UCQ is a disjunction of CQs. For brevity, we omit existential quantification when writing UCQs, and abbreviate conjunction with a comma. The other UCQ^{∞} queries are called *unbounded*, and they can be seen as an infinite disjunction of CQs, with each disjunct corresponding to a model of the query.

A natural language captured by UCQ^{∞} is *Datalog*, again without negation or inequalities. A Datalog program defines a signature of *intensional predicates*, including a 0-ary predicate Goal(), and consists of a set of *rules* which explain how intensional facts can be *derived* from other intensional facts and from facts of the instance (called *extensional*). The interpretation of the intensional predicates is defined by taking the (unique) least fixpoint of applying the rules, and the query holds iff the Goal() predicate can be derived. For formal definitions of this semantics, see, e.g., [1]. As Datalog queries are homomorphism-closed, we can see each Datalog program as a UCQ^{∞}, with the disjuncts intuitively corresponding to *derivation trees* for the program. However, note that there are homomorphism-closed queries that are not expressible in Datalog [18].

Ontology-mediated queries or OMQs [8] are another subclass of UCQ^∞. An OMQ is a pair $(Q, \mathcal{T})$, where Q is a UCQ, and $\mathcal{T}$ is an ontology. A database instance I *satisfies* an OMQ $(Q, \mathcal{T})$ if the instance I and the logical theory $\mathcal{T}$ entail the query Q in the standard sense – see, e.g., [8], for details. There are ontological languages for OMQs based on description logics [5] and on existential rules [11, 12]. Many such OMQs can be equivalently expressed as a query in Datalog or in disjunctive Datalog on an arity-two signature [8, 20, 23], thus falling in the class UCQ^∞. In particular, this is the case of any OMQ involving negation-free $\mathcal{ALCHI}$ (Theorem 6 of [8]), and of fragments of $\mathcal{ALCHI}$, e.g., $\mathcal{ELHI}$, and $\mathcal{ELI}$ as in [27].

Probabilistic query evaluation. We study the problem of probabilistic query evaluation over tuple-independent probabilistic databases. A *tuple-independent probabilistic database (TID)* over a signature σ is a pair $\mathcal{I} = (I, \pi)$ of a σ-instance I, and of a function π that maps every fact F to a probability $\pi(F)$, given as a rational number in $[0, 1]$. Formally, a TID $\mathcal{I} = (I, \pi)$ defines the following probability distribution over all *possible worlds* $I' \subseteq I$:

$$\pi(I') := \left(\prod_{F \in I'} \pi(F)\right) \times \left(\prod_{F \in I' \setminus I} (1 - \pi(F))\right).$$

Then, given a TID $\mathcal{I} = (I, \pi)$, the probability of a query Q relative to $\mathcal{I}$, denoted $\mathrm{P}_{\mathcal{I}}(Q)$, is given by the sum of the probabilities of the possible worlds that satisfy the query:

$$\mathrm{P}_{\mathcal{I}}(Q) := \sum_{I' \subseteq I, I' \models Q} \pi(I').$$

The *probabilistic query evaluation problem* (PQE) for a query Q, written $\mathrm{PQE}(Q)$, is then the task of computing $\mathrm{P}_{\mathcal{I}}(Q)$ given a TID $\mathcal{I}$ as input.

Complexity background. FP is the class of functions $f : \{0,1\}^* \mapsto \{0,1\}^*$ computable by a polynomial-time deterministic Turing machine. The class #P, introduced by Valiant [34], contains the computation problems that can be expressed as the number of accepting paths of a nondeterministic polynomial-time Turing machine. Equivalently, a function $f : \{0,1\}^* \mapsto \mathbb{N}$ is in #P if there exists a polynomial $p : \mathbb{N} \mapsto \mathbb{N}$ and a polynomial-time deterministic Turing machine M such that for every $x \in \{0,1\}^*$, it holds that $f(x) = |\{y \in \{0,1\}^{p(|x|)} \mid M \textit{ answers 1 on the input } (x, y)\ \}|$.

For a query Q, we study the *data complexity* of $\mathrm{PQE}(Q)$, which is measured as a function of the input instance I, i.e., the signature and Q are fixed. For a large class of queries, in particular for any UCQ Q, the problem $\mathrm{PQE}(Q)$ is in the complexity class $\mathrm{FP}^{\#P}$: we can use a nondeterministic Turing machine to guess a possible world according to the probability distribution of the TID (i.e., each possible world is obtained in a number of runs proportional to its probability), and then check in polynomial time data complexity if Q holds, with polynomial-time postprocessing to renormalize the number of runs to a probability. Our goal in this work is to show that the problem is also #P-hard.

To show #P-hardness, we use *polynomial-time Turing reductions* [15]. A function f is #P-complete under polynomial time Turing reductions if it is in #P and every $g \in$ #P is in FP^f. Polynomial-time Turing reductions are the most common reductions for the class #P and they are the reductions used to show #P-hardness in the dichotomy of Dalvi and Suciu [17], so we use them throughout this work.

Problems. We will show hardness by reducing from two well-known #P-hard problems. For some queries, we reduce from #PP2DNF [31], which is a standard tool to show hardness of unsafe UCQs. The original problem uses Boolean formulas; here, we give an equivalent rephrasing in terms of bipartite graphs.

▶ **Definition 2.1.** *Given a bipartite graph $H = (A, B, C)$ with edges $C \subseteq A \times B$, a* possible world *of H is a pair $\omega = (A', B')$ with $A' \subseteq A$ and $B' \subseteq B$. We call the possible world* good *if it is not an independent set, i.e., if one vertex of A' and one vertex of B' are adjacent in C; and call it* bad *otherwise. The* positive partitioned 2DNF problem (#PP2DNF) *is the following: Given a bipartite graph, compute how many of its possible worlds are good.*

It will be technically convenient to assume that H is connected. This is clearly without loss of generality, as otherwise the number of good possible worlds is simply obtained as the product of the number of good possible worlds of each connected component of H.

For other queries, we reduce from the *undirected st-connectivity problem* (#U-ST-CON) [31]:

▶ **Definition 2.2.** *The* source-to-target undirected reachability problem (#U-ST-CON) *asks the following: Given an undirected graph G with two distinguished vertices s and t, where each graph edge has probability* 0.5, *determine the probability of obtaining a* good *possible world, i.e., a subgraph ω of G where there is a path from s to t.*

3 Result Statement

The goal of this paper is to extend the dichotomy of Dalvi and Suciu [17] on PQE for UCQs. Their result states:

▶ **Theorem 3.1** ([17]). *Let Q be a UCQ. Then,* PQE(Q) *is either in FP or it is #P-hard.*

We call a UCQ *safe* if PQE(Q) is in FP, and *unsafe* otherwise. This dichotomy characterizes the complexity of PQE for UCQs, but does not apply to other homomorphism-closed queries beyond UCQs. Our contribution, when restricting to the arity-two setting, is to generalize this dichotomy to UCQ$^\infty$, i.e., to *any* query closed under homomorphisms. Specifically, we show that all such queries are intractable unless they are equivalent to a safe UCQ.

▶ **Theorem 3.2.** *Let Q be a* UCQ$^\infty$ *on an arity-two signature. Then, either Q is equivalent to a safe UCQ and* PQE(Q) *is in FP, or it is not and* PQE(Q) *is #P-hard.*

Our result relies on the dichotomy of Dalvi and Suciu for UCQ$^\infty$ queries that are equivalent to UCQs. The key point is then to show intractability for *unbounded* UCQ$^\infty$ queries. Hence, our technical contribution is to show:

▶ **Theorem 3.3.** *Let Q be an unbounded* UCQ$^\infty$ *query on an arity-two signature. Then* PQE(Q) *is #P-hard.*

Examples of unbounded UCQ$^\infty$ queries include many Datalog queries, e.g., the following program with one monadic intensional predicate U on extensional signature R, S, T:

$$R(x,y) \to U(x) \qquad U(x), S(x,y) \to U(y) \qquad U(x), T(x,y) \to \text{Goal}()$$

Thus, our result implies that the PQE problem is #P-hard for all Datalog queries that are not equivalent to a UCQ, which is the case unless the Datalog program is nonrecursive or recursion is *bounded* [24]. Unbounded UCQ$^\infty$ queries also include many regular path queries, such as RS^*T which is equivalent to the Datalog program above.

Effectiveness and uniformity. We do not study if our dichotomy result in Theorem 3.2 is effective, i.e., given a query, the problem of determining whether it is safe or unsafe. The dichotomy of Theorem 3.1 on UCQs is effective via the algorithm of [17]: this algorithm has a super-exponential bound (in the query), with the precise complexity being open. Our dichotomy concerns the very general query language UCQ$^\infty$, and its effectiveness depends on how the input is represented, which we can fix by restricting to a syntactically defined fragment. If we restrict to Datalog queries, it is not clear whether our dichotomy is effective, because it is generally undecidable whether an input Datalog program is bounded [22] – but this, on its own, does not imply undecidability for our dichotomy. However, our dichotomy is effective for more restricted query languages for which boundedness is decidable, e.g., monadic Datalog or its generalization GN-Datalog [7], or C2RPQs [6].

For unsafe queries, we also do not study the complexity of reduction *as a function of the query*, or whether this problem is even decidable. All that matters is that, once the query is fixed, some reduction procedure exists, which can be performed in polynomial time *in the input instance*. Such uniformity problems seem unavoidable, given that our language UCQ$^\infty$ is very general and includes some queries for which non-probabilistic evaluation is not even decidable, e.g., "there is a path from R to T whose length is the index of a Turing machine that halts". We leave for future work the study of the query complexity of our reduction when restricting to better-behaved query languages such as Datalog or RPQs.

Proof outline. Theorem 3.3 is proven in the rest of the paper. There are two cases, depending on the query. We study the first case in Section 4, which covers queries for which we can find a model with a so-called *non-iterable edge*. Intuitively, this is a model where we can make the query false by replacing the edge by a back-and-forth path of some length between two neighboring facts that it connects. For such queries, we can show hardness by a reduction from #PP2DNF, essentially like the hardness proof for the query $Q_0 : R(w,x), S(x,y), T(y,z)$ which is the arity-two variant of the unsafe query of [16, Theorem 5.1]. This hardness proof covers some bounded queries (including Q_0) and some unbounded ones.

In Section 5, we present a new ingredient, to be used in the second case, i.e., when there is no model with a non-iterable edge. We show that any unbounded query must always have a model with a *tight edge*, i.e., an edge where we can make the query false by replacing it by two copies that disconnect its endpoints. What is more, we can find a model with a tight edge which is *minimal* in some sense, which we call a *minimal tight pattern*.

In Section 6, we use minimal tight patterns for the second case, covering unbounded queries that have a minimal tight pattern whose edge is iterable. This applies for all queries to which Section 4 did not apply (and also for some queries to which it did). Here, we reduce from the #U-ST-CON problem, intuitively using the iterable edge for a kind of reachability test, and using the minimality and tightness of the pattern to show the soundness and completeness of the reduction.

4 Hardness with Non-Iterable Edges

In this section, we present the hardness proof for the first case where we can find a model of the query with a *non-iterable edge*. This notion will be defined relative to an *incident pair* of a *non-leaf edge*:

▶ **Definition 4.1.** *Let I be an instance. We say that an element $u \in \mathrm{dom}(I)$ of I is a* leaf *if it occurs in only one undirected edge. We say that an edge (directed or undirected) is a* leaf edge *if one of its elements (possibly both) is a leaf; otherwise, it is a* non-leaf edge.

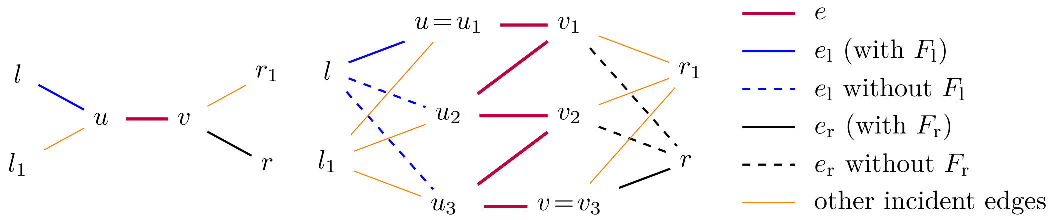

l
l_1
u
v
r_1
r
l
l_1
$u = u_1$
v_1
u_2
v_2
u_3
$v = v_3$
r_1
r
e
e_l (with F_l)
e_l without F_l
e_r (with F_r)
e_r without F_r
other incident edges

Figure 1 Example of iteration from an instance $I_{e,\Pi}$ (left) to $I^3_{e,\Pi}$ (middle). We write $\Pi = (F_\mathrm{l}, F_\mathrm{r})$ and call e_l and e_r the edges of F_l and F_r. Each line represents an edge, realized in general by multiple $\sigma^\leftrightarrow$-facts. A key is given at the right.

Let I be an instance and let $e = (u, v)$ be a non-leaf edge of I. A $\sigma^\leftrightarrow$-fact of I is left-incident *to e if it is of the form $R_\mathrm{l}(l, u)$ with $l \notin \{u, v\}$. It is* right-incident *to e if it is of the form $R_\mathrm{r}(v, r)$ with $r \notin \{u, v\}$. An* incident pair *of e is a pair of $\sigma^\leftrightarrow$-facts $\Pi = (F_\mathrm{l}, F_\mathrm{r})$, where F_l is left-incident to e and F_r is right-incident to e. We write $I_{e,\Pi}$ to denote an instance I with a non-leaf edge e and an incident pair Π of e in I.*

Note that an incident pair chooses two incident *facts* (not edges): this is intuitively because in the PQE problem, we will give probabilities to single facts and not edges. It is clear that every non-leaf edge e must have an incident pair, as we can pick F_l and F_r from the edges incident to u and v which are not e. Moreover, we must have $F_\mathrm{l} \neq F_\mathrm{r}$, and neither F_l nor F_r can be unary facts. However, as the relations R_l and R_r are $\sigma^\leftrightarrow$-relations, we may have $R_\mathrm{l} = R_\mathrm{r}$ or $R_\mathrm{l} = R_\mathrm{r}^-$, and the elements l and r may be equal if the edge (u, v) is part of a triangle with some edges $\{u, w\}$ and $\{v, w\}$.

Let us illustrate the notion of incident pair on an example.

▶ **Example 4.2.** Given an instance $I = R(a, b), T(b, b), S(c, b), R(d, c)$, the edge (b, c) is non-leaf and the only possible incident pair for it is $(R(a, b), R^-(c, d))$.

We can now define the *iteration process* on an instance $I_{e,\Pi}$, which intuitively replaces the edge e by a path of copies of e, keeping the facts of Π at the beginning and end of the path, and copying all other incident facts:

▶ **Definition 4.3.** *Let $I_{e,\Pi}$ be an instance where $e = (u, v)$, $\Pi = (F_\mathrm{l}, F_\mathrm{r})$, $F_\mathrm{l} = R_\mathrm{l}(l, u)$, $F_\mathrm{r} = R_\mathrm{r}(v, r)$, and let $n \geq 1$. The result of performing the n*-th iteration *of e in I relative to Π, denoted $I^n_{e,\Pi}$, is a σ-instance with domain $\mathrm{dom}(I^n_{e,\Pi}) := \mathrm{dom}(I) \cup \{u_2, \ldots, u_n\} \cup \{v_1, \ldots, v_{n-1}\}$, where the new elements are fresh, and where we use u_1 to refer to u and v_n to refer to v for convenience. The facts of $I^n_{e,\Pi}$ are defined by applying the following steps:*

- Copy non-incident facts: *Initialize $I^n_{e,\Pi}$ as the induced subinstance of I on $\mathrm{dom}(I) \setminus \{u, v\}$.*
- Copy incident facts F_l and F_r: *Add F_l and F_r to $I^n_{e,\Pi}$, using u_1 and v_n, respectively.*
- Copy other left-incident facts: *For each $\sigma^\leftrightarrow$-fact $F'_\mathrm{l} = R'_\mathrm{l}(l', u)$ of I that is left-incident to e (i.e., $l' \notin \{u, v\}$) and where $F'_\mathrm{l} \neq F_\mathrm{l}$, add to $I^n_{e,\Pi}$ the fact $R'_\mathrm{l}(l', u_i)$ for each $1 \leq i \leq n$.*
- Copy other right-incident facts: *For each $\sigma^\leftrightarrow$-fact $F'_\mathrm{r} = R'_\mathrm{r}(v, r')$ of I that is right-incident to e (i.e., $r' \notin \{u, v\}$) and where $F'_\mathrm{r} \neq F_\mathrm{r}$, add to $I^n_{e,\Pi}$ the fact $R'_\mathrm{r}(v_i, r')$ for each $1 \leq i \leq n$.*
- Create copies of e: *Copy the edge e (in the sense defined in the Preliminaries) on the following pairs: (u_i, v_i) for $1 \leq i \leq n$, and (u_{i+1}, v_i) for $1 \leq i \leq n - 1$.*

The iteration process is represented in Figure 1. Note that, for $n = 1$, we obtain exactly the original instance. Intuitively, we replace e by a path going back-and-forth between copies of u and v (and traversing e alternatively in one direction and another). The intermediate vertices have the same incident facts as the original endpoints except that we have not copied the left-incident fact and the right-incident fact of the incident pair.

We first notice that larger iterates have homomorphisms back to smaller iterates:

▶ **Observation 4.4.** *For any instance I, for any non-leaf edge e of I, for any incident pair Π for e, and for any $1 \leq i \leq j$, it holds that $I^j_{e,\Pi}$ has a homomorphism to $I^i_{e,\Pi}$.*

Proof. Simply merge $u_i, \ldots, u_j$, and merge $v_i, \ldots, v_j$. ◀

Hence, choosing an instance I that satisfies Q, a non-leaf edge e of I, and an incident pair Π, there are two possible regimes. Either all iterations $I^n_{e,\Pi}$ satisfy Q, or there is some iteration $I^{n_0}_{e,\Pi}$ with $n_0 > 1$ that violates Q (and, by Observation 4.4, all subsequent iterations also do). We call e *iterable* relative to Π in the first case, and *non-iterable* in the second case.

▶ **Definition 4.5.** *A non-leaf edge e of a model I of a query Q is* iterable *relative to an incident pair Π if $I^n_{e,\Pi}$ satisfies Q for each $n \geq 1$; otherwise, it is* non-iterable.

The goal of this section is to show that if a query Q has a model with a non-leaf edge which is not iterable, then PQE(Q) is intractable:

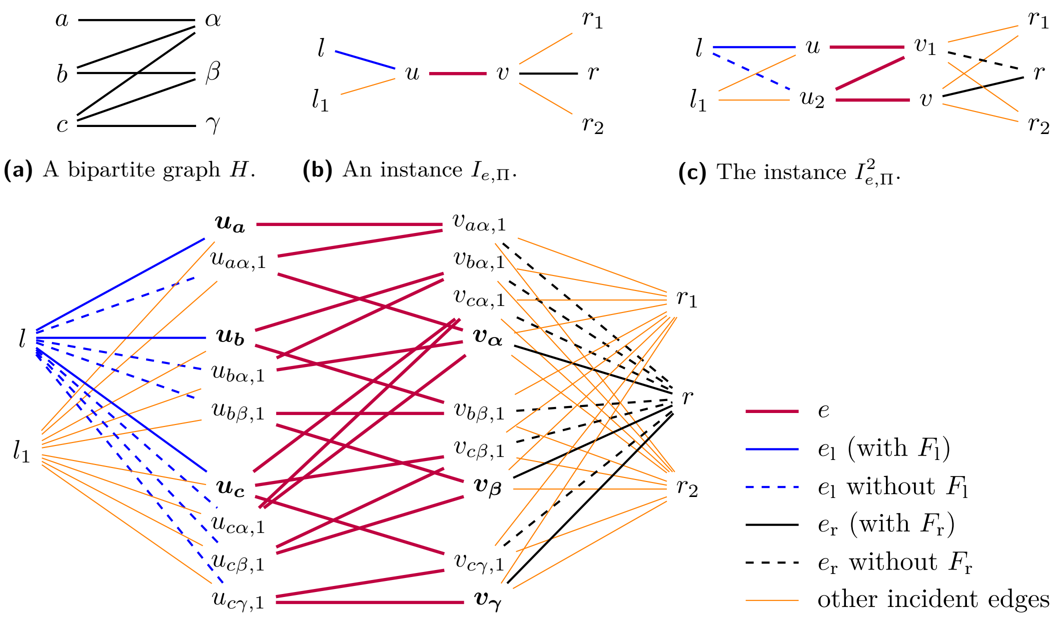
▶ **Theorem 4.6.** *For every UCQ$^\infty$ Q, if Q has a model I with a non-leaf edge e that is non-iterable relative to some incident pair, then PQE(Q) is #P-hard.*

Note that this result applies to arbitrary homomorphism-closed queries, whether they are bounded or not. Recall for instance the unsafe CQ $Q_0 : R(w,x), S(x,y), T(y,z)$. Then, the model $R(a,b), S(b,c), T(c,d)$ has an edge (b,c) which is non-leaf and non-iterable: indeed its iteration with $n = 2$ relative to the only possible incident pair yields $R(a,b), S(b,c'), S(b',c')$, $S(b',c), T(c,d)$ which does not satisfy the query. Thus, Theorem 4.6 also shows that PQE is #P-hard for this query. Of course, Theorem 4.6 is too coarse to show #P-hardness for all unsafe UCQs; for instance, it does not cover $Q'_0 : R(x,x), S(x,y), T(y,y)$, or $Q_1 : (R(w,x), S(x,y)) \vee (S(x,y), T(y,z))$. Theorem 4.6 will nevertheless be sufficient for our purpose of showing hardness for all *unbounded queries*, as we will do in the next sections.

Hence, in the rest of this section, we prove Theorem 4.6. Let $I_{e,\Pi}$ be the instance with the non-iterable edge, and let us take the smallest $n_0 > 1$ such that $I^{n_0}_{e,\Pi}$ violates the query. The idea is to use $I_{e,\Pi}$ and n_0 to show hardness of PQE by reducing from #PP2DNF (Definition 2.1). Thus, let us explain how we can use $I_{e,\Pi}$ to code a bipartite graph H in polynomial time into a TID $\mathcal{I}$. The definition of this coding does not depend on the query Q, but we will use the properties of $I_{e,\Pi}$ and n_0 to argue that it defines a reduction between #PP2DNF and PQE, i.e., there is a correspondence between the possible worlds of H and the possible worlds of $\mathcal{I}$, such that good possible worlds of H are mapped to possible worlds of $\mathcal{I}$ which satisfy Q. Let us first define the coding:

▶ **Definition 4.7.** *Let $I_{e,\Pi}$ be an instance where $e = (u,v)$, $\Pi = (F_\mathrm{l}, F_\mathrm{r})$, $F_\mathrm{l} = R_\mathrm{l}(l,u)$, $F_\mathrm{r} = R_\mathrm{r}(v,r)$, and let $n \geq 1$. Let $H = (A,B,C)$ be a connected bipartite graph. The* coding *of H relative to $I_{e,\Pi}$ and n is a TID $\mathcal{I} = (J,\pi)$ with domain $\mathrm{dom}(J) := (\mathrm{dom}(I) \setminus \{u,v\}) \cup \{u_a \mid a \in A\} \cup \{v_b \mid b \in B\} \cup \{u_{c,2}, \ldots, u_{c,n} \mid c \in C\} \cup \{v_{c,1}, \ldots, v_{c,n-1} \mid c \in C\}$, where the new elements are fresh. The facts of J and the probability mapping π are defined as follows:*

- Copy non-incident facts: *Initialize J as the induced subinstance of I on $\mathrm{dom}(I) \setminus \{u,v\}$.*
- Copy incident facts F_l and F_r: *Add to J the $\sigma^\leftrightarrow$-fact $R_\mathrm{l}(l,u_a)$ for each $a \in A$, and add to J the $\sigma^\leftrightarrow$-fact $R_\mathrm{r}(v_b,r)$ for each $b \in B$.*
- Copy other left-incident facts: *For each $\sigma^\leftrightarrow$-fact $F'_\mathrm{l} = R'_\mathrm{l}(l',u)$ of I that is left-incident to e (i.e., $l' \notin \{u,v\}$) and where $F'_\mathrm{l} \neq F_\mathrm{l}$, add to J the facts $R'_\mathrm{l}(l',u_a)$ for each $a \in A$, and add to J the facts $R'_\mathrm{l}(l',u_{c,j})$ for each $2 \leq j \leq n$ and $c \in C$.*



(a) A bipartite graph H.

(b) An instance $I_{e,\Pi}$.

(c) The instance $I^2_{e,\Pi}$.

(d) Coding of the bipartite graph H relative to $I^2_{e,\Pi}$. Bold elements correspond to vertices of H.

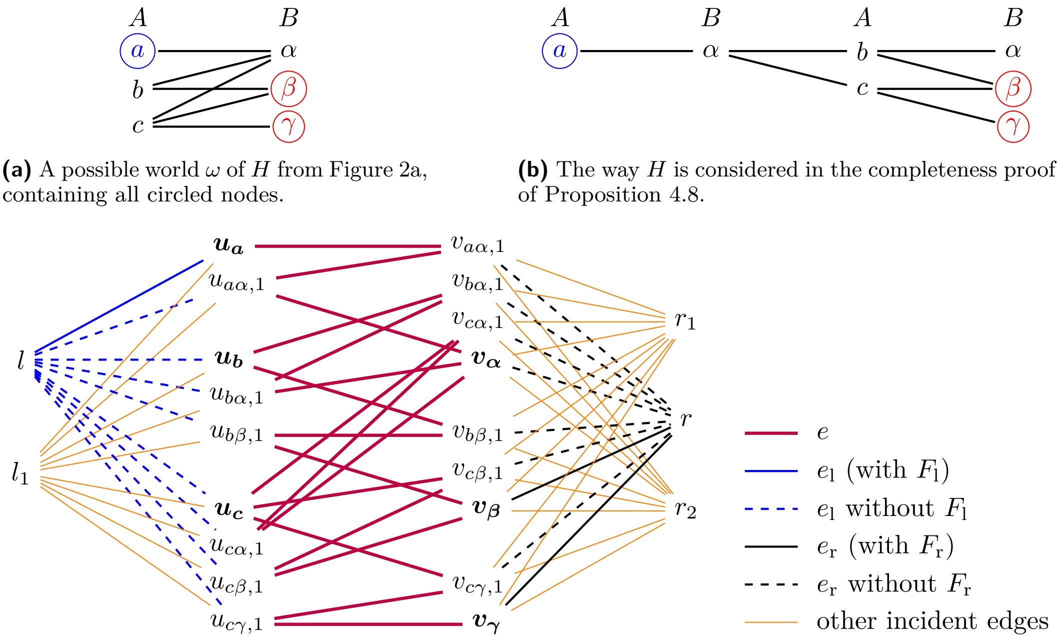
Figure 2 Example of the coding of a bipartite graph H shown in Figure 2a. We encode H relative to an instance $I_{e,\Pi}$ (Figure 2b), with a non-leaf edge e and an incident pair Π. The result $I^2_{e,\Pi}$ of iterating e in I with $n = 2$ (Definition 4.3) is shown in Figure 2c. The coding of H relative to $I_{e,\Pi}$ and $n = 2$ (Definition 4.7) is shown in Figure 2d, with the probabilistic facts being the copies of F_{l} and F_{r} in the edges in solid blue and black. A key explains the colors (bottom right).

- Copy other right-incident facts: *For each $\sigma^{\leftrightarrow}$-fact $F'_{\mathrm{r}} = R'_{\mathrm{r}}(v, r')$ of I that is right-incident to e (i.e., $r' \notin \{u, v\}$) and where $F'_{\mathrm{r}} \neq F_{\mathrm{r}}$, add to J the facts $R'_{\mathrm{r}}(v_b, r')$ for each $b \in B$ and add to J the facts $R'_{\mathrm{r}}(v_{c,j}, r')$ for each $1 \leq j \leq n-1$ and $c \in C$.*
- Create copies of e: *For each $c \in C$ with $c = (a, b)$, copy e on the following pairs: $(u_{c,i}, v_{c,i})$ for $1 \leq i \leq n$, and $(u_{c,i+1}, v_{c,i})$ for $1 \leq i \leq n-1$, where we use $u_{c,1}$ to refer to u_a and $v_{c,n}$ to refer to v_b.*

Finally, we define the function π such that it maps all the facts created in the step "Copy incident facts F_{l} and F_{r}" to 0.5, and all other facts to 1.

Observe how this definition relates to the definition of iteration (Definition 4.3): we intuitively code each edge of the bipartite graph as a copy of the path of copies of e in the definition of the n-th iteration of (u, v). Note also that there are exactly $|A| + |B|$ uncertain facts, by construction. It is clear that, for any choice of $I_{e,\Pi}$ and n, this coding is in polynomial time in H. The result of the coding is illustrated in Figure 2.

We now define the bijection ϕ, mapping each possible world ω of the connected bipartite graph H to a possible world of the TID $\mathcal{I}$. For each vertex $a \in A$, we keep the copy of F_{r} incident to u_a in $\phi(\omega)$ if a is kept in ω, and we do not keep it otherwise; we do the same for v_b, and F_{l}. It is obvious that this correspondence is bijective, and that all possible worlds have the same probability, namely, $0.5^{|A|+|B|}$. Furthermore, we can use ϕ to define a reduction, thanks to the following property:



(a) A possible world ω of H from Figure 2a, containing all circled nodes.

(b) The way H is considered in the completeness proof of Proposition 4.8.

(c) The possible world $\phi(\omega)$ of the coding (Figure 2d) for ω. The edges (l, u_b), (l, u_c), and (v_α, r) are changed to dashed lines, as they correspond to vertices of H that are not kept in ω.

Figure 3 Example for the completeness direction of the proof of Proposition 4.8. Figure 3a shows a bad possible world ω of the bipartite graph. The corresponding possible world of the coding of Figure 2d (using the instance $I^2_{e,\Pi}$ of Figure 2b) is given in Figure 3c. In the proof, we explore H as depicted in Figure 3b to argue that Figure 3c has a homomorphism to $I^5_{e,\Pi}$. A key explains the colors (bottom right).

▶ **Proposition 4.8.** *Let the TID $\mathcal{I} = (J, \pi)$ be the coding of a connected bipartite graph $H = (A, B, C)$ relative to an instance $I_{e,\Pi}$ and to $n \geq 1$ as described in Definition 4.7, and let ϕ be the bijective function defined above from the possible worlds of H to those of $\mathcal{I}$. Then:*

1. *For any* good *possible world ω of H, $\phi(\omega)$ has a homomorphism* from *$I^n_{e,\Pi}$.*
2. *For any* bad *possible world ω of H, $\phi(\omega)$ has a homomorphism* to *$I^{3n-1}_{e,\Pi}$.*

Proof sketch. The first direction is because $\phi(\omega)$ then contains a subinstance isomorphic to $I^n_{e,\Pi}$: keep the facts of the path corresponding to any edge witnessing that ω is good.

The harder part is the second direction as illustrated in Figure 3: when ω is bad, we can show how to "fold back" $\phi(\omega)$, going from the copies of F_{l} to the copies of F_{r}, into the iterate $I^{3n-1}_{e,\Pi}$. This uses the fact that ω is bad, so the copies of F_{l} and F_{r} must be sufficiently far from one another. ◀

Proposition 4.8 allows us to conclude the proof of Theorem 4.6. Indeed, we can take $I_{e,\Pi}$ which satisfies Q, and choose the smallest $n_0 > 1$ such that $I^{n_0}_{e,\Pi}$ violates Q. Hence, $I^{n_0-1}_{e,\Pi}$ satisfies Q, but $I^{3(n_0-1)-1}_{e,\Pi}$ does not. Then, by Proposition 4.8, good possible worlds of H give a possible world of $\mathcal{I}$ that satisfies Q, and bad possible worlds of H give a possible world of $\mathcal{I}$ that does not satisfy Q. This argument concludes the proof of Theorem 4.6.

5 Finding a Minimal Tight Pattern

In the previous section, we have shown hardness for queries (bounded or unbounded) that have a model with a non-iterable edge; leaving the case of unbounded queries open, for which, in all models, all non-leaf edges can be iterated. We first note that such queries indeed exist:

▶ **Example 5.1.** Consider the following Datalog program:

$$R(x,y) \to A(y), \quad A(x), S(x,y) \to B(y), \quad B(x), S(y,x) \to A(y), \quad T(x,y), B(x) \to \mathrm{Goal}().$$

This program is unbounded, as it tests if the instance contains a path of the form $R(a, a_1)$, $S(a_1, a_2), S^-(a_2, a_3), \ldots, S(a_{2n+1}, a_{2n+2}), T(a_{2n+2}, b)$. However, it has no model with a non-iterable edge: in every model, the query is satisfied by a path of the form above, and we cannot break such a path by iterating an edge (i.e., this yields a longer path of the same form).

If we tried to reduce from #PP2DNF for this query as in the previous section, then the reduction would fail because the edge is iterable: in possible worlds of the bipartite graph, where we have not retained two adjacent vertices, we would still have matches of the query in the corresponding possible world of the probabilistic instance, where we go from a chosen vertex to another by going *back-and-forth* on the copies of e that code the edges of the bipartite graph. These are the "back-and-forth matches" which were missed in [26, 27].

In light of this, we handle the case of such queries in the next two sections. In this section, we prove a general result for unbounded queries (independent from the previous section): all unbounded queries must have a model with a *tight edge*, which is additionally *minimal* in some sense. Tight edges and iterable edges will then be used in Section 6 to show hardness for unbounded queries which are not covered by the previous section.

Let us start by defining this notion of *tight edge*, via a rewriting operation on instances called a *dissociation*.

▶ **Definition 5.2.** *The* dissociation *of a non-leaf edge $e = (u, v)$ in I is the instance I' where:*

- $\mathrm{dom}(I') = \mathrm{dom}(I) \cup \{u', v'\}$ *where u' and v' are fresh.*
- *I' is I where we create a copy of the edge e on (u, v') and on (u', v), and then remove all non-unary facts that realize e in I'.*

Dissociation is illustrated in the following example (see also Figure 4).

▶ **Example 5.3.** Consider the instance $I = \{R(a, b), S(b, a), T(b, a), R(a, c), S(c, b), S(d, b), U(a, a), U(b, b)\}$. The edge (a, b) is non-leaf, as witnessed by the edges $\{a, c\}$ and $\{b, c\}$. The result of the dissociation is $I' = \{R(a, b'), S(b', a), T(b', a), R(a', b), S(b, a'), T(b, a'), R(a, c), S(c, b), S(d, b), U(a, a), U(a', a'), U(b, b), U(b', b')\}$.

We then call an edge *tight* in a model of Q if dissociating it makes Q false.

▶ **Definition 5.4.** *Let Q be a query and I be a model of Q. An edge e of I is* tight *if it is non-leaf, and the result of the dissociation of e in I does not satisfy Q. A* tight pattern *for the query Q is a pair (I, e) of a model I of Q and of an edge e of I that is tight.*

Intuitively, a tight pattern is a model of a query containing at least three edges $\{u, a\}, \{a, b\}, \{b, v\}$ (possibly $u = v$) such that performing a dissociation makes the query false. For instance, for the unsafe CQ $Q_0 : R(w, x), S(x, y), T(y, z)$ from [16], a tight pattern would be $R(a, b), S(b, c), T(c, d)$ with the edge (b, c). Again, not all unsafe CQs have a tight pattern, e.g., Q_0' and Q_1 from Section 4 do not.

For our purposes, we will not only need tight patterns, but *minimal tight patterns*:

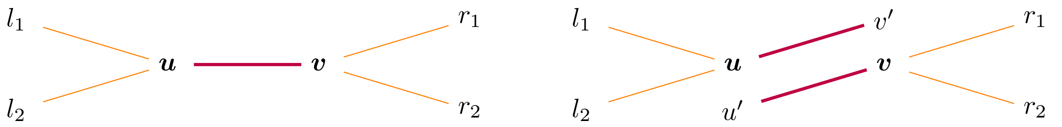


Figure 4 An instance (left) with a non-leaf edge (u, v), and the result (right) of dissociating (u, v).

▶ **Definition 5.5.** *Given an instance I with a non-leaf edge $e = (a, b)$, the* weight *of e is the number of facts that realize e in I (including unary facts). The* side weight *of e is the number of $\sigma^{\leftrightarrow}$-facts in I that are left-incident to e, plus the number of $\sigma^{\leftrightarrow}$-facts in I that are right-incident to e. Given a query Q, we say that a tight pattern (I, e) is* minimal *if:*

- *Q has no tight pattern (I', e') where the weight of e' is strictly less than that of e; and*
- *Q has no tight pattern (I', e') where the weight of e' is equal to that of e and the side weight of e' is strictly less than that of e.*

We can now state the main result of this section:

▶ **Theorem 5.6.** *Every unbounded query Q has a minimal tight pattern.*

The idea of how to find tight patterns is as follows. We first note that the only instances without non-leaf edges are intuitively disjoint unions of star-shaped subinstances. Now, if a query is unbounded, then its validity cannot be determined simply by looking at such subinstances (unlike Q_0' or Q_1 above), so there must be a model of the query with an edge that we cannot dissociate without breaking the query, i.e., a tight pattern. Once we know that there is a tight pattern, then it is simple to argue that we can find a model with a tight edge that is minimal in the sense that we require.

To formalize this intuition, let us first note that any *iterative dissociation process*, i.e., any process of iteratively applying dissociation to a given instance, will necessarily terminate. More precisely, an *iterative dissociation process* is a sequence of instances starting at an instance I and where each instance is defined from the previous one by performing the dissociation of some non-leaf edge. We say that the process *terminates* if it reaches an instance, where there is no edge left to dissociate, i.e., all edges are leaf edges.

▶ **Observation 5.7.** *For any instance I, the iterative dissociation process will terminate in n steps, where n is the number of non-leaf edges in I.*

Proof sketch. Each dissociation decreases the number of non-leaf edges by 1. ◀

Let us now consider instances with no non-leaf edges. They are intuitively disjoint unions of star-shaped subinstances, and in particular they homomorphically map to some constant-sized subset of their facts, as will be crucial when studying our unbounded query.

▶ **Proposition 5.8.** *For every signature σ, there exists a bound $k_\sigma > 0$, ensuring the following: For every instance I on σ having no non-leaf edge, there exists an instance $I' \subseteq I$ such that I has a homomorphism to I' and such that we have $|I'| < k_\sigma$.*

Proof sketch. Connected instances where we cannot perform a dissociation can have at most one non-leaf element, with all edges using this element and a leaf. Now, each edge can be described by the set of facts that realize it, for which there are finitely many possibilities (exponentially many in the signature size). We can thus show the result by collapsing together edges having the same set of facts.

Disconnected instances where we cannot perform a dissociation are unions of the connected instances of the form above, so the number of possibilities up to homomorphic equivalence is finite (exponential in the number of possible connected instances). We can then conclude by collapsing together connected components that are isomorphic. ◀

We can now prove Theorem 5.6 by appealing to the unboundedness of the query. To do this, we will rephrase unboundedness in terms of *minimal models*:

▶ **Definition 5.9.** *A* minimal model *of a query Q is an instance I that satisfies Q and such that every proper subinstance of I violates Q.*

We can rephrase the unboundedness of a UCQ^∞ Q in terms of minimal models: Q is unbounded iff it has infinitely many minimal models. Indeed, if a query Q has finitely many minimal models, then it is clearly equivalent to the UCQ formed from these minimal models, because it is closed under homomorphisms. Conversely, if Q is equivalent to a UCQ, then it has finitely many minimal models which are obtained as homomorphic images of the UCQ disjuncts. Thus, we can clearly rephrase unboundedness as follows:

▶ **Observation 5.10.** *A UCQ^∞ query Q is unbounded iff it has a minimal model I with $> k$ facts for any $k \in \mathbb{N}$.*

We can now show Theorem 5.6. We first show how to find a tight pattern, which is not necessarily minimal. To do this, take a sufficiently large minimal model I_0 of the query by Observation 5.10, and perform an iterative dissociation process, while it is possible to dissociate edges without breaking the query. By Observation 5.7, this process eventually terminates. If the result I_n of the process has a non-leaf edge which we did not dissociate, then dissociating this edge breaks the query, so it is tight and we are done. Otherwise, we reach a contradiction: as there are only leaf edges in I_n, Proposition 5.8 implies that I_n has a homomorphism to a constant-sized subset I'_n, which also satisfies Q. Now, I'_n has a homomorphism back into I_n (as a subset), then into I_0 (by undoing the dissociations). This identifies a constant-sized subset of I_0 that satisfies the query, which contradicts the definition of I_0 as a large minimal model.

Having found a tight pattern, we find a minimal tight pattern simply by minimizing first on the weight, then on the side weight, which concludes the proof of Theorem 5.6.

6 Hardness with Tight Iterable Edges

In this section, we conclude the proof of Theorem 3.3 by showing that a minimal tight pattern can be used to show hardness when it is iterable. Formally:

▶ **Theorem 6.1.** *For every query Q, if we have a minimal tight pattern (I, e) where the edge e is iterable, then $\mathrm{PQE}(Q)$ is #P-hard.*

This covers all the queries to which Section 4 did not apply, and concludes the proof of Theorem 3.3:

Proof of Theorem 3.3. Let Q be an unbounded UCQ^∞. If we have a model of Q with a non-iterable edge, then we conclude by Theorem 4.6 that $\mathrm{PQE}(Q)$ is #P-hard. Otherwise, by Theorem 5.6, we have a minimal tight pattern, and its edge is then iterable (otherwise the first case would have applied), so that we can apply Theorem 6.1. ◀

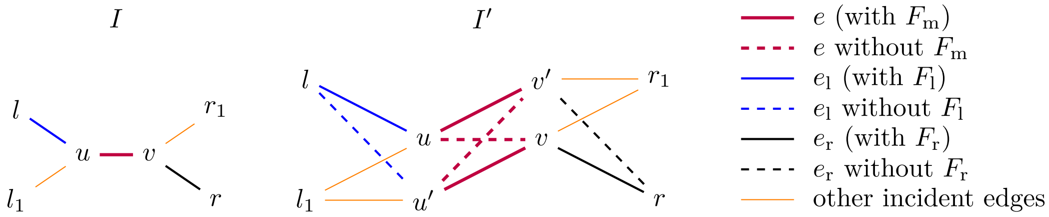

I
I'
l
r_1
u
v
l_1
r
l
v'
r_1
u
v
l_1
u'
r
e (with F_m)
e without F_m
e_l (with F_l)
e_l without F_l
e_r (with F_r)
e_r without F_r
other incident edges

Figure 5 Example of fine dissociation from an instance I (left) to I' (middle) for a choice of e, of $\Pi = (F_\mathrm{l}, F_\mathrm{r})$, and of F_m. We call e_l and e_r the edges of F_l and F_r. A key is given at the right.

Thus, it only remains to show Theorem 6.1. The idea is to use the iterable edge e of the minimal tight pattern (I, e) for some incident pair Π to reduce from the undirected st-connectivity problem #U-ST-CON (Definition 2.2). Given an input st-graph G for #U-ST-CON, we will code it as a TID $\mathcal{I}$ built using $I_{e,\Pi}$, with one probabilistic fact per edge of G. To show a reduction, we will argue that good possible worlds of G correspond to possible worlds J' of $\mathcal{I}$ containing some iterate of the instance $I^n_{e,\Pi}$ (with n being the length of the path), and J' then satisfies Q because e is iterable. Conversely, we will argue that bad possible worlds of G correspond to possible worlds J' of $\mathcal{I}$ that have a homomorphism to a so-called *fine dissociation* of e in I, and we will argue that this violates Q query thanks to our choice of (I, e) as a minimal tight pattern. Let us first define this notion of *fine dissociation*:

▶ **Definition 6.2.** *Let I be an instance, let $e = (u, v)$ be a non-leaf edge in I, let $F_\mathrm{l} = R_\mathrm{l}(l, u)$ and $F_\mathrm{r} = R_\mathrm{r}(v, r)$ be an incident pair of e in I, and let F_m be a non-unary fact realizing the edge e. The result of performing the* fine dissociation *of e in I relative to $F_\mathrm{l}, F_\mathrm{r}$ and F_m is an instance I' on the domain* $\mathrm{dom}(I') = \mathrm{dom}(I) \cup \{u', v'\}$*, where the new elements are fresh. It is obtained by applying the following steps:*

- Copy non-incident facts: *Initialize I' as the induced subinstance of I on* $\mathrm{dom}(I) \setminus \{u, v\}$*.*
- Copy incident facts F_l and F_r: *Add the facts F_l and F_r to I'.*
- Copy other left-incident facts: *For every $\sigma^{\leftrightarrow}$-fact $F'_\mathrm{l} = R'_\mathrm{l}(l', u)$ of I that is left-incident to e (i.e., $l' \notin \{u, v\}$) and where $F'_\mathrm{l} \neq F_\mathrm{l}$, add to I' the fact $R'_\mathrm{l}(l', u')$.*
- Copy other right-incident facts: *For every $\sigma^{\leftrightarrow}$-fact $F'_\mathrm{r} = R'_\mathrm{r}(v, r')$ of I that is right-incident to e (i.e., $r' \notin \{u, v\}$) and where $F'_\mathrm{r} \neq F_\mathrm{r}$, add to I' the fact $R'_\mathrm{r}(v', r')$.*
- Create the copies of e: *Copy e on the pairs (u, v') and (u', v) of I', and copy e* except the fact *F_m on the pairs (u, v) and (u', v') of I'.*

The result of a *fine dissociation* is illustrated in Figure 5. If the only non-unary fact realizing the edge e in I is F_m, then (u, v) and (u', v') are not edges in the result of the fine dissociation; otherwise, they are edges but with a smaller weight than e. Observe that fine dissociation is related both to dissociation (Section 5) and to iteration (Section 4). We will study later when fine dissociation can make the query false.

We can now start the proof of Theorem 6.1 by describing the coding, which depends on our choice of $I_{e,\Pi}$ and of a fact F_m, but does not depend on the query Q. Given an input st-graph, i.e., an undirected graph G with source s and target t, we construct a TID $\mathcal{I}$ whose possible worlds will have a bijection to those of G.

▶ **Definition 6.3.** *Let $I_{e,\Pi}$ be an instance where $e = (u, v)$, $\Pi = (F_\mathrm{l}, F_\mathrm{r})$, $F_\mathrm{l} = R_\mathrm{l}(l, u)$, $F_\mathrm{r} = R_\mathrm{r}(v, r)$ and let F_m be a non-unary fact of I realizing e. Let $G = (W, C, s, t)$ be an undirected graph with source and target. The* coding *of G relative to $I_{e,\Pi}$ and F_m is a TID $\mathcal{I} = (J, \pi)$ with domain* $\mathrm{dom}(J) := \mathrm{dom}(I) \cup \{u_c \mid c \in C\} \cup \{v_w \mid w \in W \setminus \{t\}\}$*, where the new elements are fresh, and where we use v_t to refer to v for convenience. The facts of J and the probability mapping π are defined as follows:*

- Copy non-incident facts: *Initialize J as the induced subinstance of I on* $\mathrm{dom}(I) \setminus \{u, v\}$.
- Copy incident facts F_l and F_r: *Add the facts F_l and F_r to J.*
- Copy other left-incident facts: *For every $\sigma^{\leftrightarrow}$-fact $F'_\mathrm{l} = R'_\mathrm{l}(l', u)$ of I that is left-incident to e (i.e., $l' \notin \{u, v\}$) and where $F'_\mathrm{l} \neq F_\mathrm{l}$, add to J the facts $R'_\mathrm{l}(l', u_c)$ for each edge $c \in C$.*
- Copy other right-incident facts: *For every $\sigma^{\leftrightarrow}$-fact $F'_\mathrm{r} = R'_\mathrm{r}(v, r')$ of I that is right-incident to e (i.e., $r' \notin \{u, v\}$) and where $F'_\mathrm{r} \neq F_\mathrm{r}$, add to J the facts $R'_\mathrm{r}(v_w, r')$ for each $w \in W$.*
- Create copies of e: *Copy e on the pair (u, v_s) of J, and for each edge $c = \{a, b\}$ in C, copy e on the pairs (u_c, v_a) and (u_c, v_b) of J.*

Finally, we define the function π as follows. For each edge c of C, π maps the copy of the fact F_m in the edge (u_c, v_w) to 0.5, for an arbitrary choice of $w \in c$. All other facts are mapped to 1 by π.

The coding is exemplified in Figure 6. It is important to note that the edges are coded by paths of length 2. This choice is critical, because the source graph to the reduction is undirected, but the facts on edges are directed; so, intuitively, we symmetrize by having two copies of the edge in opposite directions in order to traverse them in both ways. The choice on how to orient the edges (i.e., the choice of $w \in c$ when defining π) has no impact in how the edges can be traversed when their probabilistic fact is present, but it has an impact when the probabilistic fact is missing. Indeed, this is the reason why fine dissociation includes two copies of e with one missing fact.

It is easy to see that the given coding is in polynomial time in the input G for every choice of $I_{e,\Pi}$ and F_m. Let us now define the bijection ϕ, mapping each possible world ω of G to a possible world of the TID $\mathcal{I}$ as follows. For each edge $c \in C$, we keep the probabilistic fact incident to u_c in the instance $\phi(\omega)$ if c is kept in the possible world ω, and we do not keep it, otherwise. It is obvious that this correspondence is bijective and that all possible worlds have the same probability $0.5^{|C|}$. We can now explain why ϕ defines a reduction:

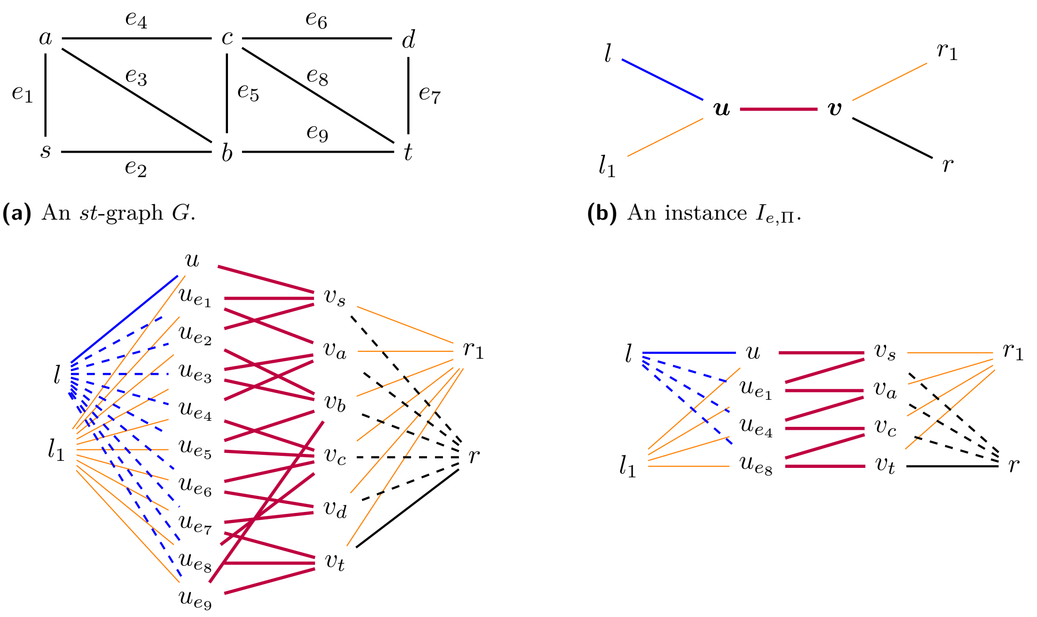
▶ **Proposition 6.4.** *Let the TID $\mathcal{I} = (J, \pi)$ be the coding of an undirected st-graph G relative to an instance $I_{e,\Pi}$ and to F_m as described in Definition 6.3. Let ϕ be the bijective function defined above from the possible worlds of G to those of $\mathcal{I}$. Then:*

1. *For any* good *possible world ω of G with a witnessing simple s, t-path traversing n edges, $\phi(\omega)$ has a homomorphism* from $I_{e,\Pi}^{n+1}$.
2. *For any* bad *possible world ω of G, $\phi(\omega)$ has a homomorphism* to *the result of finely dissociating e in I relative to* Π *and F_m.*

Proof sketch. For the forward direction, we find $I_{e,\Pi}^{n+1}$ as a subinstance of $\phi(\omega)$ by following the image in J of the witnessing path in ω.

The backward direction is again more challenging. We consider a *cut* in ω between s and t. Then, any two vertices on different sides of the cut can only be connected by two successive copies of e with one of them missing the fact F_m (it can be the first or second copy depending on the orientation choice). We then construct the homomorphism to the fine dissociation (Figure 5) by mapping the vertex u to u, mapping vertices on the s-side of the cut (including v_s) to v', mapping the edges between these vertices back-and-forth to $(v'u)$ and (u, v'), mapping all vertices on the t-side (including $v_t = v$) to v, mapping edges between them back-and-forth to (v, u') and (u', v), and mapping the edges across the cut to either (v', u') and (u', v) or to (v', u) and (u, v), depending on the orientation choice. ◀

Proposition 6.4 leads us to a proof of Theorem 6.1: good possible worlds of G give a possible world of $\mathcal{I}$ that satisfies Q thanks to the iterability of e, and bad possible worlds of G give a possible world of $\mathcal{I}$ having a homomorphism to the fine dissociation. The only



(a) An st-graph G.

(b) An instance $I_{e,\Pi}$.

(c) Coding of the graph G relative to $I_{e,\Pi}$ and some F_{m}.

(d) The image of an s-t path in the coding.

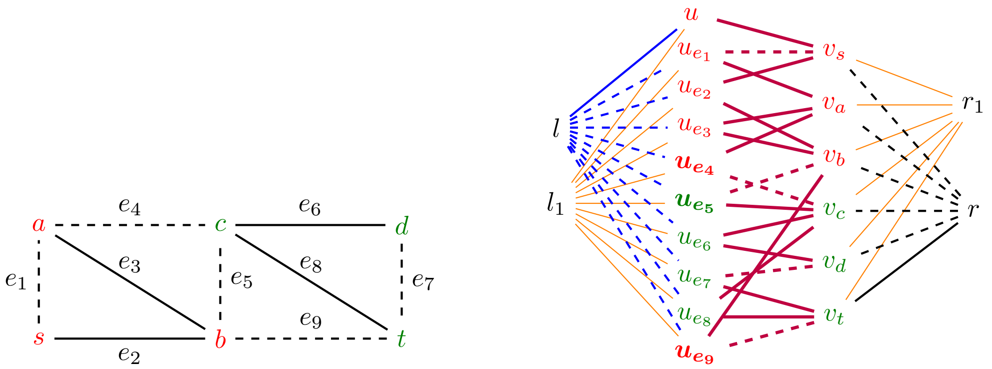
Figure 6 Example of the coding on an st-graph G shown in Figure 6a. We encode G relative to an instance $I_{e,\Pi}$ (Figure 6b) and to some choice of a non-unary fact F_{m} realizing e. The coding of G relative to $I_{e,\Pi}$ and F_{m} is shown in Figure 6c, with the probabilistic facts being *exactly one* copy of F_{m} for *one* of every pair of purple edges adjacent to an element in $\{u_{e_1}, \ldots, u_{e_9}\}$. Each st-path in G gives rise to a subinstance in the coding: consider for instance the st-path which is via the edges e_1, e_4, e_8. The corresponding subinstance in the coding for this path is shown in Figure 6d, which is an iterate of the form $I_{e,\Pi}^{n+1}$, where n is the number of edges on the path, and hence $n = 3$, in this case.

missing piece is to argue that the fine dissociation does not satisfy the query. We can do this using the minimality and tightness of the pattern:

▶ **Lemma 6.5.** *Let Q be a query, let (I, e) be a minimal tight pattern for Q, let Π be an arbitrary incident pair of e in I, and let F_{m} be an arbitrary non-unary fact realizing e in I. Then, the result of the fine dissociation of e in I relative to Π and F_{m} does not satisfy Q.*

Proof sketch. We assume that the fine dissociation I_1 satisfies Q, and show a contradiction by rewriting it in several steps. The process of the proof is illustrated as Figure 8. We first dissociate the copies of e in I_1 with F_{m} missing: as their weight is strictly less than e, the minimality of e ensures that they are not tight, so the result I_2 still satisfies Q. We then homomorphically fold the dissociated edges into the copies of e, and obtain I_3, which still satisfies Q: it is like I_1 but without the copies of e with F_{m} missing. Now, the copies of e in I_3 have a smaller side weight than in I_1, so the minimality of e ensures that they are not tight. We can dissociate them again, yielding I_4, which still satisfies Q. We can now homomorphically fold the dissociated edges and obtain I_5, which still satisfies Q, and is homomorphic to the dissociation of e in I_1. As e was tight, I_5 should not satisfy the query, so we have reached a contradiction. ◀

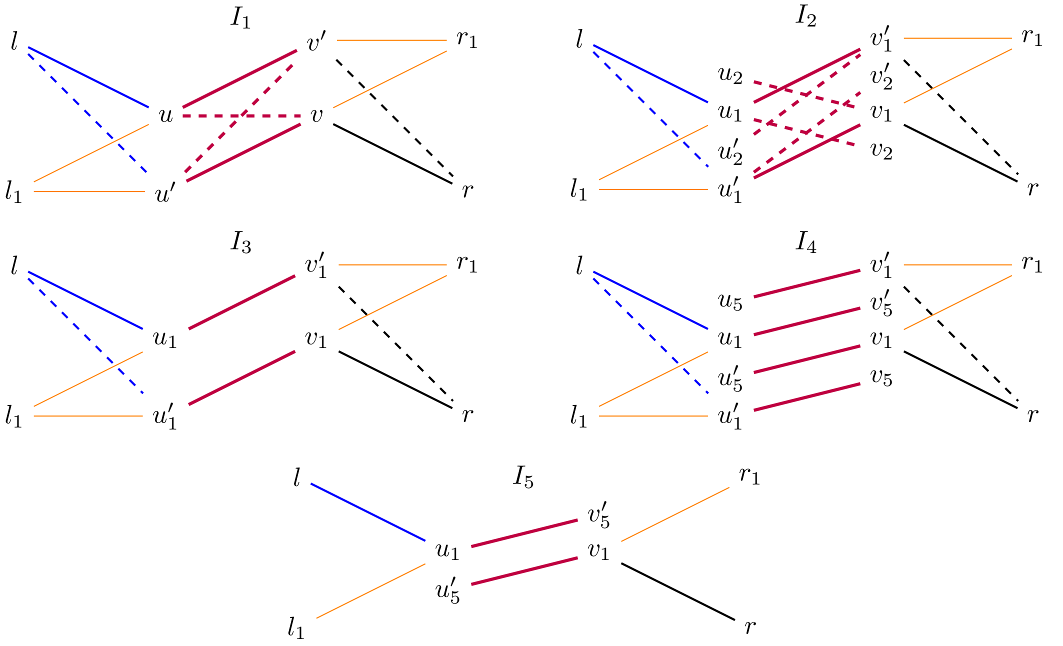
This concludes the proof of Theorem 6.1, and thus of our main theorem (Theorem 3.3).



(a) A possible world ω of G with no s,t-path (dashed edges are the ones that are not kept): the vertices are colored in red or green depending on their side of the cut.

(b) Possible world of the coding in Figure 6c for the possible world of G at the left. Copies of e are dashed when they are missing the fact F_{m}. Vertices u_{e_i} corresponding to edges across the cut are in bold.

■ **Figure 7** Illustration of a possible world (Figure 7a) of the graph G from Figure 6a, and the corresponding possible world (Figure 7b) of the coding (Figure 6c). The homomorphism of Figure 7b to the fine dissociation is given by the vertex colors: the red u-vertices are mapped to u, the red v-vertices are mapped to v', the green u-vertices are mapped to u', and the green v-vertices are mapped to v. The vertex colors are determined by the cut (Figure 7a) except for the bold vertices where it depends on the orientation choice.



■ **Figure 8** Illustration of the proof of Lemma 6.5, with I_1 being the fine dissociation I' of Figure 5, and I_5 being isomorphic to the dissociation on Figure 4.

7 Conclusion

We have shown that PQE is #P-hard for any unbounded UCQ^∞ on an arity-two signature, and hence proved a dichotomy on PQE for all UCQ^∞ queries: either they are unbounded and PQE is #P-hard, or they are bounded and the dichotomy by Dalvi and Suciu applies. Our result captures many query languages; in particular disjunctive Datalog over binary signatures, regular path queries, and all ontology-mediated queries closed under homomorphisms.

There are three natural directions to extend our result. First, we could study queries that are *not* homomorphism-closed, e.g., with disequalities or negation. We believe that this would require different techniques as the problem is still open for UCQs (beyond the results of [21]). Second, we could lift the arity restriction and work on signatures of arbitrary arity: we conjecture that PQE is still #P-hard for any unbounded UCQ^∞ in that case. Much of our proof techniques may adapt, but we do not know how to extend the definitions of dissociation, fine dissociation, and iteration. In particular, dissociation on a fact is difficult to adapt because incident facts on arbitrary arity signatures may intersect in complicated ways. For this reason, we leave the extension to arbitrary-arity signatures to future work. Third, a natural question for future work is whether our hardness result on unbounded homomorphism-closed queries also applies to the *(unweighted) model counting problem*, where all facts of the TID must have probability 0.5: the hardness of this problem has only been shown on the class of self-join free CQs [4].

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.

2 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-16)*, pages 355–370. ACM, 2016.

3 Antoine Amarilli and İsmail İlkan Ceylan. A dichotomy for homomorphism-closed queries on probabilistic graphs, 2020. Full version with proofs: `https://arxiv.org/abs/1910.02048`.

4 Antoine Amarilli and Benny Kimelfeld. Model counting for conjunctive queries without self-joins. *CoRR*, abs/1908.07093, 2019.

5 Franz Baader, Diego Calvanese, Deborah L McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic handbook*. Cambridge University Press, 2007.

6 Pablo Barceló, Diego Figueira, and Miguel Romero. Boundedness of conjunctive regular path queries, 2019. URL: `https://hal.archives-ouvertes.fr/hal-02056388/`.

7 Michael Benedikt, Balder Ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 293–304. IEEE, 2015.

8 Meghyn Bienvenu, Balder Ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Transactions on Database Systems (TODS)*, 39(4):33:1–33:44, 2014.

9 Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Ontology-mediated queries for probabilistic databases. In *Proceedings of the 31th AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 1063–1069. AAAI Press, 2017.

10 Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Ontology-mediated query answering over log-linear probabilistic data. In *Proceedings of the 33rd National Conference on Artificial Intelligence (AAAI-19)*. AAAI Press, 2019.

11 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *JAIR*, 48:115–174, 2013.

12 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

13 İsmail İlkan Ceylan. *Query answering in probabilistic data and knowledge bases.* Doctoral thesis, TU Dresden, 2017.

14 İsmail İlkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-world probabilistic databases. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR-16)*, pages 339–348. AAAI Press, 2016.

15 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC-71)*, pages 151–158. ACM, 1971.

16 Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007.

17 Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012.

18 Anuj Dawar and Stephan Kreutzer. On Datalog vs. LFP. In *International Colloquium on Automata, Languages, and Programming*, pages 160–171. Springer, 2008.

19 Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge Vault: A Web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–610. ACM, 2014.

20 Thomas Eiter, Magdalena Ortiz, Mantas Šimkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-$\mathcal{SHIQ}$ plus rules. In *AAAI*, 2012.

21 Robert Fink and Dan Olteanu. Dichotomies for queries with negation in probabilistic databases. *ACM Transactions on Database Systems (TODS)*, 41(1):4:1–4:47, 2016.

22 Haim Gaifman, Harry Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *J. ACM*, 40(3):683–713, July 1993.

23 Georg Gottlob and Thomas Schwentick. Rewriting ontological queries into small nonrecursive Datalog programs. In *KR*, 2012.

24 Gerd G Hillebrand, Paris C Kanellakis, Harry G Mairson, and Moshe Y Vardi. Undecidable boundedness problems for Datalog programs. *The Journal of Logic Programming*, 25(2):163–190, 1995.

25 Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

26 Jean Christoph Jung. *Reasoning in many dimensions: uncertainty and products of modal logics.* PhD thesis, University of Bremen, 2014.

27 Jean Christoph Jung and Carsten Lutz. Ontology-based access to probabilistic data with OWL QL. In *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, pages 182–197. Springer-Verlag, 2012.

28 T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 2302–2310, 2015.

29 Dan Olteanu and Jiewen Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *Proceedings of the 2nd International Conference on Scalable Uncertainty Management (SUM-08)*, volume 5291 of *LNCS*, pages 326–340, 2008.

30 Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pages 389–402. ACM, 2009.

31 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4), 1983 .

32 Christopher Ré and Dan Suciu. The trichotomy of HAVING queries on a probabilistic database. *The VLDB Journal*, 18(5):1091–1116, 2009.

33 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic databases*, volume 3. Morgan-Claypool, 2011.

34 Leslie Gabriel Valiant. The complexity of computing the permanent. *TCS*, 8(2):189–201, 1979.

On the Expressiveness of LARA: A Unified Language for Linear and Relational Algebra

Pablo Barceló
IMC, Pontificia Universidad Católica de Chile, Santiago, Chile
IMFD, Santiago, Chile
pbarcelo@ing.puc.cl

Nelson Higuera
Department of Computer Science, University of Chile, Santiago, Chile
IMFD, Santiago, Chile
nhiguera@dcc.uchile.cl

Jorge Pérez
Department of Computer Science, University of Chile, Santiago, Chile
IMFD, Santiago, Chile
jperez@dcc.uchile.cl

Bernardo Subercaseaux
Department of Computer Science, University of Chile, Santiago, Chile
IMFD, Santiago, Chile
bsubercaseaux@dcc.uchile.cl

Abstract

We study the expressive power of the Lara language – a recently proposed unified model for expressing relational and linear algebra operations – both in terms of traditional database query languages and some analytic tasks often performed in machine learning pipelines. We start by showing Lara to be expressive complete with respect to first-order logic with aggregation. Since Lara is parameterized by a set of user-defined functions which allow to transform values in tables, the exact expressive power of the language depends on how these functions are defined. We distinguish two main cases depending on the level of genericity queries are enforced to satisfy. Under strong genericity assumptions the language cannot express matrix convolution, a very important operation in current machine learning operations. This language is also local, and thus cannot express operations such as matrix inverse that exhibit a recursive behavior. For expressing convolution, one can relax the genericity requirement by adding an underlying linear order on the domain. This, however, destroys locality and turns the expressive power of the language much more difficult to understand. In particular, although under complexity assumptions the resulting language can still not express matrix inverse, a proof of this fact without such assumptions seems challenging to obtain.

2012 ACM Subject Classification Information systems → Query languages

Keywords and phrases languages for linear and relational algebra, expressive power, first order logic with aggregation, matrix convolution, matrix inverse, query genericity, locality of queries, safety

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.6

Related Version `https://arxiv.org/abs/1909.11693v1`

Funding This work is funded by the Millennium Institute for Foundational Research on Data (IMFD Chile). Barceló and Pérez are also funded by Fondecyt grant 1200967.

1 Introduction

Many of the actual analytics systems require both relational algebra and statistical functionalities for manipulating the data. In fact, while tools based on relational algebra are often used for preparing and structuring the data, the ones based on statistics and machine learning (ML) are applied to quantitatively reason about such data. Based on the "impedance

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 6; pp. 6:1–6:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

mismatch" that this dichotomy creates [12], the database theory community has highlighted the need of developing a standard data model and query language for such applications, meaning an extension of relational algebra with linear algebra operators that is able to express the most common ML tasks [4]. Noticeably, the ML community has also recently manifested the need for what – at least from a database perspective – can be seen as a high-level language that manipulates tensors. Indeed, despite their wide adoption, there has been a recent interest in redesigning the way in which tensors are used in deep learning code [8, 15, 16], due to some pitfalls of the current way in which tensors are abstracted.

Hutchinson et al. [10, 9] have recently proposed a data model and a query language that aims at becoming the "universal connector" that solves the aforementioned impedance. On the one hand, the data model proposed corresponds to the so-called *associative tables*, which generalize relational tables, tensors, arrays, and others. Associative tables are two-sorted, consisting of *keys* and *values* that such keys map to. The query language, on the other hand, is called LARA, and subsumes several known languages for the data models mentioned above. LARA is an algebraic language designed in a minimalistic way by only including three operators; namely, *join*, *union*, and *extension*. In rough terms, the first one corresponds to the traditional join from relational algebra, the second one to the operation of aggregation, and the third one to the extension defined by a function as in a *flatmap* operation. It has been shown that LARA subsumes all relational algebra operations and is capable of expressing several interesting linear algebra operations used in graph algorithms [9].

Based on the proposal of LARA as a unified language for relational and linear algebra, it is relevant to develop a deeper understanding of its expressive power, both in terms of the logical query languages traditionally studied in database theory and ML operations often performed in practical applications. We start with the former and show that LARA is expressive complete with respect to *first-order logic with aggregation* (FO_{Agg}), a language that has been studied as a way to abstract the expressive power of SQL without recursion; cf., [13, 14]. (To be more precise, LARA is expressive complete with respect to a suitable syntactic fragment of FO_{Agg} that ensures that formulas are *safe* and get properly evaluated over associative tables). This result can be seen as a sanity check for LARA. In fact, this language is specifically tailored to handle aggregation in conjunction with relational algebra operations, and a classical result in database theory establishes that the latter is expressive complete with respect to first-order logic (FO). We observe that while LARA consists of *positive* algebraic operators only, set difference can be encoded in the language by a combination of aggregate operators and extension functions. Our expressive completeness result is parameterized by the class of functions allowed in the extension operator. For each such a class Ω we allow FO_{Agg} to contain all built-in predicates that encode the functions in Ω.

To understand which ML operators LARA can express, one then needs to bound the class Ω of extension functions allowed in the language. We start with a tame class that can still express several relevant functions. These are the FO-expressible functions that allow to compute arbitrary numerical predicates on values, but can only compare keys with respect to (in)equality. This restriction makes the logic quite amenable for theoretical exploration. In fact, it is easy to show that the resulting "tame version" of LARA satisfies a strong *genericity* criterion (in terms of key-permutations) and is also *local*, in the sense that queries in the language can only see up to a fixed-radius neighborhood from its free variables; cf., [14]. The first property implies that this tame version of LARA cannot express non-generic operations, such as matrix *convolution*, and the second one that it cannot express inherently recursive queries, such as matrix *inverse*. Both operations are very relevant for ML applications; e.g., matrix convolution is routinely applied in dimension-reduction tasks, while matrix inverse is used for learning the matrix of coefficient values in linear regression.

We then look more carefully at the case of matrix convolution, and show that this query can be expressed if we relax the genericity properties of the language by assuming the presence of a linear order on the domain of keys. (This relaxation implies that queries expressible in the resulting version of LARA are no longer invariant with respect to key-permutations). This language, however, is much harder to understand in terms of its expressive power. In particular, it can express non-local queries, and hence we cannot apply locality techniques to show that the matrix inversion query is not expressible in it. To prove this result, then, one would have to apply techniques based on the *Ehrenfeucht-Fraïssé* games that characterize the expressive power of the logic. Showing results based on such games in the presence of a linear order, however, is often combinatorially difficult, and currently we do not know whether this is possible. In turn, it is possible to obtain that matrix inversion is not expressible in a natural restriction of our language under complexity-theoretic assumptions. This is because the data complexity of queries expressible in such a restricted language is LOGSPACE, while matrix inversion is complete for a class that is believed to be a proper extension of the latter.

The main objective of our paper is connecting the study of the expressive power of tensor-based query languages, in general, and of LARA, in particular, with traditional database theory concepts and the arsenal of techniques that have been developed in this area to study the expressiveness of query languages. We also aim at identifying potential lines for future research that appear in connection with this problem. Our work is close in spirit to the recent study of MATLANG [1, 6], a matrix-manipulation language based on elementary linear algebra operations. It is shown that this language is contained in the three-variable fragment of relational algebra with summation and, thus, it is local. This implies that the core of MATLANG cannot check for the presence of a four-clique in a graph (represented as a Boolean matrix), as this query requires at least four variables to be expressed, and neither can it express the non-local matrix inversion query. It can be shown that MATLANG is strictly contained in the tame version of LARA that is mentioned above, and thus some of our results can be seen as generalizations of the ones for MATLANG.

Organization of the paper. Basics of LARA and $\mathrm{FO}_{\mathsf{Agg}}$ are presented in Sections 2 and 3, respectively. The expressive completeness of LARA in terms of $\mathrm{FO}_{\mathsf{Agg}}$ is shown in Section 4. The tame version of LARA and some inexpressibility results relating to it are given in Section 5, while in Section 6 we present a version of LARA that can express convolution and some discussion about its expressive power. We finalize in Section 7 with concluding remarks and future work. Due to space constraints some of our proofs are in the appendix, or simply reserved to a final version of the paper.

2 The LARA Language

For integers $m \leq n$, we write $[m, n]$ for $\{m, \ldots, n\}$ and $[n]$ for $\{1, \ldots, n\}$. If $\bar{v} = (v_1, \ldots, v_n)$ is a tuple of elements, we write $\bar{v}[i]$ for v_i. We denote multisets as $\{\!\{a, b, \ldots\}\!\}$.

Data model

A *relational schema* is a finite collection σ of *two-sorted* relation symbols. The first sort consists of *key-attributes* and the second one of *value-attributes*. Each relation symbol $R \in \sigma$ is then associated with a pair $(\bar{K}, \bar{V})$, where $\bar{K}$ and $\bar{V}$ are (possibly empty) tuples of different key- and value- attributes, respectively. We write $R[\bar{K}, \bar{V}]$ to denote that $(\bar{K}, \bar{V})$ is the *sort* of R. We do not distinguish between $\bar{K}$, resp., $\bar{V}$, and the set of attributes mentioned in it.

There are two countably infinite sets of objects over which databases are populated: A domain of *keys*, which interpret key-attributes and is denoted Keys, and a domain of *values*, which interpret value-attributes and is denoted Values. A *tuple of sort* $(\bar{K}, \bar{V})$ is a function $t : \bar{K} \cup \bar{V} \to \mathsf{Keys} \cup \mathsf{Values}$ such that $t(A) \in \mathsf{Keys}$ if $A \in \bar{K}$ and $t(A) \in \mathsf{Values}$ if $A \in \bar{V}$. A *database* D over schema σ is a mapping that assigns with each relation symbol $R[\bar{K}, \bar{V}] \in \sigma$ a finite set R^D of tuples of sort $(\bar{K}, \bar{V})$. We often see D as a set of *facts*, i.e., as the set of expressions $R(t)$ such that $t \in R^D$. For ease of presentation, we write $R(\bar{k}, \bar{v}) \in D$ if $R(t) \in D$ for some tuple t with $t(\bar{K}) = \bar{k}$ and $t(\bar{V}) = \bar{v}$ (where $\bar{k} \in \mathsf{Keys}^{|\bar{K}|}$ and $\bar{v} \in \mathsf{Values}^{|\bar{V}|}$).

For a database D to be a Lara *database* we need D to satisfy an extra restriction: Key attributes define a key constraint over the corresponding relation symbols. That is,

$$R(\bar{k}, \bar{v}), R(\bar{k}, \bar{v}') \in D \quad \Longrightarrow \quad \bar{v} = \bar{v}',$$

for each $R[\bar{K}, \bar{V}] \in \sigma$, $\bar{k} \in \mathsf{Keys}^{|\bar{K}|}$, and $\bar{v}, \bar{v}' \in \mathsf{Values}^{|\bar{V}|}$. Relations of the form R^D are called *associative tables* [10]. Yet, we abuse terminology and call associative table any set A of tuples of the same sort $(\bar{K}, \bar{V})$ such that $\bar{v} = \bar{v}'$ for each $(\bar{k}, \bar{v}), (\bar{k}, \bar{v}') \in A$. In such a case, A is of sort $(\bar{K}, \bar{V})$. Notice that for a tuple $(\bar{k}, \bar{v})$ in A, we can safely denote $\bar{v} = A(\bar{k})$.

Syntax

An *aggregate operator* over domain U is a family $\oplus = \{\oplus_0, \oplus_1, \dots, \oplus_\omega\}$ of partial functions, where each $\oplus_k$ takes a multiset of k elements from U and returns a single element in U. If u is a collection of k elements in U, we write $\oplus(u)$ for $\oplus_k(u)$. This notion generalizes most aggregate operators used in practical query languages; e.g., SUM, AVG, MIN, MAX, and COUNT. For simplicity, we also see binary operations $\otimes$ on U as aggregate operators $\oplus = \{\oplus_0, \oplus_1, \dots, \oplus_\omega\}$ such that $\oplus_2 = \otimes$ and $\oplus_i$ has an empty domain for each $i \neq 2$.

The syntax of Lara is parameterized by a set of *extension functions*. This is a collection Ω of user-defined functions f that map each tuple t of sort $(\bar{K}, \bar{V})$ to a finite associative table of sort $(\bar{K}', \bar{V}')$, for $\bar{K} \cap \bar{K}' = \emptyset$ and $\bar{V} \cap \bar{V}' = \emptyset$. We say that f is of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$. As an example, an extension function might take a tuple $t = (k, v_1, v_2)$ of sort (K, V_1, V_2), for $v_1, v_2 \in \mathbb{Q}$, and map it to a table of sort (K', V') that contains a single tuple (k, v), where v is the average between v_1 and v_2.

We inductively define the set of expressions in Lara(Ω) over schema σ as follows.

- *Empty associative table.* $\emptyset$ is an expression of sort $(\emptyset, \emptyset)$.
- *Atomic expressions.* If $R[\bar{K}, \bar{V}]$ is in σ, then R is an expression of sort $(\bar{K}, \bar{V})$.
- *Join.* If e_1 and e_2 are expressions of sort $(\bar{K}_1, \bar{V}_1)$ and $(\bar{K}_2, \bar{V}_2)$, respectively, and $\otimes$ is a binary operator over Values, then $e_1 \bowtie_\otimes e_2$ is an expression of sort $(\bar{K}_1 \cup \bar{K}_2, \bar{V}_1 \cup \bar{V}_2)$.
- *Union.* If e_1, e_2 are expressions of sort $(\bar{K}_1, \bar{V}_1)$ and $(\bar{K}_2, \bar{V}_2)$, respectively, and $\oplus$ is an aggregate operator over Values, then $e_1 \mathbin{\bowtie\!\!\!\!\!\bowtie}_\oplus e_2$ is an expression of sort $(\bar{K}_1 \cap \bar{K}_2, \bar{V}_1 \cup \bar{V}_2)$.
- *Extend.* For e an expression of sort $(\bar{K}, \bar{V})$ and f a function in Ω of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$, it is the case that $\mathsf{Ext}_f\, e$ is an expression of sort $(\bar{K} \cup \bar{K}', \bar{V}')$.

We write $e[\bar{K}, \bar{V}]$ to denote that expression e is of sort $(\bar{K}, \bar{V})$.

Semantics

We assume that for every aggregate operator $\oplus$ over domain Values there is a *neutral value* $0_\oplus \in \mathsf{Values}$. Formally, $\oplus(\mathcal{V}) = \oplus(\mathcal{V}')$, for every multiset $\mathcal{V}$ of elements in Values and every extension $\mathcal{V}'$ of $\mathcal{V}$ with an arbitrary number of occurrences of $0_\oplus$. An important notion is *padding*. Let $\bar{V}_1$ and $\bar{V}_2$ be tuples of value-attributes, and $\bar{v}$ a tuple over Values of sort $\bar{V}_1$.

i	**j**	**v**$_1$	**v**$_2$
0	0	1	5
0	1	2	6
1	0	3	7
1	1	4	8

j	**k**	**v**$_2$	**v**$_3$
0	0	1	1
0	1	1	2
1	0	1	1
1	1	2	1

Figure 1 Associative tables A and B used for defining the semantics of Lara.

Then $\text{pad}_{\oplus}^{\bar{V}_2}(\bar{v})$ is a new tuple $\bar{v}'$ over Values of sort $\bar{V}_1 \cup \bar{V}_2$ such that for each $V \in \bar{V}_1 \cup \bar{V}_2$ we have that $v'(V) = v(V)$, if $V \in \bar{V}_1$, and $v'(V) = 0_{\oplus}$, otherwise.

Consider tuples $\bar{k}_1$ and $\bar{k}_2$ over key-attributes $\bar{K}_1$ and $\bar{K}_2$, respectively. We say that $\bar{k}_1$ and $\bar{k}_2$ are *compatible*, if $\bar{k}_1(K) = \bar{k}_2(K)$ for every $K \in \bar{K}_1 \cap \bar{K}_2$. If $\bar{k}_1$ and $\bar{k}_2$ are compatible, one can define the extended tuple $\bar{k}_1 \cup \bar{k}_2$ over key-attributes $\bar{K}_1 \cup \bar{K}_2$. Also, given a tuple $\bar{k}$ of sort $\bar{K}$, and a set $\bar{K}' \subseteq \bar{K}$, the restriction $\bar{k}{\downarrow_{\bar{K}'}}$ of $\bar{k}$ to attributes $\bar{K}'$ is the only tuple of sort $\bar{K}'$ that is compatible with $\bar{k}$. Finally, given a multiset T of tuples $(\bar{k}, \bar{u})$ of the same sort $(\bar{K}, \bar{V})$ we define $\mathsf{Solve}_{\oplus}(T)$ as

$$\begin{aligned}\mathsf{Solve}_{\oplus}(T) \;:=\; \{(\bar{k}, \bar{v}) \mid \text{there exists } \bar{u} \text{ such that } (\bar{k}, \bar{u}) \in T \text{ and}\\ \bar{v}[i] = \bigoplus_{\bar{v}' \,:\, (\bar{k}, \bar{v}') \in T} \bar{v}'[i], \text{ for each } i \in [|\bar{V}|]\}.\end{aligned}$$

That is, $\mathsf{Solve}_{\oplus}(T)$ turns the multiset T into an associative table T' by first grouping together tuples that have the same value over $\bar{K}$, and the solving key-conflicts by aggregating with respect to $\oplus$ (coordinate-wise).

The evaluation of a Lara(Ω) expression e over schema σ on a Lara database D, denoted e^D, is inductively defined as follows. Since the definitions are not so easy to grasp, we use the associative tables A and B in Figure 1 to construct examples. Here, **i**, **j**, and **k** are key-attributes, while **v**$_1$, **v**$_2$, and **v**$_3$ are value-attributes.

- *Empty associative table.* if $e = \emptyset$ then $e^D := \emptyset$.
- *Atomic expressions.* If $e = R[\bar{K}, \bar{V}]$, for $R \in \sigma$, then $e^D := R^D$.
- *Join.* If $e[\bar{K}_1 \cup \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \bowtie_{\otimes} e_2[\bar{K}_2, \bar{V}_2]$, then

$$\begin{aligned}e^D \;:=\; \{(\bar{k}_1 \cup \bar{k}_2, \bar{v}_1 \otimes \bar{v}_2) \mid \bar{k}_1 \text{ and } \bar{k}_2 \text{ are compatible tuples such that}\\ \bar{v}_1 = \text{pad}_{\otimes}^{\bar{V}_2}(e_1^D(\bar{k}_1)) \text{ and } \bar{v}_2 = \text{pad}_{\otimes}^{\bar{V}_1}(e_2^D(\bar{k}_2))\}.\end{aligned}$$

Here, $\bar{v}_1 \otimes \bar{v}_2$ is a shortening for $(v_1^1 \otimes v_2^1, \ldots, v_n^1 \otimes v_n^2)$ assuming that $\bar{v}_1 = (v_1^1, \ldots, v_n^1)$ and $\bar{v}_2 = (v_2^1, \ldots, v_2^n)$. For example, the result of $A \bowtie_{\times} B$ is shown in Figure 2, for $\times$ being the usual product on $\mathbb{N}$ and $0_{\times} = 1$.

- *Union.* If $e[\bar{K}_1 \cap \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \mathbin{\rotatebox[origin=c]{90}{$\bowtie$}}_{\oplus} e_2[\bar{K}_2, \bar{V}_2]$, then

$$\begin{aligned}e^D \;:= \mathsf{Solve}_{\oplus}\{\{(\bar{k}, \bar{v}) \mid \bar{k} = \bar{k}_1{\downarrow_{\bar{K}_1 \cap \bar{K}_2}} \text{ and } \bar{v} = \text{pad}_{\oplus}^{\bar{V}_2}(e_1^D(\bar{k}_1)) \text{ for some } \bar{k}_1 \in e_1^D,\\ \text{or } \bar{k} = \bar{k}_2{\downarrow_{\bar{K}_1 \cap \bar{K}_2}} \text{ and } \bar{v} = \text{pad}_{\oplus}^{\bar{V}_1}(e_2^D(\bar{k}_2)) \text{ for some } \bar{k}_2 \in e_2^D\}\}.\end{aligned}$$

In more intuitive terms, e^D is defined by first projecting over $\bar{K}_1 \cap \bar{K}_2$ any tuple in e_1^D, resp., in e_2^D. As the resulting set of tuples might no longer be an associative table (because there might be many tuples with the same keys), we have to solve the conflicts by applying the given aggregate operator $\oplus$. This is what $\mathsf{Solve}_{\oplus}$ does.
For example, the result of $A \mathbin{\rotatebox[origin=c]{90}{$\bowtie$}}_{+} B$ is shown in Figure 2, for $+$ being the addition on $\mathbb{N}$.

(a) Table $A \Join_{\times} B$

i	**j**	**k**	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$
0	0	0	1	5	1
0	0	1	1	5	2
0	1	0	2	6	1
0	1	1	2	12	1
1	0	0	3	7	1
1	0	1	3	7	2
1	1	0	4	8	1
1	1	1	4	16	1

(b) Table $A ⧖_{+} B$

j	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$
0	4	14	3
1	8	17	2

(c) Table $\mathsf{Ext}_g\, A$

i	**j**	**z**
0	0	1
0	1	1

Figure 2 The tables $A \Join_{\times} B$, $A ⧖_{+} B$,and $\mathsf{Ext}_f\, A$.

- *Extend.* If $e[\bar{K} \cup \bar{K}', \bar{V}'] = \mathsf{Ext}_f\, e_1[\bar{K}, \bar{V}]$ and f is of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$, then

  $$e^D := \{(\bar{k} \cup \bar{k}', \bar{v}') \mid (\bar{k}, \bar{v}) \in e_1^D, \text{ and } (\bar{k}', \bar{v}') \in f(\bar{k}, \bar{v})\}.$$

  Notice that in this case $\bar{k} \cup \bar{k}'$ always exists as $\bar{K} \cap \bar{K}' = \emptyset$.

  As an example, Figure 2 shows the results of $\mathsf{Ext}_g\, A$, where g is a function that does the following: If the key corresponding to attribute **i** is 0, then the tuple is associated with the associative table of sort $(\emptyset, \mathbf{z})$ containing only the tuple $(\emptyset, 1)$. Otherwise, the tuple is associated with the empty associative table.

Several useful operators, as described below, can be derived from the previous ones.

- *Map operation.* An important particular case of Ext_f occurs when f is of sort $(\bar{K}, \bar{V}) \mapsto (\emptyset, \bar{V}')$, i.e., when f does not extend the keys in the associative table but only modifies the values. Following [10], we write this operation as Map_f.
- *Aggregation.* This corresponds to an aggregation over some of the key-attributes of an associative table. Consider a LARA expression $e_1[\bar{K}_1, \bar{V}_1]$, an aggregate operator $\oplus$ over Values, and a $\bar{K} \subseteq \bar{K}_1$, then $e = ⧖_{\oplus}^{\bar{K}}\, e_1$ is an expression of sort $(\bar{K}, \bar{V}_1)$ such that $e^D := \mathsf{Solve}_{\oplus}\{\{(\bar{k}, \bar{v}) \mid \bar{k} = \bar{k}_1{\downarrow_{\bar{K}}} \text{ and } \bar{v} = e_1^D(\bar{k}_1)\}\}$. We note that $⧖_{\oplus}^{\bar{K}}\, e_1$ is equivalent to the expression $e_1 ⧖_{\oplus} \mathsf{Ext}_f(\emptyset)$, where f is the function that associates an empty table of sort $(\bar{K}, \emptyset)$ with every possible tuple.
- *Reduction.* The reduction operator, denoted by $\bar{⧖}$, is just a syntactic variation of aggregation defined as $\bar{⧖}_{\oplus}^{\bar{L}}\, e_1 \equiv ⧖_{\oplus}^{\bar{K} \setminus \bar{L}}\, e_1$.

Next we provide an example that applies several of these operators.

▶ **Example 1.** Consider the schema Seqs$[(time, batch, features), (val)]$, which represents a typical tensor obtained as the output of a recurrent neural network that processes input sequences. The structure stores a set of *features* obtained when processing input symbols from a sequence, one symbol at a *time*. For efficiency the network can simultaneously process a *batch* of examples and provide a single tensor as output.

Assume that, in order to make a prediction one wants to first obtain, for every example, the maximum value of every feature over the time steps, and then apply a *softmax* function.

One can specify all this process in Lara as follows.

$$\begin{aligned}
\text{Max} &= \bar{⧖}^{(\mathit{time})}_{\max(\cdot)}\ \text{Seqs} && (1)\\
\text{Exp} &= \mathsf{Map}_{\exp(\cdot)}\text{Max} && (2)\\
\text{SumExp} &= \bar{⧖}^{(\mathit{features})}_{\text{sum}(\cdot)}\ \text{Exp} && (3)\\
\text{Softmax} &= \text{Exp} \bowtie_{\div} \text{SumExp} && (4)
\end{aligned}$$

Expression (1) performs an aggregation over the *time* attribute to obtain the new tensor $\mathsf{Max}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ such that $\mathsf{Max}(b, f) = \max_{u=\mathsf{Seqs}(t,b,f)} u$. That is, Max stores the maximum value over all time steps (for every feature of every example). Expression (2) applies a point-wise exponential function to obtain the tensor $\mathsf{Exp}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ such that $\mathsf{Exp}(b, f) = \exp(\mathsf{Max}(b, f))$. In expression (3) we apply another aggregation to compute the sum of the exponentials of all the (maximum) features. Thus we obtain the tensor $\mathsf{SumExp}[(\mathit{batch}), (\mathit{val})]$ such that

$$\mathsf{SumExp}(b) = \sum_f \mathsf{Exp}(b, f) = \sum_f \exp(\mathsf{Max}(b, f)).$$

Finally, expression (4) applies point-wise division over the tensors $\mathsf{Exp}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ and $\mathsf{SumExp}[(\mathit{batch}), (\mathit{val})]$. This defines a tensor $\mathsf{Softmax}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ such that

$$\mathsf{Softmax}(b, f) = \frac{\mathsf{Exp}(b, f)}{\mathsf{SumExp}(b)} = \frac{\exp(\mathsf{Max}(b, f))}{\sum_{f'} \exp(\mathsf{Max}(b, f'))}.$$

Thus, we effectively compute the *softmax* of the vector of maximum features over time for every example in the batch. ◀

It is easy to see that for each Lara expression e and Lara database D, the result $e(D)$ is always an associative table. Moreover, although the elements in the evaluation $e(D)$ of an expression e over D are not necessarily in D (due to the applications of the operator $\mathsf{Solve}_{\oplus}$ and the extension functions in Ω), all Lara expressions are *safe*, i.e., $|e^D|$ is finite.

▶ **Proposition 2.** *Let e be a* Lara(Ω) *expression. Then e^D is a finite associative table, for every* Lara *database D.*

3 First-order Logic with Aggregation

We consider a two-sorted version of FO with aggregation. We thus assume the existence of two disjoint and countably infinite sets of *key-variables* and *value-variables*. The former are denoted $x, y, z, \ldots$ and the latter $i, j, k, \ldots$. In order to cope with the demands of the extension functions used by Lara (as explained later), we allow the language to be parameterized by a collection Ψ of user-defined relations R of some sort $(\bar{K}, \bar{V})$. For each $R \in \Psi$ we blur the distinction between the symbol R and its interpretation over $\mathsf{Keys}^{|\bar{K}|} \times \mathsf{Values}^{|\bar{V}|}$.

Syntax and semantics

The language contains terms of two sorts.

- *Key-terms*: Composed exclusively by the key-variables $x, y, z \ldots$.
- *Value-terms*: Composed by the constants of the form $0_{\oplus}$, for each aggregate operator $\oplus$, the value-variables $i, j, \ldots$, and the *aggregation terms* defined next. Let $\tau(\bar{x}, \bar{y}, \bar{i}, \bar{j})$ be a value-term mentioning only key-variables among those in $(\bar{x}, \bar{y})$ and value-variables

among those in $(\bar{i},\bar{j})$, and $\phi(\bar{x},\bar{y},\bar{i},\bar{j})$ a formula whose *free* key- and value-variables are those in $(\bar{x},\bar{y})$ and $(\bar{i},\bar{j})$, respectively (i.e., the variables that do not appear under the scope of a quantifier). Then for each aggregate operator $\oplus$ we have that

$$\tau'(\bar{x},\bar{i}) \;:=\; \mathsf{Agg}_{\oplus}\bar{y},\bar{j}\;\big(\tau(\bar{x},\bar{y},\bar{i},\bar{j}),\phi(\bar{x},\bar{y},\bar{i},\bar{j})\big) \tag{5}$$

is a value-term whose free variables are those in $\bar{x}$ and $\bar{i}$.

Let Ψ be a set of relations R as defined above. The set of formulas in the language $\mathrm{FO}_{\mathsf{Agg}}(\Psi)$ over schema σ is inductively defined as follows:

- Atoms $\bot$, $x = y$, and $\iota = \kappa$ are formulas, for x, y key-variables and ι, κ value-terms.
- If $R[\bar{K},\bar{V}] \in \sigma \cup \Psi$, then $R(\bar{x},\bar{\iota})$ is a formula, where $\bar{x}$ is a tuple of key-variables of the same arity as $\bar{K}$ and $\bar{\iota}$ is a tuple of value-terms of the same arity as $\bar{V}$.
- If ϕ,ψ are formulas, then $(\neg\phi)$, $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $\exists x\phi$, and $\exists i\phi$ are formulas, where x and i are key- and value-variables, respectively.

We now define the semantics of $\mathrm{FO}_{\mathsf{Agg}}(\Psi)$. Let D be a LARA database and η an *assignment* that interprets each key-variable x as an element $\eta(x) \in \mathsf{Keys}$ and value-variable i as an element $\eta(i) \in \mathsf{Values}$. If $\tau(\bar{x},\bar{i})$ is a value-term only mentioning variables in $(\bar{x},\bar{i})$, we write $\tau^D(\eta(\bar{x},\bar{i}))$ for the *interpretation* of τ over D when variables are interpreted according to η. Also, if $\phi(\bar{x},\bar{i})$ is a formula of the logic whose free key- and value-variables are those in $(\bar{x},\bar{i})$, we write $D \models \phi(\eta(\bar{x},\bar{i}))$ if D *satisfies* ϕ when $\bar{x},\bar{i}$ is interpreted according to η, and ϕ^D for the set of tuples $\eta(\bar{x},\bar{i})$ such that $D \models \phi(\eta(\bar{x},\bar{i}))$ for some assignment η.

The notion of satisfaction is inherited from the semantics of two-sorted FO. The notion of interpretation, on the other hand, requires explanation for the case of value-terms. Let η be an assignment as defined above. Constants $0_{\oplus}$ are interpreted as themselves and value-variables are interpreted over Values according to η. Consider now an aggregate term of the form (5). Let D be a LARA database and assume that $\eta(\bar{x}) = \bar{k}$, for $\bar{k} \in \mathsf{Keys}^{|\bar{x}|}$, and $\eta(\bar{i}) = \bar{v}$, for $\bar{v} \in \mathsf{Values}^{|\bar{i}|}$. Let $(\bar{k}'_1,\bar{v}'_1),(\bar{k}'_2,\bar{v}'_2),\dots,$ be an enumeration of all tuples $(\bar{k}',\bar{v}') \in \mathsf{Keys}^{|\bar{y}|} \times \mathsf{Values}^{|\bar{j}|}$ such that $D \models \phi(\bar{k},\bar{k}',\bar{v},\bar{v}')$, i.e. there is an assignment η' that coincides with η over all variables in $(\bar{x},\bar{i})$ and satisfies $\eta'(\bar{y},\bar{j}) = (\bar{k}',\bar{v}')$. Then

$$\tau'(\eta(\bar{x},\bar{i})) \;=\; \tau'(\bar{k},\bar{v}) \;:=\; \bigoplus \{\!\{\tau(\bar{k},\bar{k}'_1,\bar{v},\bar{v}'_1),\tau(\bar{k},\bar{k}'_2,\bar{v},\bar{v}'_2),\dots\}\!\} \in \mathsf{Values}.$$

4 Expressive Completeness of LARA with respect to $\mathrm{FO}_{\mathsf{Agg}}$

We prove that LARA(Ω) has the same expressive power as a suitable restriction of $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$, where Ψ_Ω is a set that contains relations that represent the extension functions in Ω. In particular, for every extension function $f \in \Omega$ of sort $(\bar{K},\bar{V}) \mapsto (\bar{K}',\bar{V}')$, there is a relation $R_f \subseteq \mathsf{Keys}^{|\bar{K}|+|\bar{K}'|} \times \mathsf{Values}^{|\bar{V}|+|\bar{V}'|}$ in Ψ_Ω such that for every $(\bar{k},\bar{v}) \in \mathsf{Keys}^{|\bar{K}|} \times \mathsf{Values}^{|\bar{V}|}$:

$$f(\bar{k},\bar{v}) \;=\; \{(\bar{k}',\bar{v}') \mid (\bar{k},\bar{k}',\bar{v},\bar{v}') \in R_f\}.$$

Since LARA is defined in a minimalistic way, we require some assumptions for our expressive completeness result to hold. First, we assume that $\mathsf{Keys} = \mathsf{Values}$, which allows us to interchangeably move from keys to values in the language (an operation that LARA routinely performs in several of its applications [10, 9]). Moreover, we assume that there are two reserved values, $\diamondsuit$ and $\heartsuit$, which are allowed to be used as constants in both $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ and LARA, but do not appear in any LARA database. In particular, we allow $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ to express formulas of the form $x = c$ and $i = c$, for x and i key- and value-variables, respectively, and $c \in \{\diamondsuit, \heartsuit\}$. (Notice that, by definition, $\diamondsuit$ and $\heartsuit$ are different to all neutral elements of the form $0_\oplus$, for $\oplus$ an aggregate operator). With these constants we can "mark" tuples in some specific cases, and thus solve an important semantic mismatch between the two languages. In fact, LARA deals with multisets in their semantics while FO_{Agg} is based on sets only. This causes problems, e.g., when taking the union of two associative tables A and B both of which contain an occurrence of the same tuple $(\bar{k}, \bar{v})$. While LARA would treat both occurrences of $(\bar{k}, \bar{v})$ as different in $A \mathbin{\rotatebox[origin=c]{90}{$\bowtie$}} B$, and hence would be forced to restore the "key-functionality" of $\bar{k}$ based on some aggregate operator, for FO_{Agg} the union of A and B contains only one occurrence of $(\bar{k}, \bar{v})$.

From LARA to FO_{Agg}

We show first that the expressive power of LARA(Ω) is bounded by that of $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$.

▶ **Theorem 3.** *For every expression $e[\bar{K}, \bar{V}]$ of LARA(Ω) there is a formula $\phi_e(\bar{x}, \bar{i})$ of $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ such that $e^D = \phi_e^D$, for every LARA database D.*

Before proving the theorem, we present an example of how the join operation is translated, as this provides a good illustration of the main ideas behind the proof. Assume that we are given expressions $e_1[K_1, K_2, K_3, V_1, V_2]$ and $e_2[K_3, K_2, V_2, V_3]$ of LARA(Ω). Inductively, there are formulas $\phi_{e_1}(x_1, x_2, x_3, i_1, i_2)$ and $\phi_{e_2}(x_3', x_2', i_2', i_3)$ of $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ such that $e_1^D = \phi_{e_1}^D$ and $e_2^D = \phi_{e_2}^D$, for every LARA database D. We want to be able to express $e = e_1 \bowtie_\otimes e_2$ in $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$. Let us define a formula $\alpha(x, y, z, i, j, k, f)$ as

$$\exists i', j', k' \Big(\phi_{e_1}(x, y, z, i', j') \wedge \phi_{e_2}(y, z, j', k') \wedge \\ \big((i = i' \wedge j = j' \wedge k = 0_\otimes \wedge f = \diamondsuit) \vee (i = 0_\otimes \wedge j = j' \wedge k = k' \wedge f = \heartsuit) \big) \Big).$$

Notice that when evaluating α over a LARA database D we obtain the set of tuples $(k_1, k_2, k_3, v_1, v_2, v_3, f)$ such that there exist tuples of the form $(k_1, k_2, k_3, \cdot, \cdot) \in e_1^D$ and $(k_2, k_3, \cdot, \cdot) \in e_2^D$, and either one of the following holds:

- $e_1^D(k_1, k_2, k_3) = (v_1, v_2)$; $(v_1, v_2, v_3) = \text{pad}_\otimes^{(V_2, V_3)}(v_1, v_2)$; and $f = \diamondsuit$; or
- $e_2^D(k_2, k_3) = (v_2, v_3)$; $(v_1, v_2, v_3) = \text{pad}_\otimes^{(V_1, V_2)}(v_2, v_3)$; and $f = \heartsuit$.

The reason why we want to distinguish tuples from e_1^D or e_2^D with a $\diamondsuit$ or a $\heartsuit$ in the position of variable f, respectively, it is because it could be the case that $\text{pad}_\otimes^{(V_2, V_3)}(v_1, v_2) = \text{pad}_\otimes^{(V_1, V_2)}(v_2, v_3)$. The semantics of LARA, which is based on aggregation over multisets of tuples, forces us to treat them as two different tuples. The way we do this in FO_{Agg} is by distinguishing them with the extra flag f.

We finally define $\phi_e(x, y, z, i, j, k)$ in $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ as

$$\exists i', j', k', f' \Big(\alpha(x, y, z, i', j', k', f) \wedge i = \mathsf{Agg}_\otimes i', j', k', f' \left(i',\, \alpha(x, y, z, i', j', k', f') \right) \wedge \\ j = \mathsf{Agg}_\otimes i', j', k', f' \left(j',\, \alpha(x, y, z, i', j', k', f') \right) \wedge \\ k = \mathsf{Agg}_\otimes i', j', k', f' \left(k',\, \alpha(x, y, z, i', j', k', f') \right) \Big).$$

That is, the evaluation of ϕ_e on D outputs all tuples $(k_1, k_2, k_3, v_1, v_2, v_3)$ such that:

1. There are tuples $(k_1, k_2, k_3, w_1, w_2, 0_\otimes, \cdot, \diamondsuit)$ and $(k_1, k_2, k_3, 0_\otimes, w_2', w_3', \heartsuit)$ in α^D.
2. $v_1 = w_1 \otimes 0_\otimes = w_1$; $v_2 = w_2 \otimes w_2'$; and $v_3 = 0_\otimes \otimes w_3' = w_3'$.

Clearly, then, $\phi_e^D = e^D$ over every Lara database D.

Proof of Theorem 3. By induction on e.

- If $e = \emptyset$, then $\phi_e = \bot$.
- If $e = R[\bar{K}, \bar{V}]$, for $R \in \sigma$, then $\phi_e(\bar{x}, \bar{i}) = R(\bar{x}, \bar{i})$, where $\bar{x}$ and $\bar{i}$ are tuples of distinct key- and value-variables of the same arity as $\bar{K}$ and $\bar{V}$, respectively.
- Consider the expression $e[\bar{K}_1 \cup \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \bowtie_{\otimes} e_2[\bar{K}_2, \bar{V}_2]$, and assume that $\phi_{e_1}(\bar{x}_1, \bar{i}_1)$ and $\phi_{e_2}(\bar{x}_2, \bar{i}_2)$ are the formulas obtained for $e_1[\bar{K}_1, \bar{V}_1]$ and $e_2[\bar{K}_2, \bar{V}_2]$, respectively, by induction hypothesis. Let us first define a formula $\alpha_e(\bar{x}_1, \bar{x}_2, \bar{j}, f)$ as

$$\exists \bar{i}_1, \bar{i}_2 \Bigg(\phi_{e_1}(\bar{x}_1, \bar{i}_1) \wedge \phi_{e_2}(\bar{x}_2, \bar{i}_2) \wedge \chi_{\bar{K}_1 \cap \bar{K}_2}(\bar{x}_1, \bar{x}_2) \wedge \\ \big((\bar{j} = \mathrm{pad}_{\otimes}^{\bar{V}_2}(\bar{i}_1) \wedge f = \diamondsuit) \vee (\bar{j} = \mathrm{pad}_{\otimes}^{\bar{V}_1}(\bar{i}_2) \wedge f = \heartsuit) \big) \Bigg),$$

  assuming that $\bar{x}_1$ and $\bar{x}_2$ share no variables; the same holds for $\bar{i}_1$ and $\bar{i}_2$; and the formula $\chi_{\bar{K}_1 \cap \bar{K}_2}(\bar{x}_1, \bar{x}_2)$ takes the conjunction of all atomic formulas of the form $y_1 = y_2$, where y_1 and y_2 are variables that appear in $\bar{x}_1$ and $\bar{x}_2$, respectively, in the position of some attribute $K \in \bar{K}_1 \cap \bar{K}_2$. Notice that $\bar{j} = \mathrm{pad}_{\otimes}^{\bar{V}_2}(\bar{i}_1)$ is expressible in $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ as a conjunction of formulas of the form $j = k$, for each $j \in \bar{j}$, where k is the corresponding variable of $\bar{i}_1$ if j falls in the position of an attribute in $\bar{V}_1$, and it is is $0_{\otimes}$ otherwise. For instance, if $\bar{i} = (i_1, i_2)$ is of sort $\bar{V}_1 = (V_1, V_2)$ and $\bar{V}_2 = (V_2, V_3)$, then $\bar{j} = \mathrm{pad}_{\otimes}^{\bar{V}_2}(\bar{i})$ is the formula $j_1 = i_1 \wedge j_2 = i_2 \wedge j_3 = 0_{\otimes}$. Analogously we can define $\bar{j} = \mathrm{pad}_{\otimes}^{\bar{V}_1}(\bar{i}_2)$. As in the example given before this proof, we use the value of f to distinguish whether a tuple comes from e_1^D or e_2^D.

  It is easy to see that for every Lara database D we have that α_e^D computes all tuples of the form $(\bar{k}_1, \bar{k}_2, \bar{v}, w)$ such that $\bar{k}_1 \in (\exists \bar{i}_1 \phi_{e_1})^D$, $\bar{k}_2 \in (\exists \bar{i}_2 \phi_{e_2})^D$, and $\bar{k}_1$ and $\bar{k}_2$ are compatible tuples, and either one of the following statements hold:
  - $\bar{v}_1 = e_1^D(\bar{k}_1)$, $\bar{v} = \mathrm{pad}_{\otimes}^{\bar{V}_2}(\bar{v}_1)$, and $w = \diamondsuit$; or
  - $\bar{v}_2 = e_2^D(\bar{k}_2)$, $\bar{v} = \mathrm{pad}_{\otimes}^{\bar{V}_1}(\bar{v}_2)$, and $w = \heartsuit$.

  Notice then that for every $(\bar{k}_1, \bar{k}_2)$ that belongs to the evaluation of $\exists \bar{j}, f\, \alpha_e(\bar{x}_1, \bar{x}_2, \bar{j}, f)$ over D there are exactly two tuples of the form $(\bar{k}_1, \bar{k}_2, \bar{v}, w) \in \alpha_e^D$: the one which satisfies $\bar{v} = \mathrm{pad}_{\otimes}^{\bar{V}_2}(e_1^D(\bar{k}_1))$ and $w = \diamondsuit$, and the one that satisfies $\bar{v} = \mathrm{pad}_{\otimes}^{\bar{V}_1}(e_2^D(\bar{k}_2))$ and $w = \heartsuit$. It should be clear then that $\phi_e(\bar{x}_1, \bar{x}_2, \bar{i})$ can be expressed as:

$$\exists \bar{j}, f\, \alpha_e(\bar{x}_1, \bar{x}_2, \bar{j}, f) \wedge \bigwedge_{\ell \in [|\bar{j}|]} \bar{i}[\ell] = \mathsf{Agg}_{\otimes} \bar{j}, f\, (\bar{j}[\ell], \alpha_e(\bar{x}, \bar{j}, f)).$$

  Notice here that the aggregation is always performed on multisets with exactly two elements (by our previous observation). Clearly, the evaluation of ϕ_e on D, for D a Lara database, contains all tuples $(\bar{k}_1 \cup \bar{k}_2, \bar{v}_1 \otimes \bar{v}_2)$ such that $\bar{k}_1$ and $\bar{k}_2$ are compatible tuples in e_1^D and e_2^D, respectively, and it is the case that $\bar{v}_1 = \mathrm{pad}_{\otimes}^{\bar{V}_2}(e_1^D(\bar{k}_1))$ and $\bar{v}_2 = \mathrm{pad}_{\otimes}^{\bar{V}_1}(e_2^D(\bar{k}_2))$.
- Consider the expression $e[\bar{K}_1 \cap \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \mathbin{⧖}_{\oplus} e_2[\bar{K}_2, \bar{V}_2]$, and assume that $\phi_{e_1}(\bar{x}_1, \bar{i}_1)$ and $\phi_{e_2}(\bar{x}_2, \bar{i}_2)$ are the formulas obtained for $e_1[\bar{K}_1, \bar{V}_1]$ and $e_2[\bar{K}_2, \bar{V}_2]$, respectively, by induction hypothesis. We start by defining a formula $\alpha_{e_1}(\bar{x}, \bar{j}, f)$ as

$$\exists \bar{x}_1, \bar{i}_1 \left(\phi_{e_1}(\bar{x}_1, \bar{i}_1) \wedge \eta_1^{\bar{K}_1 \cap \bar{K}_2}(\bar{x}, \bar{x}_1) \wedge \bar{j} = \mathrm{pad}_{\oplus}^{\bar{V}_2}(\bar{i}_1) \wedge f = \diamondsuit \right),$$

where $\eta_1^{\bar{K}_1 \cap \bar{K}_2}(\bar{x}, \bar{x}_1)$ states that $\bar{x}$ is the extension of $\bar{x}_1$ to represent a tuple in $\bar{K}_1 \cup \bar{K}_2$. Formally, each variable in $\bar{x}$ that represents a position in $\bar{K}_1$ receives the same value than the variable in the corresponding position of $\bar{x}$, and each variable representing a position in $\bar{K}_2 \setminus \bar{K}_1$ receives value $\diamondsuit$. As an example, if $\bar{K}_1 = (K_1, K_2)$ and $\bar{K}_2 = (K_1, K_3)$, then $\eta_1^{\bar{K}_1 \cup \bar{K}_2}(\bar{x}, \bar{x}_1)$ for $\bar{x}_1 = (y_1, y_2)$ and $\bar{x} = (z_1, z_2, z_3)$ is $z_1 = y_1 \wedge z_2 = y_2 \wedge z_3 = 0$. Analogously, we define $\alpha_{e_2}(\bar{x}, \bar{j}, f)$ as

$$\exists \bar{x}_2, \bar{i}_2 \left(\phi_{e_2}(\bar{x}_2, \bar{i}_2) \wedge \eta_2^{\bar{K}_1 \cap \bar{K}_2}(\bar{x}, \bar{x}_2) \wedge \bar{j} = \mathrm{pad}_{\oplus}^{\bar{V}_1}(\bar{i}_2) \wedge f = \heartsuit \right).$$

As before, we use distinguished constants $\diamondsuit$ and $\heartsuit$ as a way to distinguish tuples coming from e_1^D and e_2^D, respectively.
Let us now define $\alpha(\bar{x}, \bar{j}, f) := \alpha_{e_1}(\bar{x}, \bar{j}, f) \vee \alpha_{e_2}(\bar{x}, \bar{j}, f)$. It is not hard to see then that the evaluation of α_e on a Lara database D consists precisely of the tuples of the form $(\bar{k}, \bar{v}, w)$ such that one of the following statements hold:

- $\bar{k}_1 = \bar{k}{\downarrow}_{\bar{K}_1}$ for some $\bar{k}_1 \in (\exists \bar{i}_1 \phi_{e_1})^D$; $\bar{k}$ takes value $\diamondsuit$ for those positions in $\bar{K}_2 \setminus \bar{K}_1$; $\bar{v} = \mathrm{pad}_{\oplus}^{\bar{V}_2}(e_1^D(\bar{k}_1))$; and $w = \diamondsuit$; or
- $\bar{k}_2 = \bar{k}{\downarrow}_{\bar{K}_2}$ for some $\bar{k}_2 \in (\exists \bar{i}_2 \phi_{e_2})^D$; $\bar{k}$ takes value $\heartsuit$ for those positions in $\bar{K}_1 \setminus \bar{K}_2$; $\bar{v} = \mathrm{pad}_{\oplus}^{\bar{V}_1}(e_2^D(\bar{k}_1))$; and $w = \heartsuit$.

Let us write $\alpha(\bar{x}, \bar{j}, f)$ as $\alpha(\bar{x}', \bar{x}'', \bar{j}, f)$ to denote that $\bar{x}'$ is the subtuple of $\bar{x}$ that corresponds to variables in $\bar{K}_1 \cap \bar{K}_2$, while $\bar{x}''$ contains all other variables in $\bar{x}$. Notice that in the output of our desired formula ϕ_e we are only interested in the value that takes the tuple $\bar{x}'$. It should be clear then that $\phi_e(\bar{x}, \bar{i})$ can be expressed as:

$$\exists \bar{x}', \bar{x}'', \bar{j}, f \Big(\bar{x} = \bar{x}' \wedge \alpha(\bar{x}', \bar{x}'', \bar{j}, f) \wedge \bigwedge_{\ell \in [|\bar{j}|]} \bar{i}[\ell] = \mathsf{Agg}_{\oplus} \bar{x}'', \bar{j}, f \left(\bar{j}[\ell], \alpha(\bar{x}', \bar{x}'', \bar{j}, f) \right) \Big).$$

- Consider the expression $e[\bar{K} \cup \bar{K}', \bar{V}'] = \mathsf{Ext}_f\, e_1[\bar{K}, \bar{V}]$, where f is of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$, and assume that $\phi_{e_1}(\bar{x}_1, \bar{i}_1)$ is the formula obtained for $e_1[\bar{K}_1, \bar{V}_1]$ by induction hypothesis. It is straightforward to see then that we can define

$$\phi_e(\bar{x}_1, \bar{x}, \bar{i}) := \exists \bar{i}_1 \left(\phi_{e_1}(\bar{x}_1, \bar{i}_1) \wedge R_f(\bar{x}_1, \bar{x}, \bar{i}_1, \bar{i}) \right).$$

This finishes the proof of the theorem. ◀

It is worth noticing that the translation from Lara to $\mathrm{FO}_{\mathsf{Agg}}$ given in the proof of Theorem 3 does not require the use of negation. In the next section we show that at least safe negation can be encoded in Lara by a suitable combination of aggregate operators and extension functions.

From $\mathrm{FO}_{\mathsf{Agg}}$ to LARA

We now prove that the other direction holds under suitable restrictions and assumptions on the language. First, we need to impose two restrictions on $\mathrm{FO}_{\mathsf{Agg}}$ formulas, which ensure that the semantics of the formulas considered matches that of Lara. In particular, we need to ensure that the evaluation of $\mathrm{FO}_{\mathsf{Agg}}$ formulas is safe and only outputs associative tables.

- *Safety.* Formulas of $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ are not necessarily safe, i.e., their evaluation can have infinitely many tuples (think, e.g., of the formula $i = j$, for i, j value-variables, or $R_f(\bar{x}, \bar{x}', \bar{i}, \bar{i}')$, for $R_f \in \Psi_\Omega$). While safety issues relating to the expressive completeness of relational algebra with respect to first order logic are often resolved by relativizing

all operations to the *active* domain of databases (i.e., the set of elements mentioned in relations in databases), such a restriction only makes sense for keys in our context, but not for values. In fact, several useful formulas compute a new value for a variable based on some aggregation terms over precomputed data (see, e.g., the translations of the join and union operator of Lara into $\mathrm{FO}_{\mathsf{Agg}}$ provided in the proof of Theorem 3).
To overcome this issue we develop a suitable syntactic restriction of the logic that can only express safe queries. This is achieved by "guarding" the application of value-term equalities, relations encoding extension functions, and Boolean connectives as follows.

- We only allow equality of value-terms to appear in formulas of the form $\phi(\bar{x}, \bar{i}) \wedge j = \tau(\bar{x}, \bar{i})$, where j is a value-variable that does not necessarily appear in $\bar{i}$ and τ is an arbitrary value-term whose value only depends on $(\bar{x}, \bar{i})$. This formula computes the value of the aggregated term τ over the precomputed evaluation of ϕ, and then output it as the value of j. In the same vein, atomic formulas of the form $R(\bar{x}, \bar{\iota})$ must satisfy that every element in $\bar{\iota}$ is a value-variable.
- Relations $R_f \in \Psi_\Omega$ can only appear in formulas of the form $\phi(\bar{x}, \bar{i}) \wedge R_f(\bar{x}, \bar{x}', \bar{i}, \bar{i}')$, i.e., we only allow to compute the set $f(\bar{x}, \bar{i})$ for specific precomputed values of $(\bar{x}, \bar{i})$.
- Also, negation is only allowed in the restricted form $\phi(\bar{x}, \bar{i}) \wedge \neg\psi(\bar{x}, \bar{i})$ and disjunction in the form $\phi(\bar{x}, \bar{i}) \vee \psi(\bar{x}, \bar{i})$, i.e., when formulas have exactly the same free variables.

We denote the resulting language as $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$. These restrictions are meaningful, as the translation from Lara(Ω) to $\mathrm{FO}_{\mathsf{Agg}}(\Omega_\Psi)$ given in the proof of Theorem 3 always builds a formula in $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$.

- *Key constraints.* We also need a restriction on the interpretation of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ formulas that ensures that the evaluation of any such a formula on a Lara database is an associative table. For doing this, we modify the syntax of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ formulas in such a way that every formula ϕ of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ now comes equipped with an aggregate operator $\oplus$ over Values. The operator $\oplus$ is used to "solve" the key violations introduced by the evaluation of ϕ. Thus, formulas in this section should be understood as pairs $(\phi, \oplus)$. The evaluation of $(\phi, \oplus)$ over a Lara database D, denoted $\phi^D_\oplus$, is $\mathsf{Solve}_\oplus(\phi^D)$. This definition is recursive; e.g., a formula in $\phi(\bar{x}, \bar{i})$ in $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ which is of the form $\alpha(\bar{x}, \bar{i}) \vee \beta(\bar{x}, \bar{i})$ should now be specified as $(\phi, \oplus) = (\alpha, \oplus_\alpha) \vee (\beta, \oplus_\beta)$. The associative table $\phi^D_\oplus$ corresponds then to

$$\mathsf{Solve}_\oplus(\alpha^D_{\oplus_\alpha} \cup \beta^D_{\oplus_\beta}).$$

We also require some natural assumptions on the extension functions that Lara is allowed to use. In particular, we need these functions to be able to express traditional relational algebra operations that are not included in the core of Lara; namely, copying attributes, selecting rows based on (in)equality, and projecting over value-attributes (the projection over key-attributes, in turn, can be expressed with the union operator). Formally, we assume that Ω contains the following families of extension functions.

- $\mathsf{copy}_{\bar{K},\bar{K}'}$ and $\mathsf{copy}_{\bar{V},\bar{V}'}$, for $\bar{K}, \bar{K}'$ tuples of key-attributes of the same arity and $\bar{V}, \bar{V}'$ tuples of value-attributes of the same arity. Function $\mathsf{copy}_{\bar{K},\bar{K}'}$ takes as input a tuple $t = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V})$, where $\bar{K} \subseteq \bar{K}_1$ and $\bar{K}' \cap \bar{K}_1 = \emptyset$, and produces a tuple $t' = (\bar{k}, \bar{k}', \bar{v})$ of sort $(\bar{K}_1, \bar{K}', \bar{V})$ such that $t'(\bar{K}') = t(\bar{K})$, i.e., $\mathsf{copy}_{\bar{K},\bar{K}'}$ copies the value of attributes $\bar{K}$ in the new attributes $\bar{K}'$. Analogously, we define the function $\mathsf{copy}_{\bar{V},\bar{V}'}$.
- $\mathsf{copy}_{\bar{V},\bar{K}}$, for $\bar{V}$ a tuple of value-attributes and $\bar{K}$ a tuple of key-attributes. It takes as input a tuple $t = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V}_1)$, where $\bar{V} \subseteq \bar{V}_1$ and $\bar{K} \cap \bar{K}_1 = \emptyset$, and produces a tuple $t' = (\bar{k}, \bar{k}', \bar{v})$ of sort $(\bar{K}_1, \bar{K}, \bar{V})$ such that $t'(\bar{V}) = t'(\bar{K})$, i.e., this function copies the values in $\bar{V}$ as keys in $\bar{K}$. (Here it is important our assumption that $\mathsf{Keys} = \mathsf{Values}$). The reason why this is useful is because Lara does not allow to aggregate with respect to values (only with respect to keys), while $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ can clearly do this. Analogously, we define $\mathsf{copy}_{\bar{K},\bar{V}}$, but this time we copy keys in $\bar{K}$ to values in $\bar{V}$.

- $\mathsf{add}_{V,0_\oplus}$, for V an attribute-value and $\oplus$ an aggregate operator. Function $\mathsf{add}_{V,0_\oplus}$ takes as input a tuple $t = (\bar{k}, \bar{v}')$ of sort $(\bar{K}, \bar{V}')$, where $V \notin \bar{V}'$, and produces a tuple $t' = (\bar{k}, \bar{v}', 0_\oplus)$ of sort $(\bar{K}, \bar{V}', V)$, i.e., $\mathsf{add}_{V,0_\oplus}$ adds a new value-attribute V that always takes value $0_\oplus$. Analogously, we define functions $\mathsf{add}_{V,\diamondsuit}$ and $\mathsf{add}_{V,\heartsuit}$.
- $\mathsf{eq}_{\bar{K},\bar{K}'}$ and $\mathsf{eq}_{\bar{V},\bar{V}'}$, for $\bar{K}, \bar{K}'$ tuples of key-attributes of the same arity and $\bar{V}, \bar{V}'$ tuples of value-attributes of the same arity. The function $\mathsf{eq}_{\bar{K},\bar{K}'}$ takes as input a tuple $t = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V})$, where $\bar{K}, \bar{K}' \subseteq \bar{K}_1$, and produces as output the tuple $t' = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V})$, if $t(\bar{K}) = t(\bar{K}')$, and the empty associative table otherwise. Hence, this function acts as a filter over an associative table of sort $(\bar{K}_1, \bar{V})$, extending only those tuples t such that $t(\bar{K}) = t(\bar{K}')$. Analogously, we define the function $\mathsf{eq}_{\bar{V},\bar{V}'}$.
- In the same vein, extension functions $\mathsf{neq}_{\bar{K},\bar{K}'}$ and $\mathsf{neq}_{\bar{V},\bar{V}'}$, for $\bar{K}, \bar{K}'$ tuples of key-attributes of the same arity and $\bar{V}, \bar{V}'$ tuples of value-attributes of the same arity. These are defined exactly as $\mathsf{eq}_{\bar{K},\bar{K}'}$ and $\mathsf{eq}_{\bar{V},\bar{V}'}$, only that we now extend only those tuples t such that $t(\bar{K}) \neq t(\bar{K}')$ and $t(\bar{V}) \neq t(\bar{V}')$, respectively.
- The projection $\pi_{\bar{V}}$, for $\bar{V}$ a tuple of value-attributes, takes as input a tuple $(\bar{k}, \bar{v}')$ of sort $(\bar{K}, \bar{V}')$, where $\bar{V} \subseteq \bar{V}'$, and outputs the tuple $(\bar{k}, \bar{v})$ of sort $(\bar{K}, \bar{V})$ such that $\bar{v} = \bar{v}'\!\downarrow_{\bar{V}}$.

We now establish our result.

▶ **Theorem 4.** *Let us assume that Ω contains all extension functions specified above. For every pair $(\phi, \oplus)$, where $\phi(\bar{x}, \bar{\imath})$ is a formula of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ and $\oplus$ is an aggregate operator on* Values*, there is a* Lara(Ω) *expression $e_{\phi,\oplus}[\bar{K}, \bar{V}]$ such that $e^D_{\phi,\oplus} = \phi^D_\oplus = \mathsf{Solve}_\oplus(\phi^D)$, for each* Lara *database D.*

Proof. When f is one of the distinguished extension functions f defined above, we abuse notation and write simply f instead of Ext_f. We first define several useful operations and expressions.

- The projection $\pi^\oplus_{\bar{K}}\, e$ over keys with respect to aggregate operator $\oplus$, defined as $⧖^{\bar{K}}_\oplus\ e$. Notice that this removes key-, but not value-attributes from e, i.e., if e is of sort $[\bar{K}', \bar{V}]$ then $\pi^\oplus_{\bar{K}}\, e$ is of sort $[\bar{K}, \bar{V}]$.
- The *rename* operator $\rho_{\bar{K}\to\bar{K}'}\, e$ as $\pi_{\bar{K}'}\,(\mathsf{copy}_{\bar{K},\bar{K}'}\, e)$, where π has no superscript $\oplus$ as no aggregation is necessary in this case. This operation simply renames the key-attributes $\bar{K}$ to a fresh set of key-attributes $\bar{K}'$. Analogously, we define $\rho_{\bar{V}\to\bar{V}'}\, e$, $\rho_{\bar{V}\to\bar{K}} e$ and $\rho_{\bar{K}\to\bar{V}} e$.
- The *active domain* expression e_{ActDom}, which takes as input a Lara database D and returns all elements $k \in \mathsf{Keys}$ that appear in some fact of D. It is defined as follows. First choose a key attribute not present in any table of D; say it is Z. For each $R[\bar{K}, \bar{V}] \in \sigma$ we define an expression $R^{\mathsf{Keys}} := \pi_\emptyset\, R$, which removes all attribute-values in $\bar{V}$ from R. For each $K \in \bar{K}$ we then define $R^{\mathsf{Keys}}_K := \pi_K\, R^{\mathsf{Keys}}$ as the set of keys that appear in the position of attribute K in $R[\bar{K}, \bar{V}]$ (no need to specify superscript $\oplus$ on π in this case). Finally, we define $e^R_{\mathsf{ActDom}} := ⧖_{K\in\bar{K}}\ \rho_{K\to Z} R^{\mathsf{Keys}}_K$ and $e_{\mathsf{ActDom}} := ⧖_{R\in\sigma}\ e^R_{\mathsf{ActDom}}$.

We now prove the theorem by induction on ϕ.

- If $\phi = \bot$ then $e_{\phi,\oplus} = \emptyset$ for every aggregate operator $\oplus$.
- If $\phi = (x = y)$, for x, y key-variables, then $e_{\phi,\oplus}[K, K'] := \mathsf{eq}_{K,K'}\big(e_{\mathsf{ActDom}}[K] \bowtie \rho_{K\to K'} e_{\mathsf{ActDom}}[K]\big)$ for every aggregate operator $\oplus$. Notice that there is no need to specify an aggregate operator for $\bowtie$ in this case as the tables that participate in the join only consist of keys.
- Consider now $\phi = R(\bar{x}, \bar{\imath})$, for $R \in \sigma$. We assume all variables in $\bar{x}$ and $\bar{\imath}$, respectively, to be pairwise distinct, as repetition of variables can always be simulated with equalities. Then $e_{\phi,\oplus}[\bar{K}, \bar{V}] := R[\bar{K}, \bar{V}]$ for every aggregate operator $\oplus$.

- Assume that $(\phi, \oplus) = (\phi', \oplus') \wedge \neg(\phi'', \oplus'')$. Let $e_{\phi',\oplus'}[\bar{K}, \bar{V}]$ and $e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$ be the expressions obtained for $(\phi', \oplus')$ and $(\phi'', \oplus'')$, respectively, by induction hypothesis. We construct the expression $e_{\phi,\oplus}$ as follows.
  - First, we take the union of $e_{\phi',\oplus'}$ and $e_{\phi'',\oplus''}$, resolving conflicts with an aggregate operator func that simply checks for which tuples of keys it requires to restore "key-functionality" after performing the union. In particular, func takes as input a multiset of values. If it contains more than one element, it returns the distinguished symbol $\diamondsuit$ which appears in no Lara database D. Otherwise it returns the only element in the multiset. For instance, $\mathsf{func}(\{\!\{a, a\}\!\}) = \diamondsuit$ and $\mathsf{func}(\{\!\{a\}\!\}) = a$.
  Let us define then $e_1[\bar{K}, \bar{V}] := e_{\phi',\oplus'} ⧖_{\mathsf{func}}\, e_{\phi'',\oplus''}$. Notice that e_1^D, for D a Lara database, contains the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\oplus'}$ such that there is no tuple of the form $(\bar{k}, \bar{v}') \in e^D_{\phi'',\oplus''}$, plus the tuples of the form $(\bar{k}, \diamondsuit, \dots, \diamondsuit)$ such that there are tuples of the form $(\bar{k}, \bar{v}') \in e^D_{\phi',\oplus'}$ and $(\bar{k}, \bar{v}'') \in e^D_{\phi'',\oplus''}$. In other words, by evaluating e_1 on D we have marked with $(\diamondsuit, \dots, \diamondsuit)$ those tuples $\bar{k}$ of keys that are candidates to be removed when computing the difference $e_{\phi',\oplus'} \setminus e_{\phi'',\oplus''}$.
  - Second, we take the join $e_{\phi',\oplus'}[\bar{K}, \bar{V}] \bowtie e_1'[\bar{K}, \bar{V}']$, where $e_1'[\bar{K}, \bar{V}'] := \rho_{\bar{V} \to \bar{V}'} e_1[\bar{K}, \bar{V}]$ is obtained by simply renaming $\bar{V}$ as $\bar{V}'$ in e_1 (there is no need to specify an aggregate operator for $\bowtie$ as $\bar{V}$ and $\bar{V}'$ have no attributes in common), and apply the extension function $\mathsf{eq}_{\bar{V},\bar{V}'}$ over it. It is easy to see that when evaluating the resulting expression $e_\alpha[\bar{K}, \bar{V}] := \mathsf{eq}_{\bar{V},\bar{V}'}(e_{\phi',\oplus'} \bowtie e_1')$ on a Lara database D, we obtain precisely the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\otimes'}$ such that there is no tuple of the form $(\bar{k}, \bar{w}) \in e^D_{\phi'',\otimes''}$. In fact, for those tuples we also have that $(\bar{k}, \bar{v})$ belongs to e_1, as explained above, and thus $\mathsf{eq}_{\bar{V},\bar{V}'}$ applies no filter. On the contrary, if there is a tuple of the form $(\bar{k}, \bar{w}) \in e^D_{\phi'',\oplus''}$ then $(\bar{k}, \diamondsuit, \dots, \diamondsuit) \in e_1^D$ as explained above. This means that the filter $\mathsf{eq}_{\bar{V},\bar{V}'}$ is applied and no tuple of the form $(\bar{k}, \dots)$ appears in the result. (Notice that for the latter to hold we use, in an essential way, the "key-functionality" of tables $e^D_{\phi',\oplus'}$ and $e^D_{\phi'',\oplus''}$ and the fact that $\diamondsuit$ does not appear in D). By definition, then, $e^D_\alpha \subseteq e^D_{\phi,\oplus}$.
  - Third, we take the join $e_{\phi',\oplus'}[\bar{K}, \bar{V}] \bowtie e'_{\phi'',\oplus''}[\bar{K}, \bar{V}']$, where

  $$e'_{\phi'',\oplus''}[\bar{K}, \bar{V}'] := \rho_{\bar{V} \to \bar{V}'} e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$$

  is obtained by simply renaming $\bar{V}$ as $\bar{V}'$ in $e_{\phi'',\oplus''}$ (there is no need to specify an aggregate operator for $\bowtie$ as $\bar{V}$ and $\bar{V}'$ have no attributes in common), and apply the extension function $\mathsf{neq}_{\bar{V},\bar{V}'}$ over it. It is easy to see that when evaluating the resulting expression $e_\beta[\bar{K}, \bar{V}] := \pi_{\bar{V}} \mathsf{neq}_{\bar{V},\bar{V}'}(e_{\phi',\oplus'} \bowtie e'_{\phi'',\oplus''})$ on a Lara database D, we obtain precisely the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\otimes'}$ such that there is a tuple of the form $(\bar{k}, \bar{w}) \in e^D_{\phi'',\oplus''}$ for which $\bar{v} \neq \bar{w}$. This means that the evaluation of e_β on D contains precisely the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\oplus'}$ that do not belong to e^D_α, yet they belong to $e^D_{\phi',\oplus'} \setminus e^D_{\phi'',\oplus''}$. (Notice that for the latter to hold we use, in an essential way, the "key-functionality" of tables $e^D_{\phi',\oplus'}$ and $e^D_{\phi'',\oplus''}$).
  - Summing up, we can now define $e_{\phi,\oplus}[\bar{K}, \bar{V}]$ as $e_\alpha[\bar{K}, \bar{V}] ⧖ e_\beta[\bar{K}, \bar{V}]$. Notice that there is no need to specify an aggregate operator for ⧖ here, as by construction we have that there are no tuples $\bar{k}$ of keys that belong to both $\pi_{\bar{K}} e^D_\alpha$ and $\pi_{\bar{K}} e^D_\beta$.
- Assume that $(\phi, \oplus) = (\phi', \oplus') \wedge (\phi'', \oplus'')$. Let $e_{\phi',\oplus'}[\bar{K}, \bar{V}]$ and $e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$ be the expressions obtained for $(\phi', \oplus')$ and $(\phi'', \oplus'')$, respectively, by induction hypothesis. We construct the expression $e_{\phi,\oplus}$ by taking the join $e_{\phi',\oplus'}[\bar{K}, \bar{V}] \bowtie e'_{\phi'',\oplus''}[\bar{K}, \bar{V}']$, where

  $$e'_{\phi'',\oplus''}[\bar{K}, \bar{V}'] := \rho_{\bar{V} \to \bar{V}'} e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$$

is obtained by simply renaming $\bar{V}$ as $\bar{V}'$ in $e_{\phi'',\oplus''}$ (there is no need to specify an aggregate operator for $\bowtie$ as $\bar{V}$ and $\bar{V}'$ have no attributes in common), and apply $\pi_{\bar{V}}\mathsf{eq}_{\bar{V},\bar{V}'}$ over it. In fact, if a tuple $(\bar{k},\bar{v})$ is selected by this expression it means that $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'} \cap e^D_{\phi'',\oplus''}$ by definition of $\mathsf{eq}_{\bar{V},\bar{V}'}$. In turn, if $(\bar{k},\bar{v})$ is not selected by the expression it means either that there is no tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$, or the unique tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$ satisfies that $\bar{v} \neq \bar{w}$ (due to the way in which $\mathsf{eq}_{\bar{V},\bar{V}'}$ is defined). In any of the two cases we have that $(\bar{k},\bar{v}) \notin e^D_{\phi',\oplus'} \cap e^D_{\phi'',\oplus''}$.

- Assume that $(\phi,\oplus) = (\phi',\oplus') \vee (\phi'',\oplus'')$. Let $e_{\phi',\oplus'}[\bar{K},\bar{V}]$ and $e_{\phi'',\oplus''}[\bar{K},\bar{V}]$ be the expressions obtained for $(\phi',\oplus')$ and $(\phi'',\oplus'')$, respectively, by induction hypothesis. We construct the expression $e_{\phi,\oplus}$ by taking the union of the following expressions.
  - An expression e_1 such that, when evaluated on a LARA database D, it computes the tuples $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'}$ for which there is no tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$. This can be done in the same way as we constructed e_α for the case when $(\phi,\oplus) = (\phi',\oplus') \wedge \neg(\phi'',\oplus'')$ (see above).
  - Analogously, an expression e_2 that computes the tuples $(\bar{k},\bar{v}) \in e^D_{\phi'',\oplus''}$ for which there is no tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi',\oplus'}$.
  - An expression e_3 such that, when evaluated on a LARA database D, it computes the tuples $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'}$ for which there is a tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$ that satisfies $\bar{v} = \bar{w}$. This can be done in the same way as we did in the previous point.
  - An expression e_4 such that, when evaluated on a LARA database D, it computes the tuples $(\bar{k},\bar{v})$ that are of the form $(\bar{k},\bar{w}_1 \oplus \bar{w}_2)$ for $(\bar{k},\bar{w}_1) \in e^D_{\phi',\oplus'}$ and $(\bar{k},\bar{w}_2) \in e^D_{\phi'',\oplus''}$ with $\bar{w}_1 \neq \bar{w}_2$. The expression e_4 can be defined as $e_\alpha \mathbin{⧖} e_\beta$, where e^D_α contains all tuples $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'}$ such that there is a tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$ that satisfies $\bar{v} \neq \bar{w}$, and e^D_β contains all tuples $(\bar{k},\bar{v}) \in e^D_{\phi'',\oplus''}$ such that there is a tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi',\oplus'}$ that satisfies $\bar{v} \neq \bar{w}$. It is easy to see how to express e_α and e_β by using techniques similar to the ones developed in the previous points.
- Assume that $(\phi,\oplus) = (\phi',\oplus') \wedge k = \tau(\bar{x},\bar{i})$, for a formula $\phi'(\bar{x},\bar{i})$ and a value-term τ of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$, and k a value-variable not necessarily present in $\bar{i}$. We only consider the case when k is not in $\bar{i}$. The other case is similar. Before we proceed we prove the following lemma which is basic for the construction (the proof is omitted due to lack of space).

  ▶ **Lemma 5.** *For every pair $(\alpha,\oplus_\alpha)$, where $\alpha(\bar{x},\bar{i})$ is a formula of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ and $\oplus_\alpha$ is an aggregate operator over* Values, *and for every value-term $\lambda(\bar{x},\bar{i})$ of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$, there is an expression $e_{\alpha,\oplus_\alpha,\lambda}[\bar{K},\bar{V},V_1]$ of LARA(Ω) such that for every LARA database D:*

  $$(\bar{k},\bar{v},v_1) \in e^D_{\alpha,\oplus_\alpha,\lambda} \quad \Longleftrightarrow \quad \big((\bar{k},\bar{v}) \in \alpha^D_{\oplus_\alpha} \ \text{ and } \ \lambda(\bar{k},\bar{v}) = v_1\big).$$

  It should be clear then that $e_{\phi,\oplus} = e_{\phi',\oplus',\tau}[\bar{K},\bar{V},V_1]$, where $e_{\phi',\oplus',\tau}[\bar{K},\bar{V},V_1]$ is the expression constructed for $(\phi',\oplus')$ and τ by applying Lemma 5.
- Assume that $(\phi,\oplus) = (\phi',\oplus') \wedge R_f(\bar{x},\bar{x}',\bar{i},\bar{i}')$, where $\phi(\bar{x},\bar{i})$ is a formula of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$. Let $e_{\phi',\oplus'}[\bar{K},\bar{V}]$ be the expression obtained for $(\phi',\oplus')$ by induction hypothesis. We can then define the expression $e_{\phi,\oplus}$ as $\mathsf{Ext}_f(e_{\phi',\oplus'}[\bar{K},\bar{V}]) \bowtie e_{\phi',\oplus'}[\bar{K},\bar{V}]$, assuming that f is of sort $(\bar{K},\bar{V}) \to (\bar{K}',\bar{V}')$. There is no need to specify an aggregate operator for $\bowtie$ here, since by assumption we have that $\bar{V} \cap \bar{V}' = \emptyset$.
- The cases $(\phi,\oplus) = \exists x(\phi',\oplus')$ and $(\phi,\oplus) = \exists i(\phi',\oplus')$ can be translated as $\pi^{\oplus}_{\bar{K}}\, e_{\phi',\oplus'}[K,\bar{K},\bar{V}]$ and $\pi_{\bar{V}}\, e_{\phi',\oplus'}[\bar{K},\bar{V},V]$, respectively, assuming that $e_{\phi',\oplus'}$ is the expression obtained for $(\phi',\oplus')$ by induction hypothesis.

This finishes the proof of the theorem. ◀

Discussion

The results presented in this section imply that Lara has the same expressive power as $\mathrm{FO}_{\mathsf{Agg}}$, which in turn is tightly related to the expressiveness of SQL [13]. One might wonder then why to use Lara instead of SQL. While it is difficult to give a definite answer to this question, we would like to note that Lara is especially tailored to deal with ML objects, such as matrices or tensors, which are naturally modeled as associative tables. As the proof of Theorem 4 suggests, in turn, $\mathrm{FO}_{\mathsf{Agg}}$ requires of several cumbersome tricks to maintain the "key-functionality" of associative tables.

5 Expressiveness of LARA in terms of ML Operators

We assume in this section that $\mathsf{Values} = \mathbb{Q}$. Since extension functions in Ω can a priori be arbitrary, to understand what Lara can express we first need to specify which classes of functions are allowed in Ω. In rough terms, this is determined by the operations that one can perform when comparing keys and values, respectively. We explain this below.

- Extensions of two-sorted logics with aggregate operators over a *numerical* sort $\mathcal{N}$ often permit to perform arbitrary numerical comparisons over $\mathcal{N}$ (in our case $\mathcal{N} = \mathsf{Values} = \mathbb{Q}$). It has been noted that this extends the expressive power of the language, while at the same time preserving some properties of the logic that allow to carry out an analysis of its expressiveness based on well-established techniques (see, e.g., [14]).
- In some cases in which the expressive power of the language needs to be further extended, one can also define a linear order on the non-numerical sort (which in our case is the set Keys) and then perform suitable arithmetic comparisons in terms of such a linear order. A well-known application of this idea is in the area of descriptive complexity [11].

We start in this section by considering the first possibility only. That is, we allow comparing elements of $\mathsf{Values} = \mathbb{Q}$ in terms of arbitrary numerical operations. Elements of Keys, in turn, can only be compared with respect to equality. This yields a logic that is amenable for theoretical exploration – in particular, in terms of its expressive power – and that at the same time is able to express many extension functions of practical interest (e.g., several of the functions used in examples in [9, 10]).

We design a simple logic $\mathrm{FO}(=, \mathsf{All})$ for expressing extension functions. Intuitively, the name of this logic states that it can only compare keys with respect to equality but it can compare values in terms of arbitrary numerical predicates. The formulas in the logic are standard FO formulas where the only atomic expressions allowed are of the following form:

- $x = y$, for x, y key-variables;
- $P(i_1, \dots, i_k)$, for $P \subseteq \mathbb{Q}^k$ a numerical relation of arity k and $i_1, \dots, i_k$ value-variables or constants of the form $0_\oplus$.

The semantics of this logic is standard. In particular, an assignment η from value-variables to $\mathbb{Q}$ *satisfies* a formula of the form $P(i_1, \dots, i_k)$, for $P \subseteq \mathbb{Q}^k$, whenever $\eta(i_1, \dots, i_k) \in P$.

Let $\phi(\bar{x}, \bar{y}, \bar{i}, \bar{j})$ be a formula of $\mathrm{FO}(=, \mathsf{All})$. For a tuple $t = (\bar{k}, \bar{k}', \bar{v}, \bar{v}') \in \mathsf{Keys}^{|\bar{k}|+|\bar{k}'|} \times \mathsf{Values}^{|\bar{v}|+|\bar{v}'|}$ we abuse terminology and say that $\phi(\bar{k}, \bar{k}', \bar{v}, \bar{v}')$ *holds* if $D_t \models \phi(\bar{k}, \bar{k}', \bar{v}, \bar{v}')$, where D_t is the database composed exclusively by tuple t. In addition, an extension function f of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$ is *definable* in $\mathrm{FO}(=, \mathsf{All})$, if there is a formula $\phi_f(\bar{x}, \bar{y}, \bar{i}, \bar{j})$ of $\mathrm{FO}(=, \mathsf{All})$, for $|\bar{x}| = |\bar{K}|$, $|\bar{y}| = |\bar{K}'|$, $|\bar{i}| = |\bar{V}|$, and $|\bar{j}| = |\bar{V}'|$, such that for every tuple $(\bar{k}, \bar{v})$ of sort $(\bar{K}, \bar{V})$ it is the case

$$f(\bar{k}, \bar{v}) \;=\; \{(\bar{k}', \bar{v}') \mid \phi(\bar{k}, \bar{k}', \bar{v}, \bar{v}') \text{ holds}\}.$$

This gives rise to the definition of the following class of extension functions:

$$\Omega_{(=,\mathsf{All})} = \{f \mid f \text{ is an extension function that is definable in FO}(=,\mathsf{All})\}.$$

Recall that extension functions only produce finite associative tables by definition, and hence only some formulas in FO$(=,\mathsf{All})$ define extension functions.

The extension functions $\mathsf{copy}_{\bar{K},\bar{K}'}$, $\mathsf{copy}_{\bar{V},\bar{V}'}$, $\mathsf{add}_{V,0_\oplus}$, $\mathsf{eq}_{\bar{K},\bar{K}'}$, $\mathsf{eq}_{\bar{V},\bar{V}'}$, $\mathsf{neq}_{\bar{K},\bar{K}'}$, $\mathsf{neq}_{\bar{V},\bar{V}'}$, and $\pi_{\bar{V}}$, as shown in the previous section, are in $\Omega_{(=,\mathsf{All})}$. In turn, $\mathsf{copy}_{\bar{V},\bar{K}}$ and $\mathsf{copy}_{\bar{K},\bar{V}}$ are not, as FO$(=,\mathsf{All})$ cannot compare keys with values. Next we provide more examples.

▶ **Example 6.** We use $i + j = k$ and $ij = k$ as a shorthand notation for the ternary numerical predicates of addition and multiplication, respectively. Consider first a function f that takes a tuple t of sort (K_1, K_2, V) and computes a tuple t' of sort (K_1', K_2', V') such that $t(K_1, K_2) = t'(K_1', K_2')$ and $t'(V') = 1 - t(V)$. Then f is definable in FO$(=,\mathsf{All})$ as $\phi_f(x, y, x', y', i, j) := \big(x = x' \wedge y = y' \wedge i + j = 1\big)$. This function can be used, e.g., to interchange 0s and 1s in a Boolean matrix.

Consider now a function g that takes a tuple t of sort (K, V_1, V_2) and computes a tuple t' of sort (K', V') such that $t(K) = t'(K')$ and $t'(V)$ is the average between $t(V_1)$ and $t(V_2)$. Then g is definable in FO$(=,\mathsf{All})$ as $\phi_g(x, y, i_1, i_2, j) := \big(x = y \wedge \exists i\,(i_1 + i_2 = i \wedge 2j = i)\big)$. ◀

As an immediate corollary to Theorem 3 we obtain the following result, which formalizes the fact that – in the case when $\mathsf{Values} = \mathbb{Q}$ – for translating $\textsc{Lara}(\Omega_{(=,\mathsf{All})})$ expressions it is not necessary to extend the expressive power of $\mathrm{FO}_{\mathsf{Agg}}$ with the relations in $\Psi_{\Omega_{(=,\mathsf{All})}}$ as long as one has access to all numerical predicates over $\mathbb{Q}$. Formally, let us denote by $\mathrm{FO}_{\mathsf{Agg}}(\mathsf{All})$ the extension of $\mathrm{FO}_{\mathsf{Agg}}$ with all formulas of the form $P(\iota_1, \ldots, \iota_k)$, for $P \subseteq \mathbb{Q}^k$ and $\iota_1, \ldots, \iota_k$ value-terms, with the expected semantics. Then one can prove the following result.

▶ **Corollary 7.** *For every expression $e[\bar{K}, \bar{V}]$ of $\textsc{Lara}(\Omega_{(=,All)})$ there is a formula $\phi_e(\bar{x}, \bar{i})$ of $\mathrm{FO}_{\mathsf{Agg}}(All)$ such that $e^D = \phi_e^D$, for every* $\textsc{Lara}$ *database D.*

It is known that queries definable in $\mathrm{FO}_{\mathsf{Agg}}(\mathsf{All})$ satisfy two important properties, namely, *genericity* and *locality*, which allow us to prove that neither convolution of matrices nor matrix inversion can be defined in the language. From Corollary 7 we obtain then that none of these queries is expressible in $\textsc{Lara}(\Omega_{(=,\mathsf{All})})$. We explain this next.

Convolution

Let A be an arbitrary matrix and K a square matrix. For simplicity we assume that K is of odd size $(2n + 1) \times (2n + 1)$. The convolution of A and K, denoted by $A * K$, is a matrix of the same size as A whose entries are defined as

$$(A * K)_{k\ell} = \sum_{s=1}^{2n+1} \sum_{t=1}^{2n+1} A_{k-n+s,\ell-n+t} \cdot K_{st}. \tag{6}$$

Notice that $k - n + s$ and $\ell - n + t$ could be invalid indices for matrix A. The standard way of dealing with this issue is *zero padding*. This simply assumes those entries outside A to be 0. In the context of the convolution operator, one usually calls K a *kernel*.

We represent A and K over the schema $\sigma = \{\mathsf{Entry}_A[K_1, K_2, V], \mathsf{Entry}_K[K_1, K_2, V]\}$. Assume that $\mathsf{Keys} = \{\mathsf{k}_1, \mathsf{k}_2, \mathsf{k}_3, \ldots\}$ and $\mathsf{Values} = \mathbb{Q}$. If A is a matrix of values in $\mathbb{Q}$ of dimension $m \times p$, and K is a matrix of values in $\mathbb{Q}$ of dimensions $(2n + 1) \times (2n + 1)$ with $m, p, n \geq 1$, we represent the pair (A, K) as the $\textsc{Lara}$ database $D_{A,K}$ over σ that contains all facts $\mathsf{Entry}_A(\mathsf{k}_i, \mathsf{k}_j, A_{ij})$, for $i \in [m]$, $j \in [p]$, and all facts $\mathsf{Entry}_K(\mathsf{k}_i, \mathsf{k}_j, K_{ij})$, for $i \in [2n + 1]$,

$j \in [2n+1]$. The query Convolution over schema σ takes as input a LARA database of the form $D_{A,K}$ and returns as output an associative table of sort $[K_1, K_2, V]$ that contains exactly the tuples $(\mathsf{k}_i, \mathsf{k}_j, (A * K)_{ij})$. We can then prove the following result.

▶ **Proposition 8.** *Convolution is not expressible in* LARA$(\Omega_{(=,\mathit{All})})$.

The proof is based on a simple *genericity* property for the language that is not preserved by convolution.

Matrix inverse

It has been shown by Brijder et al. [1] that matrix inversion is not expressible in MATLANG by applying techniques based on locality. The basic idea is that MATLANG is subsumed by $\mathrm{FO}_{\mathsf{Agg}}(\emptyset) = \mathrm{FO}_{\mathsf{Agg}}$, and the latter logic can only define *local* properties. Intuitively, this means that formulas in $\mathrm{FO}_{\mathsf{Agg}}$ can only distinguish up to a *fixed-radius* neighborhood from its free variables (see, e.g., [14] for a formal definition). On the other hand, as shown in [1], if matrix inversion were expressible in MATLANG there would also be a $\mathrm{FO}_{\mathsf{Agg}}$ formula that defines the transitive closure of a binary relation (represented by its adjacency Boolean matrix). This is a contradiction as transitive closure is the prime example of a non-local property. We use the same kind of techniques to show that matrix inversion is not expressible in LARA$(\Omega_{(=,\mathsf{All})})$. For this, we use the fact that $\mathrm{FO}_{\mathsf{Agg}}(\mathsf{All})$ is also local.

We represent Boolean matrices as databases over the schema $\sigma = \{\mathsf{Entry}[K_1, K_2, V]\}$. Assume that $\mathsf{Keys} = \mathbb{N}$ and $\mathsf{Values} = \mathbb{Q}$. The Boolean matrix M of dimension $n \times m$, for $n, m \geq 1$, is represented as the LARA database D_M over σ that contains all facts $\mathsf{Entry}(i, j, b_{ij})$, for $i \in [n]$, $j \in [m]$, and $b_{ij} \in \{0, 1\}$, such that $M_{ij} = b_{ij}$. Consider the query Inv over schema σ that takes as input a LARA database of the form D_M and returns as output the LARA database $D_{M^{-1}}$, for M^{-1} the inverse of M. Then:

▶ **Proposition 9.** LARA$(\Omega_{(=,\mathit{All})})$ *cannot express* Inv *over Boolean matrices. That is, there is no* LARA$(\Omega_{(=,\mathit{All})})$ *expression* $e_{\mathsf{Inv}}[K_1, K_2, V]$*over* σ *such that* $e_{\mathsf{Inv}}(D_M) = \mathsf{Inv}(D_M)$*, for every* LARA *database of the form* D_M *that represents a Boolean matrix* M*.*

6 Adding Built-in Predicates over Keys

In Section 5 we have seen that there are important linear algebra operations, such as matrix inverse and convolution, that LARA$(\Omega_{(=,\mathsf{All})})$ cannot express. The following result shows, on the other hand, that a clean extension of LARA$(\Omega_{(=,\mathsf{All})})$ can express matrix convolution. This extension corresponds to the language LARA$(\Omega_{(<,\mathsf{All})})$, i.e., the extension of LARA$(\Omega_{(=,\mathsf{All})})$ in which we assume the existence of a strict linear-order $<$ on Keys and extension functions are definable in the logic FO$(<, \mathsf{All})$ that extends FO$(=, \mathsf{All})$ by allowing atomic formulas of the form $x < y$, for x, y key-variables. Even more, the only numerical predicates from All we need are $+$ and $\times$. We denote the resulting logic as LARA$(\Omega_{(<,\{+,\times\})})$.

▶ **Proposition 10.** CONVOLUTION *is expressible in* LARA$(\Omega_{(<,\{+,\times\})})$.

It is worth remarking that Hutchison et al. [9] showed that for every fixed kernel K, the query $(A * K)$ is expressible in LARA. However, the LARA expression they construct depends on the values of K, and hence their construction does not show that in general convolution is expressible in LARA. Our construction is stronger, as we show that there exists a *fixed* LARA$(\Omega_{(<,\{+,\times\})})$ expression that takes A and K as input and produces $(A * K)$ as output.

Current ML libraries usually have specific implementations for the convolution operator. Although specific implementations can lead to very efficient ways of implementing a single convolution, they could prevent the optimization of pipelines that merge several convolutions with other operators. Proposition 10 shows that convolution can be expressed in LARA just using general abstract operators such as aggregation and filtering. This could open the possibility for optimizing expressions that mix convolution and other operators.

Can LARA$(\Omega_{(<,\{+,\times\})})$ express inverse?

We believe that LARA$(\Omega_{(<,\{+,\times\})})$ cannot express INV. However, this seems quite challenging to prove. First, the tool we used for showing that INV is not expressible in LARA$(\Omega_{(=,\mathsf{All})})$, namely, locality, is no longer valid in this setting. In fact, queries expressible in LARA$(\Omega_{(<,\{+,\times\})})$ are not necessarily local.

▶ **Proposition 11.** *LARA$(\Omega_{(<,\{+,\times\})})$ can express non-local queries.*

This implies that one would have to apply techniques more specifically tailored for the logic, such as *Ehrenfeucht-Fraïssé* games, to show that INV is not expressible in LARA$(\Omega_{(<,\{+,\times\})})$. Unfortunately, it is often combinatorially difficult to apply such techniques in the presence of built-in predicates, e.g., a linear order, on the domain; cf., [5, 17, 7]. So far, we have not managed to succeed in this regard.

On the other hand, we can show that INV is not expressible in a natural restriction of LARA$(\Omega_{(<,\{+,\times\})})$ under complexity-theoretic assumptions. To start with, INV is complete for the complexity class DET, which contains all those problems that are logspace reducible to computing the *determinant* of a matrix. It is known that LOGSPACE $\subseteq$ DET, where LOGSPACE is the class of functions computable in logarithmic space, and this inclusion is believed to be proper [3].

In turn, most of the aggregate operators used in practical applications, including standard ones such as SUM, AVG, MIN, MAX, and COUNT, can be computed in LOGSPACE (see, e.g., [2]). Combining this with well-known results on the complexity of computing relational algebra and arithmetic operations, we obtain that the fragment LARA$_{\mathrm{st}}(\Omega_{(<,\{+,\times\})})$ of LARA$(\Omega_{(<,\{+,\times\})})$ that only mentions the standard aggregate operators above, and whose formulas defining extension functions are *safe*, can be evaluated in LOGSPACE in *data complexity*, i.e., assuming formulas to be fixed.

▶ **Proposition 12.** *Let $e[\bar{K},\bar{V}]$ be a fixed expression of LARA$_{\mathrm{st}}(\Omega_{(<,\{+,\times\})})$. There is a LOGSPACE procedure that takes as input a LARA database D and computes e^D.*

Hence, proving INV to be expressible in the language LARA$_{\mathrm{st}}(\Omega_{(<,\{+,\times\})})$ would imply the surprising result that LOGSPACE = DET.

7 Final Remarks and Future Work

We believe that the work on query languages for analytics systems that integrate relational and statistical functionalities provides interesting perspectives for database theory. In this paper we focused on the LARA language, which has been designed to become the core algebraic language for such systems. As we have observed, expressing interesting ML operators in LARA requires the addition of complex features, such as arithmetic predicates on the numerical sort and built-in predicates on the domain. The presence of such features complicates the study of the expressive power of the languages, as some known techniques no longer hold, e.g., genericity and locality, while others become combinatorially difficult to apply, e.g., Ehrenfeucht-Fraïssé games. In addition, the presence of a built-in linear order might turn the logic capable of characterizing some parallel complexity classes, and thus inexpressibility results could be as hard to prove as some longstanding conjectures in complexity theory.

A possible way to overcome these problems might be not looking at languages in its full generality, but only at extensions of the tame fragment LARA($\Omega_{(=,\mathsf{All})}$) with some of the most sophisticated operators. For instance, what if we extend LARA($\Omega_{(=,\mathsf{All})}$) directly with an operator that computes Convolution? Is it possible to prove that the resulting language (LARA($\Omega_{(=,\mathsf{All})}$) + Convolution) cannot express matrix inverse Inv? Somewhat a similar approach has been followed in the study of MATLANG; e.g., [1] studies the language (MATLANG + INV), which extends MATLANG with the matrix inverse operator.

Another interesting line of work corresponds to identifying which kind of operations need to be added to LARA in order to be able to express in a natural way recursive operations such as matrix inverse. One would like to do this in a general yet minimalistic way, as adding too much recursive expressive power to the language might render it impractical. It would be important to start then by identifying the most important recursive operations one needs to perform on associative tables, and then abstract from them the minimal primitives that the language needs to possess for expressing such operations.

References

1 Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. On the Expressive Power of Query Languages for Matrices. In *ICDT*, pages 10:1–10:17, 2018.

2 Mariano P. Consens and Alberto O. Mendelzon. Low Complexity Aggregation in GraphLog and Datalog. *Theor. Comput. Sci.*, 116(1):95–116, 1993.

3 Stephen A. Cook. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control*, 64(1-3):2–21, 1985.

4 Serge Abiteboul et al. Research Directions for Principles of Data Management (Dagstuhl Perspectives Workshop 16151). *Dagstuhl Manifestos*, 7(1):1–29, 2018.

5 Ronald Fagin, Larry J. Stockmeyer, and Moshe Y. Vardi. On Monadic NP vs. Monadic co-NP. *Inf. Comput.*, 120(1):78–92, 1995.

6 Floris Geerts. On the Expressive Power of Linear Algebra on Graphs. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 7:1–7:19, 2019.

7 Martin Grohe and Thomas Schwentick. Locality of Order-Invariant First-Order Formulas. In *MFCS*, pages 437–445, 1998.

8 Stephan Hoyer, Joe Hamman, and xarray developers. xarray Development roadmap, 2018. Available at `http://xarray.pydata.org/en/stable/roadmap.html`, retrieved on March 2019.

9 Dylan Hutchison, Bill Howe, and Dan Suciu. Lara: A Key-Value Algebra underlying Arrays and Relations. *CoRR*, abs/1604.03607, 2016.

10 Dylan Hutchison, Bill Howe, and Dan Suciu. LaraDB: A Minimalist Kernel for Linear and Relational Algebra Computation. In *Proceedings of BeyondMR@SIGMOD 2017*, pages 2:1–2:10, 2017.

11 Neil Immerman. *Descriptive complexity.* Graduate texts in computer science. Springer, 1999.

12 Andreas Kunft, Alexander Alexandrov, Asterios Katsifodimos, and Volker Markl. Bridging the gap: towards optimization across linear and relational algebra. In *Proceedings of BeyondMR@SIGMOD 2016*, page 1, 2016.

13 Leonid Libkin. Expressive power of SQL. *Theor. Comput. Sci.*, 296(3):379–404, 2003.

14 Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

15 Alexander M. Rush. Tensor Considered Harmful. Technical report, Harvard NLP Blog, 2019. Available at `http://nlp.seas.harvard.edu/NamedTensor`, retrieved on March 2019.

16 Alexander M. Rush. Tensor Considered Harmful Pt. 2. Technical report, Harvard NLP Blog, 2019. Available at `http://nlp.seas.harvard.edu/NamedTensor2`, retrieved on March 2019.

17 Thomas Schwentick. On Winning Ehrenfeucht Games and Monadic NP. *Ann. Pure Appl. Logic*, 79(1):61–92, 1996.

Random Sampling and Size Estimation Over Cyclic Joins

Yu Chen
Hong Kong University of Science and Technology, Hong Kong
ychenbh@cse.ust.hk

Ke Yi
Hong Kong University of Science and Technology, Hong Kong
yike@cse.ust.hk

Abstract

Computing joins is expensive, and often unnecessary when the output size is large. In 1999, Chaudhuri et al. [7] posed the problem of random sampling over joins as a potentially effective approach to avoiding computing the join in full, while obtaining important statistical information about the join results. Unfortunately, no significant progress has been made in the last 20 years, except for the case of acyclic joins. In this paper, we present the first non-trivial result on sampling over cyclic joins. We show that after a linear-time preprocessing step, a join result can be drawn uniformly at random in expected time $O(\mathrm{IN}^\rho/\mathrm{OUT})$, where IN^ρ is known as the AGM bound of the join and OUT is its output size. This result holds for all joins on binary relations, as well as certain joins on relations of higher arity. We further show how this algorithm immediately leads to a join size estimation algorithm with the same running time.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Random sampling, joins, join size estimation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.7

Funding This work has been supported by HKRGC under grants 16202317, 16201318, and 16201819.

1 Introduction

A join query can be modeled as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, with $|\mathcal{V}| = n$ vertices and $m = |\mathcal{E}|$ hyperedges. Each vertex models an attribute, and for each hyperedge $F \in \mathcal{E}$, there is a relation R_F on attribute set F. We use $Q := \bowtie_{F\in\mathcal{E}} R_F$ to denote the set of join results. In this paper, we consider the data complexity of query evaluation, i.e., the running time of the algorithms will be measured by the total input size $\mathrm{IN} = \sum_{F\in\mathcal{E}} |R_F|$ and the output size $\mathrm{OUT} = |Q|$, while assuming n and m are constants.

Algorithms for computing Q have been extensively studied. It is well known that Q can be computed in $O(\mathrm{IN}^\rho)$ time, where ρ is the optimal solution of the following linear program [3, 13, 14, 16]:

$$\begin{aligned} &\min_{x_F, F\in\mathcal{E}} \log_{\mathrm{IN}} |R_F| \cdot x_F \\ \text{s.t. } &\sum_{F, v\in F} x_F \geq 1, \text{ for every } v \in \mathcal{V}, \\ &0 \leq x_F \leq 1, \text{ for every } F \in \mathcal{E}. \end{aligned} \tag{1}$$

This is worst-case optimal since OUT can be as large as $\Theta(\mathrm{IN}^\rho)$ on some instances. Alternatively, output-sensitive algorithms are known that compute Q in $O(\mathrm{IN}^w + \mathrm{OUT})$ time, where w is certain notion of *width* of the hypergraph $\mathcal{H}$ [10, 11]. But this is still very expensive for highly cyclic queries, for which w is close to ρ, or when OUT is large. Indeed, computing multi-way joins is still the bottleneck in query evaluation in modern database systems.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 7; pp. 7:1–7:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One key observation made in as early as 1999 [7] is that rarely are full join results required by the user. In almost all use cases, the join results are aggregated and presented to the user in a succinct form. The aggregation function can be as simple as a count or a sum, a random sample, or an arbitrary UDF. Thus, the intriguing question is, can we compute the final query result without computing the join in full? To make the problem well defined and amenable to theoretical investigation, in this paper we consider two particular aggregation functions: random sampling and (approximate) counting. These are arguably the two most basic aggregates, which other more complicated aggregations can be based upon. For example, sum can be considered as a weighted version of the count, while many UDFs (e..g, median and quantiles) can be computed approximated from a random sample in lieu of full data.

1.1 Previous results on random sampling over joins

The starting observation from the pioneering work of Chaudhuri et al. [7] is that the sampling operator cannot be pushed down through a join operator, i.e, sample(R_1) $\bowtie$ sample(R_2) $\neq$ sample($R_1 \bowtie R_2$). To see this, consider a binary join $R_1(A,B) \bowtie R_2(B,C)$, with $R_1 = \{(a_1,b_1),(a_1,b_2),\ldots,(a_N,b_2)\}$ and $R_2 = \{(b_1,c_1),(b_1,c_2),\ldots,(b_1,c_N),(b_2,c_1)\}$. Note that the join size is $\mathrm{OUT} = 2N$. Thus, to be able to sample a join result uniformly at random from these $2N$ join results, one has to *non-uniformly* sample tuples from either R_1 or R_2. In particular, the (a_1,b_1) in R_1 must be sampled with a probability that is N times larger than the other tuples in R_1, because it joins with N tuples in R_2. More formally, Chaudhuri et al. [7] prove that, without precomputing some auxiliary information from the data, drawing a sample from $R_1 \bowtie R_2$ requires at least $\Omega(\mathrm{IN})$ time. In view of this negative result, they first collect the frequency information from R_2, i.e., $|\sigma_{B=b}R_2|$ for each distinct b, and build a weighted sampling data structure on R_1 using these frequencies as weights. One can use the "alias method" [6] to build a weighted sampling structure in linear time, which supports drawing a weighted sample in $O(1)$ time. After drawing a weighted sample t_1 from R_1, we randomly draw a tuple t_2 from $R_2 \ltimes t_1$[1], which can be done in constant time if there is an index on R_2. Note that an index can also be built in linear time. Therefore, the formal result of Chaudhuri et al. [7] is that a data structure can be built in linear time, which allows one to draw a random sample from $R_1 \bowtie R_2$ in $O(1)$ time.

At the same SIGMOD conference with Chaudhuri et al. [7], Acharya et al. [1] studied the problem of random sampling over multi-way joins. However, their algorithm only works for a very special type of joins with foreign-key constraints, which imply that there is a one-to-one correspondence between the join results and tuples in the largest table. Thus, random sampling from the join reduces to sampling from a single table, which is trivial.

This problem then stayed dormant for almost 20 years, until Zhao et al. [19] designed an algorithm to sample from multi-way acyclic joins. They show that, on any acyclic join, a data structure can be build in linear time that allows one to draw a random sample from the join results in constant time. Their observation is that, on a multi-way join Q, one should sample a tuple t with probability proportional to $|Q_t|$, where $Q_t = \bowtie_{F\in\mathcal{E}} (R_F \ltimes t)$ is the *residual query* of t. Note that on the binary join $R_1(A,B) \bowtie R_2(B,C)$, the size of the residual query of some tuple $t = (a,b) \in R_1$ is exactly the frequency of b in R_2. Then, the idea is to do so recursively, with the algorithm of Chaudhuri et al. [7] becoming the base case. Finally, they make use of the (probably folklore) result that, for an acyclic join [18], all the residual query

[1] We use $R \ltimes t$ as an abbreviation for $R \ltimes \{t\}$.

sizes can be computed in $O(\mathrm{IN})$ time. Then they organize these residual query sizes in an appropriate data structure so that a sample can be drawn in $O(1)$ time. They also adapt their algorithm to cyclic queries, but there is no formal guarantee on its performance.

1.2 Previous results on join size estimation

For an acyclic query Q, it is well known that $\mathrm{OUT} = |Q|$ can be computed in $O(\mathrm{IN})$ time. But the problem looks very difficult for cyclic queries. Even for the simplest cyclic query, the triangle query $Q_\triangle = R_{\{A,B\}} \bowtie R_{\{B,C\}} \bowtie R_{\{A,C\}}$, we currently do not have any algorithm that can compute OUT faster than $O(\mathrm{IN}^\rho)$ time[2], i.e., counting seems to be as difficult as computing the full join results. On general cyclic queries, the $O(\mathrm{IN}^w + \mathrm{OUT})$-time output-sensitive join algorithm above can be modified so as to compute OUT in $O(\mathrm{IN}^w)$ time, but this is still very expensive for highly cyclic queries.

In most applications, we do not need an accurate OUT, while a reasonable estimate would be good enough. In the database literature, this is known as the "query size estimation" problem, and has been extensively studied. However, most results in this area are heuristics without formal guarantees on the accuracy of the estimate.

The problem of estimating $|Q_\triangle|$, i.e., counting triangles in a graph, has received particular attention. A commonly used model in the algorithms community is the *property testing* model, which assumes that the graph has already been preprocessed into some standard graph data structure (e.g., adjacent lists with hash tables), so that one can perform the following operations in constant time: randomly sampling a vertex, randomly sampling an edge, returning the degree of a vertex, returning the i-th neighbor of a vertex, and testing if an edge exist between two vertices. In this model, Eden et al. [8] designed an $\tilde{O}\left(\frac{\mathrm{IN}^{3/2}}{\mathrm{OUT}}\right)$-time[3] algorithm that returns a constant-factor approximation of OUT with constant probability, and showed that this is optimal. Their algorithm has been later extended to counting length-k cycles and size-k cliques for any constant k, and the running time becomes $\tilde{O}\left(\frac{\mathrm{IN}^{k/2}}{\mathrm{OUT}}\right)$ [4, 9]. Very recently, Assadi et al. [2] extended this algorithm to computing a constant-factor approximation of the join size of any query on binary relations in time of $\tilde{O}\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$. They also proved a matching lower bound, which actually holds for the problem of distinguishing between an input with no join result and one with OUT join results. So it holds for both the sampling problem and the join size estimation problem. However, their algorithm only works for the join size estimation problem, not sampling.

Unfortunately, the aforementioned algorithms perform very badly in practice, despite their theoretical optimality. For the triangle counting problem, one of the most practically efficient algorithms is *wedge sampling* [15], which departs from the property testing model slightly. A *wedge* is just a length-2 path. The basic idea of wedge sampling is to first uniformly sample a wedge, and then check if the wedge is closed, i.e., forms a triangle. The standard property testing model does not allow one to uniformly sample a wedge in constant time, but this can be easily supported by building a weighted sampling structure on all the vertices, where the weight of each vertex v is $d(v)(d(v)-1)/2$, which is exactly the number of wedges centered at v. This weighted sampling data structure can be built cheaply in a linear-time

[2] On the triangle query, the optimal solution to the linear program (1) is either $x_1 = x_2 = x_3 = 1/2$, or $x_1 = x_2 = 1, x_3 = 0$ (ignoring the other two symmetric cases). In the former case, $O(\mathrm{IN}^\rho) = O(\mathrm{IN}^{3/2})$; in the latter case, $O(\mathrm{IN}^\rho) = O(|R_1| \cdot |R_2|)$.

[3] The bound stated in [8] has an extra term, since they did not use edge sampling, which is sometimes not included in the standard property testing model. When edge sampling is allowed, the bound simplifies to $\tilde{O}\left(\frac{\mathrm{IN}^{3/2}}{\mathrm{OUT}}\right)$.

preprocessing step. Extending the idea of wedge sampling, MOSS-5 [17] is an algorithm that can count any pattern with up to 5 vertices. Instead of sampling wedges, MOSS-5 samples spanning trees of up to 5 vertices, and then checks if the other required edges are present. The same algorithm also applies to the join size estimation problem on joins over binary relations involving up to 5 attributes, but it does not have any theoretical guarantees. Wander Join [12] is an effective approach to answering join-aggregate queries approximately, with join size estimation as a special case. It returns non-uniform samples from the join results, and de-biases them using the Horvitz-Thompson estimator. The estimator is unbiased, but there is no guarantee on its error.

1.3 Our results

In this paper, we present the first nontrivial algorithm for random sampling over arbitrary cyclic queries. More formally, we show how to construct a data structure in linear time (the size of the data structure is thus necessarily no more than linear), so that a sample can be drawn uniformly at random from the join results in $O\left(\frac{\mathrm{IN}^{\rho}}{\mathrm{OUT}}\right)$ time in expectation, for the class of *sequenceable* queries. The precise definition of sequenceable queries is a bit technical; please see Section 2.5 for details. They include all queries on binary relations, as well as certain queries on relations of higher arity. For non-sequenceable queries, the running time for drawing a sample is $O\left(\frac{\mathrm{IN}^{\rho+1}}{\mathrm{OUT}}\right)$. These results hold for both full join queries and join-project queries. Prior to this work, the only solution to this problem with formal guarantees is to either precompute the full join results, which has $O(\mathrm{IN}^{\rho})$ preprocessing and storage cost[4], and $O(1)$ sampling cost, or compute the full join at sampling time, which has no preprocessing cost but $O(\mathrm{IN}^{\rho})$ sampling cost.

We also adapt our algorithm to solve the join size estimation problem. We show that after drawing a constant number of samples, a constant-factor approximation to the join size can be obtained with constant probability. This matches the recent result of Assadi et al. [2] on joins over binary relations[5]. Compared with [2], our result is different in the following aspects: (1) We have to build some additional data structures (still in linear time), while only standard graph representations are needed in [2], so their result in stronger in this respect. (2) Our algorithm supports random sampling from the join results, with join size estimation as a simple corollary, while [2] does not support random sampling. (3) Our algorithm supports certain queries on relations of higher arity, while [2] only supports binary relations. (4) The algorithm of [2] has some hidden logarithmic factors, while ours does not. (5) Unlike the algorithm [2] which is of only theoretical interest, our algorithm is actually very practical. We conducted some experiments in Section A, showing that our algorithm is competitive with the best known heuristics on the join size estimation problem.

The $O\left(\frac{\mathrm{IN}^{\rho}}{\mathrm{OUT}}\right)$ bound may not look attractive when OUT is small. In particular, if $\mathrm{OUT} = O(1)$, our algorithm is no better than computing the join results in full. However, improving this can be very difficult. Note that when $\mathrm{OUT} = O(1)$, random sampling from the join results is the same as finding all the results. Even for the triangle query, the best algorithm to date still takes $O(\mathrm{IN}^{1.408})$ time when $\mathrm{OUT} = O(1)$ [5], using the highly impractical fast matrix multiplication algorithm. This is only slightly better than $O(\mathrm{IN}^{\rho}) = O(\mathrm{IN}^{1.5})$.

[4] By combining the acyclic join sampling algorithm [19] with the generalized tree decomposition framework [10], the preprocessing cost can be driven down to $O(\mathrm{IN}^{\mathrm{fhw}})$, where fhw is the *fractional hypertree width* of the query, but $\rho = \mathrm{fhw}$ for highly cyclic queries.

[5] In fact, our work was done independent of [2], which we just came to know before submitting this paper. Also, our approach is completely different from [2].

2 Random Sampling over Cyclic Queries

2.1 Overview of approach

Order the vertices (attributes) in $\mathcal{V}$ arbitrarily as $v_1, v_2, \ldots, v_n$. Let $\mathrm{dom}(v_i)$ be the domain of attribute v_i. For any $X \subseteq \mathcal{V}$, let $\mathcal{E}_X = \{F \in \mathcal{E} \mid F \cap X \neq \emptyset\}$. We use I to denote the attribute set $\{v_1, v_2, \ldots, v_i\}$ and J denotes $\{v_{i+1}, \ldots, v_n\}$, for $i = 1, \ldots, n$. Define $I = \emptyset$ when $i = 0$. For a tuple t on attributes I, define the residual query on t as:

$$Q_t = \bowtie_{F \in \mathcal{E}_J} \pi_J(R_F \ltimes t). \tag{2}$$

Our starting point is Generic-Join [14], an elegant worst-case optimal join algorithm. A particular version of the algorithm is shown in Algorithm 1 . In the algorithm description, $\langle\rangle$ denotes the empty tuple; (t, y) denotes the concatenation of t and y, where t is a tuple on attributes I, and $y \in \mathrm{dom}(v_{i+1})$.

Algorithm 1 Generic-Join(i, t):

```
// $t$ is a tuple on $I = \{v_1, \ldots, v_i\}$ and $t \in \pi_I R_F$ for all $F \in \mathcal{E}_I$
if $i = n$ then return $\{\langle\rangle\}$
$Q_t \leftarrow \emptyset$
$L_t \leftarrow \bigcap_{F \in \mathcal{E}_{\{v_{i+1}\}}} \pi_{v_{i+1}}(R_F \ltimes t)$
foreach $y \in L_t$ do
    $Q_{(t,y)} \leftarrow$ Generic-Join$(i+1, (t, y))$
    $Q_t \leftarrow Q_t \cup \{y\} \times Q_{(t,y)}$
return $Q_t$
```

Let $(x_F, F \in \mathcal{E})$ be the optimal solution to linear program (1). It is known that the total running time of the Generic-Join algorithm is bounded by the AGM bound [3] of the query:

$$\mathsf{AGM}(Q) = \prod_{F \in \mathcal{E}} |R_F|^{x_F}.$$

Furthermore, the time spent on each recursive call Generic-Join(i, t) is bounded by the AGM bound on the residual query Q_t (define $0^0 = 0$):

$$\mathsf{AGM}(Q_t) = \prod_{F \in \mathcal{E}_J} |\pi_J(R_F \ltimes t)|^{x_F} = \prod_{F \in \mathcal{E}_J} |R_F \ltimes t|^{x_F}. \tag{3}$$

Unfolding the recursion, the execution process of the Generic-Join algorithm forms a tree $\mathcal{T}$. The root node of $\mathcal{T}$ corresponds to the initial call Generic-Join$(0, \langle\rangle)$; every node on the i-th level of $\mathcal{T}$ corresponds to a tuple t on attribute $I = (v_1, \ldots, v_i)$; a leaf node on level n corresponds to a join result. This tree has exactly OUT leaves. Below, we will not differentiate between a node in the tree and its corresponding tuple t.

If we know the residual query size $|Q_t|$ for every t, then this algorithm immediately yields a random sampling algorithm: At each node t of $\mathcal{T}$, instead of exploring all its children (t, y) for $y \in L_t$, we just sample one of them with probability proportional to its subtree size, i.e., sample (t, y) with probability $|Q_{(t,y)}|/|Q_t|$. This way, we will reach every leaf with equal probability. In fact, this is exactly the basic idea of the random sampling algorithm over acyclic queries [19].

For cyclic queries, unfortunately, there is no efficient way to compute all the residual query sizes $|Q_t|$. Our idea is to assume that each node t had a subtree size of $\mathsf{AGM}(Q_t)$, and perform the sampling using these subtree size upper bounds. This can be equivalently viewed as adding "rejection nodes" at various places of $\mathcal{T}$, such that each node t has exactly $\mathsf{AGM}(Q_t)$ leaves below, which include $|Q_t|$ "accept nodes", which correspond to true join results, and $\mathsf{AGM}(Q_t) - |Q_t|$ rejection nodes, which correspond to failed sampling paths.

More precisely, to sample a join result, we start at the root of $\mathcal{T}$. At each node t, we randomly sample a child $y \in L_t$ with probability $\mathsf{AGM}(Q_{(t,y)})/\mathsf{AGM}(Q_t)$, and reject with probability $1 - \sum_{y \in L_t}(\mathsf{AGM}(Q_{(t,y)})/\mathsf{AGM}(Q_t))$. Note that the sum of the sampling probabilities of all children $y \in L_t$ will not exceed 1, due to the query decomposition lemma [14], which states that

$$\sum_{y \in L_t} \mathsf{AGM}(Q_{(t,y)}) \leq \mathsf{AGM}(Q_t). \tag{4}$$

Finally, the probability to reach any leaf $t = (y_1, y_2, \dots, y_n) \in Q$ is

$$\frac{\mathsf{AGM}(Q_{(y_1)})}{\mathsf{AGM}(Q)} \cdot \frac{\mathsf{AGM}(Q_{(y_1,y_2)})}{\mathsf{AGM}(Q_{(y_1)})} \cdot \dots \cdot \frac{\mathsf{AGM}(Q_{(y_1,\dots,y_n)})}{\mathsf{AGM}(Q_{(y_1,\dots,y_{n-1})})} = \frac{1}{\mathsf{AGM}(Q)},$$

i.e., all join results are uniformly sampled. The probability to successfully reach a leaf node is $\frac{\mathrm{OUT}}{\mathsf{AGM}(Q)}$, so it takes $O\left(\frac{\mathsf{AGM}(Q)}{\mathrm{OUT}}\right) = O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$ attempts in expectation to draw a sample, as desired. Note that when Generic-Join is used as a sampling algorithm, there is no need to materialize the whole tree $\mathcal{T}$; only one root-to-leaf path needs to be explored.

However, to achieve a running time of $O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$, we only have constant time for each attempt. This means that we need to perform the sampling of a child $y \in L_t$ in constant time, which poses the two main technical difficulties that we must resolve in the rest of the paper: (1) How to avoid computing L_t, which would take super-constant time, and (2) how to build appropriate weighted sampling data structures so as to sample a y with probability $\mathsf{AGM}(Q_{(t,y)})/\mathsf{AGM}(Q_t)$ in constant time. We resolve these two difficulties in the rest of this section.

2.2 Avoid computing L_t

First, we order the attributes as $\mathcal{V} = \{v_1, \dots, v_n\}$ in a way such that for any $2 \leq j \leq n$ there is an $1 \leq i \leq j-1$ with $\{v_i, v_j\} \subseteq F$ for some $F \in \mathcal{E}$. Here, we assume that the query is connected; otherwise, we can just sample a result from each connected component $Q_1, \dots, Q_k$ and return their concatenation. The sampling time would be

$$\frac{\mathsf{AGM}(Q_1)}{|Q_1|} + \dots + \frac{\mathsf{AGM}(Q_k)}{|Q_k|} \leq \frac{\mathsf{AGM}(Q_1)}{|Q_1|} \cdot \dots \cdot \frac{\mathsf{AGM}(Q_k)}{|Q_k|} = \frac{\mathsf{AGM}(Q)}{\mathrm{OUT}}.$$

We will also assume that there are no relations of arity 1. Such relations can be easily removed in a preprocessing step: Suppose there is an arity-1 relation $R_{\{v_i\}}$. We just replace every other relation R_F with $R_F \ltimes R_{\{v_i\}}$, and then we remove $R_{\{v_i\}}$.

Suppose t is a tuple on $I = \{v_1, \dots, v_i\}$. Computing L_t requires computing the set intersection of $\pi_{v_{i+1}}(R_F \ltimes t)$ for $F \in \mathcal{E}_{\{v_{i+1}\}}$, which we cannot afford. Instead, we take one of these sets, chosen as follows:

$$F_t^* = \begin{cases} \arg\min_{F \in \mathcal{E}_{\{v_1\}}} |R_F|, & \text{if } i = 0; \\ \arg\min_{F \in \mathcal{E}_{\{v_{i+1}\}} \cap \mathcal{E}_I} |R_F \ltimes t|, & \text{if } i \geq 1. \end{cases} \tag{5}$$

Then we use

$$L'_t = \pi_{v_{i+1}}(R_{F^*_t} \ltimes t) \tag{6}$$

instead of L_t as the children of t in $\mathcal{T}$. Note that we always have $L'_t \supseteq L_t$.

Because every L'_t depends on only one particular relation, they are readily available from standard index structures (e.g., hash tables). More precisely, we build an index on each relation R_F, such that for any tuple t, the index returns the list $\pi_v(R_F \ltimes t)$, as well as its size, for any $v \in F$. This list is also known as the *neighbor list* of t in F on v, and its size the *degree* of $\pi_F t$ in F on v. In fact, the Generic-Join algorithm requires exactly the same index. Using these indexes, L'_t can be found in $O(1)$ time for any t on $I = \{v_1, \ldots, v_i\}$. For $i = 0$, F^*_t is a fixed relation, and L'_t is simply $\pi_{v_1} R_{F^*_t}$. For $i \geq 1$, F^*_t is one of the relations in $\mathcal{E}_{\{v_{i+1}\}}$. For any $F \in \mathcal{E}_{\{v_{i+1}\}}$, the neighbor list $\pi_{v_{i+1}}(R_F \ltimes t)$ is available from the index. By comparing their sizes, we can determine F^*_t and L'_t in $O(1)$ time. Finally, it is easy to see that the total size of all the neighbor lists is linear.

2.3 The sampling algorithm

Replacing L_t with L'_t, together with the discussion in Section 2.1, we obtain the Generic-Join-Sample algorithm, as shown in Algorithm 2. In addition, we need to check the validity of t in line 2. This is because we now sample from L'_t, which is a superset of L_t, so the parent call cannot guarantee its validity as in Algorithm 1.

Algorithm 2 Generic-Join-Sample(i, t):

```
1 // $t$ is a tuple on $I = \{v_1, \ldots, v_i\}$
2 if $t \notin \pi_I R_F$ for any $F \in \mathcal{E}_I$ then reject
3 if $i = n$ then return $t$
4 Set $F^*_t$ and $L'_t$ as in (5) and (6)
5 $q_t \leftarrow \sum_{y \in L'_t} \mathsf{AGM}(Q_{(t,y)})$
6 $y \leftarrow$ a random sample from $L'_t$ with probability $\mathsf{AGM}(Q_{(t,y)})/q_t$
7 With probability $q_t/\mathsf{AGM}(Q_t)$ return Generic-Join-Sample$(i+1, (t, y))$
8 else reject
```

Note that $\mathsf{AGM}(Q_t)$ is never zero in line 7. This is because in line 2, we reject t if $|R_F \ltimes t| = 0$ for any $F \in \mathcal{E}_I$, which implies that $|R_F \ltimes t| > 0$ for any $F \in \mathcal{E}_J$ (assuming the input relations are all nonempty).

Except line 5–6, all other operations in Algorithm 2 take $O(1)$ time using the indexes. Before describing how to implement line 5–6 efficiently, we first see an example.

An example

We illustrate the Generic-Join-Sample algorithm on the query $R_{\{A,B\}} \bowtie R_{\{B,C\}} \bowtie R_{\{A,C\}} \bowtie R_{\{C,D\}} \bowtie R_{\{C,E\}} \bowtie R_{\{D,E\}}$, with the database instance shown in Figure 1. For simplicity, we will write e.g. $\{A, B\}$ as AB, then $R_{\{A,B\}}$ is written as R_{AB}. Similarly, we write $x_{AB}, x_{BC}, \ldots$ as the optimal solution to (1). Suppose we order the attributes as A, B, C, D, E. We start the algorithm by calling Generic-Join-Sample$(0, \langle\rangle)$. When $i = 0$, F^*_t is always AB, since $|\pi_{v_1} R_{AB}| = 2$ and $|\pi_{v_1} R_{AC}| = 3$. So $L'_t = \{a_1, a_2\}$. We sample each $y \in L'_t$ with probability

$$\begin{aligned}\frac{\mathsf{AGM}(Q_y)}{\mathsf{AGM}(Q)} &= \frac{|R_{AB} \ltimes y|^{x_{AB}} |R_{AC} \ltimes y|^{x_{AC}} |R_{BC}|^{x_{BC}} |R_{CD}|^{x_{CD}} |R_{CE}|^{x_{CE}} |R_{DE}|^{x_{DE}}}{|R_{AB}|^{x_{AB}} |R_{AC}|^{x_{AC}} |R_{BC}|^{x_{BC}} |R_{CD}|^{x_{CD}} |R_{CE}|^{x_{CE}} |R_{DE}|^{x_{DE}}} \\ &= \frac{|R_{AB} \ltimes y|^{x_{AB}} |R_{AC} \ltimes y|^{x_{AC}}}{|R_{AB}|^{x_{AB}} |R_{AC}|^{x_{AC}}}.\end{aligned}$$

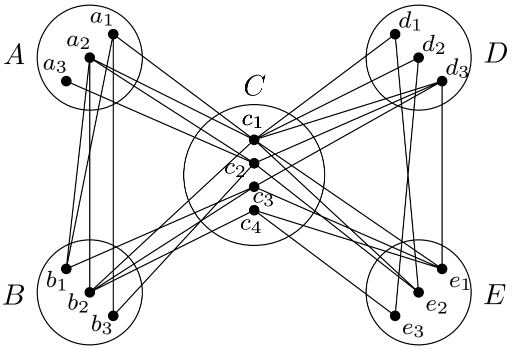


Figure 1 A database instance: The big circles represent attributes, the vertices inside a circle represent values in the domain of that attribute. The edges between two attributes, say, A, B represent tuples in the relation R_{AB}.

So a_1 and a_2 are sampled with probabilities $(\frac{2}{4})^{x_{AB}}(\frac{1}{4})^{x_{AC}}$ and $(\frac{2}{4})^{x_{AB}}(\frac{2}{4})^{x_{AC}}$, respectively, and reject otherwise. Suppose a_2 is sampled. Then we call Generic-Join-Sample$(1, \langle a_2 \rangle)$. When $i = 1$, F_t^* can only be AB and $L'_t = \{b_1, b_2\}$. Canceling the common terms as above, we sample each $y \in L'_t$ with probability

$$\frac{\mathsf{AGM}(Q_{(t,y)})}{\mathsf{AGM}(Q_t)} = \frac{|R_{BC} \ltimes (t,y)|^{x_{BC}}}{|R_{AB} \ltimes t|^{x_{AB}}|R_{BC}|^{x_{BC}}} = \frac{|R_{BC} \ltimes y|^{x_{BC}}}{|R_{AB} \ltimes t|^{x_{AB}}|R_{BC}|^{x_{BC}}}.$$

So b_1 and b_2 are sampled with probabilities $(\frac{1}{2})^{x_{AB}}(\frac{1}{5})^{x_{BC}}$ and $(\frac{1}{2})^{x_{AB}}(\frac{3}{5})^{x_{BC}}$, respectively. Suppose b_2 is sampled.

Then we call Generic-Join-Sample$(2, \langle a_2, b_2 \rangle)$, which is the most interesting step. When $i = 2$, F_t^* is either AC or BC, depending on t. With $t = \langle a_2, b_2 \rangle$, we take $F_t^* = AC$ and thus $L'_t = \{c_1, c_2\}$. (If we had sampled b_1 from B in the previous step, we would take $F_t^* = BC$ and $L'_t = \{c_3\}$.) Then we sample each $y \in L'_t$ with probability

$$\begin{aligned}\frac{\mathsf{AGM}(Q_{(t,y)})}{\mathsf{AGM}(Q_t)} &= \frac{|R_{CD} \ltimes (t,y)|^{x_{CD}}|R_{CE} \ltimes (t,y)|^{x_{CE}}}{|R_{AC} \ltimes t|^{x_{AC}}|R_{BC} \ltimes t|^{x_{BC}}|R_{CD}|^{x_{CD}}|R_{CE}|^{x_{CE}}} \\ &= \frac{|R_{CD} \ltimes y|^{x_{CD}}|R_{CE} \ltimes y|^{x_{CE}}}{|R_{AC} \ltimes t|^{x_{AC}}|R_{BC} \ltimes t|^{x_{BC}}|R_{CD}|^{x_{CD}}|R_{CE}|^{x_{CE}}}.\end{aligned}$$

So c_1 and c_2 are sampled with probabilities $(\frac{1}{2})^{x_{AC}}(\frac{1}{3})^{x_{BC}}(\frac{3}{5})^{x_{CD}}(\frac{1}{5})^{x_{CE}}$ and $(\frac{1}{2})^{x_{AC}}(\frac{1}{3})^{x_{BC}}(\frac{1}{5})^{x_{CD}}(\frac{1}{5})^{x_{CE}}$, respectively. Note that $\mathsf{AGM}(Q_{\langle a_2,b_2,c_2 \rangle}) \neq 0$, although $\langle a_2, b_2, c_2 \rangle$ is not part of any valid join result. This is because the residual query Q_t is defined (see definition (2)) only over relations containing at least one free variable (i.e., attributes not appearing in t). This is exactly where we depart from Generic-Join: In Generic-Join, c_2 is not in L_t because it does not join with $t = \langle a_2, b_2 \rangle$ in R_{BC}. More precisely, $L_t = \pi_C(R_{AB} \ltimes t) \cap \pi_C(R_{BC} \ltimes t)$, but L'_t is only the smaller of the two sets. In general, L'_t is a superset of L_t, but as argued before, we cannot afford to compute this set intersection during sampling time, so can only sample from L'_t. In particular, this means that c_2 also has a chance to be sampled at this step, but it will be rejected immediately in the next recursive call, in line 2 of Algorithm 2. Note that this line is not needed in the Generic-Join algorithm, because every $y \in L_t$ is guaranteed to join with t in every relation. Although we have avoided computing L'_t, one immediate concern is that whether the sum of the sampling probabilities $\frac{\mathsf{AGM}(Q_{(t,y)})}{\mathsf{AGM}(Q_t)}$ over all $y \in L'_t$ would still be at most 1, as the query decomposition lemma only guarantees so when summed over L_t. We show in the next subsection that this is indeed still the case, which can be actually considered as a stronger version of the query decomposition lemma.

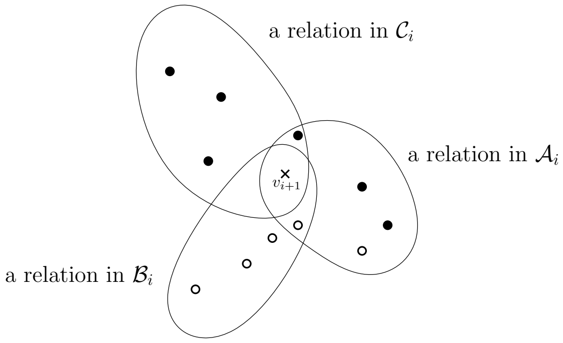

a relation in $\mathcal{C}_i$
a relation in $\mathcal{A}_i$
a relation in $\mathcal{B}_i$
v_{i+1}

Figure 2 Three types of relations in $\mathcal{E}_{\{v_{i+1}\}}$. The attributes in I are represented as solid disks and attributes in J' are represented as hollow circles.

Let us finish the example. As mentioned above, if c_2 is sampled, it will be rejected in the next step and we will start over. Now suppose c_1 is sampled. Then we move on to Generic-Join-Sample$(3, \langle a_2, b_2, c_1\rangle)$. When $i = 3$, F_t^* can only be CD, and $L'_t = \{d_1, d_2, d_3\}$. Each of them will be sampled with the same probability $(\frac{1}{3})^{x_{CD}}(\frac{1}{3})^{x_{DE}}$. Suppose d_3 is sampled, we move on to Generic-Join-Sample$(4, \langle a_2, b_2, c_1, d_3\rangle)$. Then $F_t^* = DE$, $L'_t = \{e_1\}$. e_1 will be sampled with probability $(\frac{1}{2})^{x_{CE}}(\frac{1}{1})^{x_{DE}}$. Finally, we call Generic-Join-Sample$(5, \langle a_2, b_2, c_1, d_3, e_1\rangle)$, which checks that e_1 joins with c_1 in R_{CE}, and then returns $\langle a_2, b_2, c_1, d_3, e_1\rangle$ as a sampled join result.

2.4 Correctness

In each step of the Generic-Join-Sample algorithm, we sample from some L'_t that is a superset of L_t, so the algorithm can reach every valid join result. In addition, each $y \in L'_t$ is still sampled with probability $\mathsf{AGM}(Q_{(t,y)})/\mathsf{AGM}(Q_t)$, so the uniformity argument in Section 2.1 that every join result is sampled with probability $1/\mathsf{AGM}(Q)$ is not affected. To prove the correctness of the algorithm, it only remains to show that $q_t = \sum_{y\in L'_t} \mathsf{AGM}(Q_{(t,y)}) \leq \mathsf{AGM}(Q_t)$, so that line 7 of Algorithm 2 is well defined.

▶ **Lemma 1.** *For any $i = 0, 1, \ldots, n$ and any tuple t on attributes $I = (v_1, v_2, \ldots, v_i)$, let F_t^* and L'_t be defined as in (5) and (6). Then*

$$\sum_{y\in L'_t} \mathsf{AGM}(Q_{(t,y)}) \leq \mathsf{AGM}(Q_t). \tag{7}$$

As mentioned, if L'_t is replaced by L_t in (7), this is just the query decomposition lemma. However, since L'_t is a superset of L_t, this requires another proof. Before giving the proof, we first cancel out the common factors on both sides on (7). The remaining factors are all on relations in $\mathcal{E}_{\{v_{i+1}\}}$. Denote $I \cup \{v_{i+1}\}$ as I' and $J \setminus \{v_{i+1}\}$ as J'. We partition $\mathcal{E}_{\{v_{i+1}\}}$ into the following three types:

1. $\mathcal{A}_i = \mathcal{E}_{\{v_{i+1}\}} \cap \mathcal{E}_I \cap \mathcal{E}_{J'}$.
2. $\mathcal{B}_i = \mathcal{E}_{\{v_{i+1}\}} \setminus \mathcal{E}_I$.
3. $\mathcal{C}_i = \mathcal{E}_{\{v_{i+1}\}} \setminus \mathcal{E}_{J'}$.

Please see Figure 2 for an illustration of these three types of relations. Note that $\mathcal{A}_i, \mathcal{B}_i, \mathcal{C}_i$ depend on the particular ordering of the attributes, and these three types of relations will also play an important role in characterizing the class of queries we can sample from efficiently. Now F_t^* can be equivalently defined as $F_t^* = \arg\min_{F\in\mathcal{A}_i\cup\mathcal{C}_i} |\pi_{v_{i+1}}(R_F \ltimes t)|$.

Note that $Q_{(t,y)}$ does not involve any relation in $\mathcal{C}_i$, so (7) can be simplified to the following:

$$\sum_{y \in L'_t} \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes (t,y)|^{x_F} \le \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i \cup \mathcal{C}_i} |R_F \ltimes t|^{x_F}. \tag{8}$$

Since only the relations in $\mathcal{A}_i, \mathcal{B}_i, \mathcal{C}_i$ are relevant, we introduce the notation

$$\begin{aligned} \mathsf{AGM}'(Q_{(t,y)}) &= \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes (t,y)|^{x_F}, \\ \mathsf{AGM}'(Q_t) &= \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i \cup \mathcal{C}_i} |R_F \ltimes t|^{x_F}. \end{aligned}$$

Define $q'_t = \sum_{y \in L'_t} \mathsf{AGM}'(Q_{(t,y)})$. Then to prove Lemma 1, we just need to prove

$$q'_t \le \mathsf{AGM}'(Q_t) \tag{9}$$

for all $i \ge 1$.

Proof. We first consider the case when $i = 0$ and $t = \langle\rangle$. In this case, $L_t = \bigcap_{F \in \mathcal{E}_{\{v_1\}}} \pi_{v_1} R_F$. Observe that for any $y \notin L_t$, there is some $F \in \mathcal{E}_{\{v_1\}}$ such that $R_F \ltimes y = \emptyset$. Also, F cannot be $\{v_1\}$ since we assumed that there are no relations of arity 1. Thus, $\mathsf{AGM}(Q_{(t,y)}) = \mathsf{AGM}(Q_y) = 0$ for any $y \notin L_t$. So $\sum_{y \in L'_t} \mathsf{AGM}(Q_{(t,y)}) = \sum_{y \in L_t} \mathsf{AGM}(Q_{(t,y)}) \le \mathsf{AGM}(Q_t)$, following directly from the query decomposition lemma (4). Next we show the case for $i \ge 1$.

In the following discussion, we fix an arbitrary $i \in [1, n]$. If $|R_F \ltimes t| = 0$ for some $F \in \mathcal{A}_i \cup \mathcal{B}_i \cup \mathcal{C}_i$, then (7) clearly holds because the both sides of (7) become 0. (In fact, the Generic-Join-Sample algorithm will never reach this case – such a t would be rejected in line 2.) Below, we assume $|R_F \ltimes t| \ge 1$ for every $F \in \mathcal{A}_i \cup \mathcal{B}_i \cup \mathcal{C}_i$.

Let $x_{\mathcal{A}_i\mathcal{B}_i} = \sum_{F \in \mathcal{A}_i \cup \mathcal{B}_i} x_F$ and $x_{\mathcal{C}_i} = \sum_{F \in \mathcal{C}_i} x_F$. Consider the following three cases.

1. $0 < x_{\mathcal{A}_i\mathcal{B}_i} < 1$. In this case, we have

$$\begin{aligned} q'_t &= \sum_{y \in L'_t} \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes (t,y)|^{x_F} \\ &= \sum_{y \in L'_t} \left(1 \cdot \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes (t,y)|^{x_F} \right) \\ &\le \left(\sum_{y \in L'_t} 1^{\frac{1}{1 - x_{\mathcal{A}_i\mathcal{B}_i}}} \right)^{1 - x_{\mathcal{A}_i\mathcal{B}_i}} \cdot \left(\sum_{y \in L'_t} \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes (t,y)|^{\frac{x_F}{x_{\mathcal{A}_i\mathcal{B}_i}}} \right)^{x_{\mathcal{A}_i\mathcal{B}_i}}, \end{aligned} \tag{10}$$

where the last inequality is due to Hölder's inequality. Applying Hölder's inequality again on the term in the second parentheses, we have

$$\sum_{y \in L'_t} \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes (t,y)|^{\frac{x_F}{x_{\mathcal{A}_i\mathcal{B}_i}}} \le \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} \left(\sum_{y \in L'_t} |R_F \ltimes (t,y)| \right)^{\frac{x_F}{x_{\mathcal{A}_i\mathcal{B}_i}}} \le \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes t|^{\frac{x_F}{x_{\mathcal{A}_i\mathcal{B}_i}}}. \tag{11}$$

Meanwhile, the term in the first parentheses of (10) is

$$\sum_{y \in L'_t} 1^{\frac{1}{1 - x_{\mathcal{A}_i\mathcal{B}_i}}} = |L'_t| = |\pi_{v_{i+1}}(R_{F^*_t} \ltimes t)|. \tag{12}$$

Substituting (11) and (12) into (10), we obtain

$$q'_t \le |\pi_{v_{i+1}}(R_{F_t^*} \ltimes t)|^{1-x_{\mathcal{A}_i\mathcal{B}_i}} \cdot \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes t|^{x_F}. \tag{13}$$

Note that when $x_{\mathcal{A}_i\mathcal{B}_i} < 1$, $\mathcal{C}_i$ cannot be empty. By the definition of F_t^*, we have

$$|\pi_{v_{i+1}}(R_{F_t^*} \ltimes t)| \le |\pi_{v_{i+1}}(R_F \ltimes t)| \le |R_F \ltimes t|$$

for any $F \in \mathcal{C}_i$. Also, since v_{i+1} is covered by $\mathcal{A}_i \cup \mathcal{B}_i \cup \mathcal{C}_i$, $x_{\mathcal{A}_i\mathcal{B}_i} + x_{\mathcal{C}_i} \ge 1$. Applying these to (13), we obtain

$$\begin{aligned} q'_t \le & |\pi_{v_{i+1}}(R_{F_t^*} \ltimes t)|^{x_{\mathcal{C}_i}} \cdot \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes t|^{x_F} \\ = & \prod_{F \in \mathcal{C}_i} |\pi_{v_{i+1}}(R_{F_t^*} \ltimes t)|^{x_F} \cdot \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes t|^{x_F} \\ \le & \prod_{F \in \mathcal{C}_i} |R_F \ltimes t|^{x_F} \cdot \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes t|^{x_F} = \mathsf{AGM}'(Q_t). \end{aligned}$$

2. $x_{\mathcal{A}_i\mathcal{B}_i} = 0$. Then $x_{\mathcal{C}_i} \ge 1$. Recall that we define $0^0 = 0$. Thus,

$$q'_t \le \sum_{y \in L'_t} 1 = |R_{F_t^*} \ltimes t| \le |R_{F_t^*} \ltimes t|^{x_{\mathcal{C}_i}} = \prod_{F \in \mathcal{C}_i} |R_{F_t^*} \ltimes t|^{x_F} \le \prod_{F \in \mathcal{C}_i} |R_F \ltimes t|^{x_F}, \tag{14}$$

where the last inequality follows from the definition of F_t^* and the observation that $|\pi_{v_{i+1}}(R_F \ltimes t)| = |R_F \ltimes t|$ for any $F \in \mathcal{C}_i$. If $\mathcal{A}_i \cup \mathcal{B}_i = \emptyset$, then (14) is the same as the RHS of (8). Otherwise, recall that we have assumed $|R_F \ltimes t| \ge 1$ for every $F \in \mathcal{A}_i \cup \mathcal{B}_i$, so (14) must be no more than $\mathsf{AGM}'(Q_t)$.

3. $x_{\mathcal{A}_i\mathcal{B}_i} \ge 1$. In this case, we apply Hölder's inequality directly on q'_t:

$$q'_t \le \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} \left(\sum_{y \in L'_t} |R_F \ltimes (t, y)| \right)^{x_F} \le \prod_{F \in \mathcal{A}_i \cup \mathcal{B}_i} |R_F \ltimes t|^{x_F}. \tag{15}$$

If $\mathcal{C}_i = \emptyset$, then (15) is the same as the RHS of (8). Otherwise, recall that we have assumed $|R_F \ltimes t| \ge 1$ for every $F \in \mathcal{C}_i$, so (15) must be no more than $\mathsf{AGM}'(Q_t)$. ◀

2.5 Weighted sampling

It remains to show how to perform the sampling step in line 5–7 of the Generic-Join-Sample algorithm. Recall from Section 2.2 that L'_t is just one of the neighbor lists, which are all available in the index, and we can find the right one in $O(1)$ time during each sampling step. However, since we do weighted sampling, a simple list is not enough. If we can compute all the sampling probabilities in advance, then we can build a weighted sampling data structure [6] on every neighbor list, which can then support drawing a weighted sample in $O(1)$ time when we conduct the sampling. The total preprocessing time will be linear since the total size of all the neighbor lists is linear, and the weighted sampling data structure can also be built in linear time [6].

Lines 5–7 of the Generic-Join-Sample algorithm sample each $y \in L'_t$ with probability $p_{(t,y)} = \frac{\mathsf{AGM}(Q_{(t,y)})}{\mathsf{AGM}(Q_t)}$. Canceling out the common factors on the numerator and the denominator, this can be simplified as (using the notation $\mathsf{AGM}'(Q_{(t,y)})$, $\mathsf{AGM}'(Q_t)$, and q'_t introduced after Lemma 1)

$$p_{(t,y)} = \frac{\mathsf{AGM}'(Q_{(t,y)})}{\mathsf{AGM}'(Q_t)}.$$

To sample a $y \in L'_t$ in constant time according to $p_{(t,y)}$, we perform rejection sampling in the following two steps:

(1) Sampling: sample a $y \in L'_t$ with probability $\frac{\mathsf{AGM}'(Q_{(t,y)})}{q'_t}$.
(2) Rejection: keep the sample with probability $\frac{q'_t}{\mathsf{AGM}'(Q_t)}$.

Note that $\mathsf{AGM}'(Q_t)$ can be computed in constant time by looking up $|R_F \ltimes t|$ from the index, so it only remains to describe how to do step (1) and compute q'_t. For each i, the corresponding sets $\mathcal{A}_i$, $\mathcal{B}_i$, $\mathcal{C}_i$ must be in the following there cases.

The case with $\mathcal{A}_i = \emptyset$

In this case, $\mathsf{AGM}'(Q_{(t,y)}) = \prod_{F\in\mathcal{B}_i} |R_F \ltimes (t,y)|^{x_F} = \prod_{F\in\mathcal{B}_i} |R_F \ltimes y|^{x_F}$; please also refer to the example in Section 2.3, where we made the same simplification. Note that this does not depend on t. Thus, we can precompute all $\mathsf{AGM}'(Q_{(t,y)})$ and construct a weighted sampling structure on every neighbor list $\pi_{v_{i+1}}(R_F \ltimes t)$ to support sampling step (1) in constant time. We also store the value of q'_t together with the list to do the rejection sampling (2) in constant time.

Going back to the example in Section 2.3, to support Generic-Join-Sample$(2,t)$ for any possible t, we precompute $\mathsf{AGM}'(Q_{(t,y)})$ for every $y \in \text{dom}(C)$: $\mathsf{AGM}'(Q_{(t,c_1)}) = 3^{x_{CD}}2^{x_{CE}}$, $\mathsf{AGM}'(Q_{(t,c_2)}) = \mathsf{AGM}'(Q_{(t,c_3)}) = 1^{x_{CD}}1^{x_{CE}}$, $\mathsf{AGM}'(Q_{(y,c_4)}) = 0$. Note that they do not actually depend on t. Using these as weights, we build a weighted sampling structure on each neighbor list of R_{AC} and R_{BC}, i.e., $\pi_C(R_{AC} \ltimes \langle A = a\rangle)$ for every $a \in \text{dom}(A)$ and $\pi_C(R_{BC} \ltimes \langle B = b\rangle)$ for every $b \in \text{dom}(B)$. Note that the total size of these neighbor list is $O(|R_{AC}| + |R_{BC}|)$. Then we store q_t with each neighbor list. For example, with the list $\pi_C(R_{AC} \ltimes \langle A = a_2\rangle)$, we store $p_t = \mathsf{AGM}'(Q_{(t,c_1)}) + \mathsf{AGM}'(Q_{(t,c_2)})$. During the call to Generic-Join-Sample$(2,t)$, we first decide F^*_t, and then use the neighbor list $\pi_C(R_{F^*_t} \ltimes t)$ to perform the sampling.

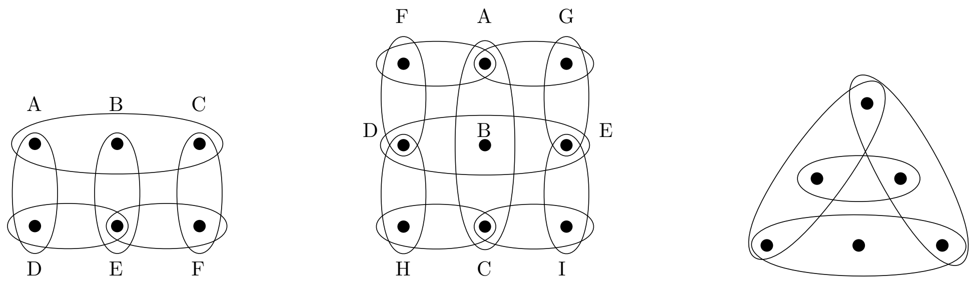
The case with $|\mathcal{A}_i| = 1$ and $\mathcal{C}_i = \emptyset$

When type-$\mathcal{A}$ relations are present, $\mathsf{AGM}'(Q_{(t,y)})$ will depend on t, which creates complications. However, when there is just one type-$\mathcal{A}$ relation (call it A) and no type-$\mathcal{C}$ relations, then we must have $F^*_t = A$. In this case, we are sampling a $y \in L'_t = \pi_{v_{i+1}}(R_A \ltimes t)$ with weights $\mathsf{AGM}'(Q_{(t,y)}) = |R_A \ltimes (t,y)|^{x_A} \prod_{F\in\mathcal{B}_i} |R_F \ltimes y|^{x_F}$. Although this depends on t, the key observation is that it only depends on the attributes of t that are included in $F^*_t = A$. Thus, we can precompute all the $\mathsf{AGM}'(Q_{(t,y)})$ and construct a weighted sampling structure on each neighbor list $\pi_{v_{i+1}}(R_A \ltimes t)$ to support sampling step (1) in constant time. We also store the value of q_t together with the list to do the rejection sampling (2) in constant time.

Other cases

Unfortunately, for other cases, we do not currently know how to preprocess the weights $\mathsf{AGM}'(Q_{(t,y)})$ in linear time and support constant-time sampling. Nevertheless, we can always compute all the $\mathsf{AGM}'(Q_{(t,y)})$ for $y \in L'_t$ on-the-fly, as well as q_t, which takes $O(|L'_y|) = O(\text{IN})$ time. This means that each sampling attempt will take $O(\text{IN})$ time as opposed to constant time.

We call a query Q *sequenceable* if there is an ordering of the attributes such that for every i, there is either $\mathcal{A}_i = \emptyset$, or $|\mathcal{A}_i| = 1$ and $\mathcal{C}_i = \emptyset$ for every i. Note that if Q is disconnected, Q is sequenceable iff every connected component is sequenceable. We arrive at the main result of this paper:



(a) Sequenceable in order: A, B, C, D, E, F.

(b) Sequenceable in order: A, B, C, D, E, F, G, H, I.

(c) Not sequenceable.

Figure 3 Sequenceable and non-sequenceable queries.

▶ **Theorem 2.** *Given a sequenceable join query, after a linear-preprocessing step, the Generic-Join-Sample algorithm returns a join result uniformly at random in expected time* $O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$*. For a non-sequenceable query, it returns a sample in expected time* $O\left(\frac{\mathrm{IN}^{\rho+1}}{\mathrm{OUT}}\right)$.

As a type-$\mathcal{A}$ relation must have at least 3 attributes, any query over binary relations is sequenceable. In addition, Figure 3 shows two sequenceable queries of higher arity, as well as a query that is not sequenceable. Indeed, the definition of sequenceable queries is a rather technical one, which follows from the two cases above that we know how to handle efficiently. Whether more general queries can be handled remains an interesting open problem.

3 Sampling from Join-Project Queries

In this section, we extend our sampling algorithm to *join-project queries* $\pi_O(Q)$ (a.k.a. *conjunctive queries*), where $O \subseteq \mathcal{V}$ is the set of output attributes. Let $\bar{O} = \mathcal{V} - O$. The algorithm consists of two simple steps: (1) Use Generic-Join-Sample to sample a join result t from $Q_O = \Join_{F\in\mathcal{E}_O} (\pi_O R_F)$. Note that it is possible that Q_O is disconnected; in which case we take a sample from each connected component, as described at the beginning of Section 2.2. (2) Check if $Q_{\bar{O}}(t) = \Join_{F\in\mathcal{E}_{\bar{O}}} (R_F \ltimes t)$ is empty. If not, we return t as a sampled result, otherwise we repeat.

The Correctness of the algorithm is straightforward: Observe that $\pi_O(Q) \subseteq Q_O$. The Generic-Join-Sample algorithm returns a sample t from Q_O uniformly at random. Then we return it iff $t \in \pi_O(Q)$, so every $t \in \pi_O(Q)$ has equal probability to be returned. Next we analyze its running time.

We will assume that Q_O is sequenceable. The analysis for the case when Q_O is not sequenceable is similar. We iterate steps (1) and (2) until a t sampled from Q_O is in $\pi_O(Q)$, which happens with probability $p = \frac{|\pi_O(Q)|}{|Q_O|}$. Let X_i denote the running time of the i-th iteration. Note that $X_i = 0$ if the i-th iteration does not take place. The total expected running time is thus $\sum_{i\geq 1} \mathsf{E}[X_i]$. Conditioned on the i-th iteration taking place, step (1) takes time $O\left(\frac{\mathsf{AGM}(Q_O)}{|Q_O|}\right) = O\left(\frac{\mathsf{AGM}(Q)}{|Q_O|}\right)$ in expectation. Step (2) takes time $O(\mathsf{AGM}(Q_{\bar{O}}(t))$, using any worst-case optimal join algorithm, e.g., Generic-Join. Because each $t \in Q_O$ is sampled with probability $1/|Q_O|$, the expected running time of step (2) is (the big-Oh of)

$$\sum_{t\in Q_O} \frac{1}{|Q_O|} \cdot \mathsf{AGM}(Q_{\bar{O}}(t)) = \frac{\sum_{t\in Q_O} \mathsf{AGM}(Q_{\bar{O}}(t))}{|Q_O|} \leq \frac{\mathsf{AGM}(Q)}{|Q_O|},$$

where the last inequality follows from the query decomposition lemma. Thus, we have

$\mathsf{E}[X_i \mid \text{the } i\text{-th iteration takes place}] = O\left(\frac{\mathsf{AGM}(Q)}{|Q_O|}\right)$. Since the i-th iteration takes place with probability $(1-p)^{i-1}$, we have $\mathsf{E}[X_i] = (1-p)^{i-1} \cdot O\left(\frac{\mathsf{AGM}(Q)}{|Q_O|}\right)$. Thus, the total expected running time is

$$\sum_{i\geq 1}(1-p)^{i-1} \cdot O\left(\frac{\mathsf{AGM}(Q)}{|Q_O|}\right) = \frac{1}{p} \cdot O\left(\frac{\mathsf{AGM}(Q)}{|Q_O|}\right) = \frac{|Q_O|}{|\pi_O(Q)|} \cdot O\left(\frac{\mathsf{AGM}(Q)}{|Q_O|}\right) = \frac{\mathsf{AGM}(Q)}{|\pi_O(Q)|}.$$

▶ **Theorem 3.** *Given a join-project query $\pi_O(Q)$, after a linear-preprocessing step, we can return a query result uniformly at random in expected time $O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$ if Q_O is sequenceable, and $O\left(\frac{\mathrm{IN}^{\rho+1}}{\mathrm{OUT}}\right)$ time otherwise.*

Note that Theorem 3 degenerates into Theorem 2 when $O = \mathcal{V}$.

4 Join Size Estimation

Because the Generic-Join-Sample algorithm succeeds in returning a random sample with probability exactly $\frac{\mathrm{OUT}}{\mathrm{IN}^\rho}$, this can be turned into a join size estimation algorithm using standard techniques. More precisely, we simply make k attempts, and see how many of them succeed. Suppose X out of the k attempts succeed, then we return $\mathrm{IN}^\rho \cdot \frac{X}{k}$ as an estimator of OUT.

It is obvious that this estimator is unbiased. Its variance is

$$\left(\frac{\mathrm{IN}^\rho}{k}\right)^2 \mathsf{Var}[X] \leq \left(\frac{\mathrm{IN}^\rho}{k}\right)^2 \cdot k \cdot \frac{\mathrm{OUT}}{\mathrm{IN}^\rho} = \frac{\mathrm{IN}^\rho \cdot \mathrm{OUT}}{k}.$$

To obtain a constant-factor approximation with constant probability, it is sufficient to make this variance smaller than $\mathrm{OUT}^2/4$, and it takes $k = O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$ attempts to achieve so. Since each attempt takes $O(1)$ time (assuming Q is sequenceable), this means that we can obtain a constant-factor approximation of OUT in $O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$ time.

However, a technical issue is that k depends on OUT, which is exactly the value we want to estimate. In [2], the standard technique of making repeated guesses for OUT by a binary search is used, which results in a logarithmic-factor increase in the running time. For our algorithm, a simpler strategy can be deployed: We simply keep repeating the attempts until c samples have been successfully obtained, where c is some constant. Note that the total running time is still $O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$ in expectation. Below, we show that it returns a constant-factor approximation of OUT with constant probability.

Note that in this version of the algorithm, k becomes a random variable. We need to show that with at least constant probability, we stop the algorithm with, say, $\frac{c}{2}\cdot\frac{\mathrm{IN}^\rho}{\mathrm{OUT}} \leq k \leq 2c\cdot\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}$. For $k \geq 2c\cdot\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}$ to happen, we must have collected less than c samples when $2c\cdot\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}$ attempts have been made. We know that in expectation, we should have collected $2c \cdot \frac{\mathrm{IN}^\rho}{\mathrm{OUT}} \cdot \frac{\mathrm{OUT}}{\mathrm{IN}^\rho} = 2c$ so far. As each attempt is independent, we can use the Chernoff inequality to bound

$$\Pr\left[k \geq 2c \cdot \frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right] \leq e^{-\frac{(\frac{1}{2})^2 \cdot 2c}{2}} = e^{-\frac{c}{4}},$$

which can be made as small as possible by choose c large enough. Using a similar argument, we can show that $k \leq \frac{c}{2} \cdot \frac{\mathrm{IN}^\rho}{\mathrm{OUT}}$ also happens with a constant probability small enough. Then by a union bound, $\frac{c}{2} \cdot \frac{\mathrm{IN}^\rho}{\mathrm{OUT}} \leq k \leq 2c \cdot \frac{\mathrm{IN}^\rho}{\mathrm{OUT}}$ happens with at least a constant probability.

Finally, standard techniques can be applied to boost the accuracy and success probability to achieve an (ϵ, δ) guarantee. We omit the detailed proof of the following result. This result also holds for join-project queries.

▶ **Theorem 4.** *Given a join-project query* $\pi_O(Q)$*, after a linear-preprocessing step, we can return a* $(1+\epsilon)$*-approximation of* OUT *with probability at least* $1-\delta$*. The running time is* $O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\cdot\frac{\mathrm{IN}^{\rho}}{\mathrm{OUT}}\right)$ *if* Q_O *is sequenceable, and* $O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\cdot\frac{\mathrm{IN}^{\rho+1}}{\mathrm{OUT}}\right)$ *otherwise.*

A more practical algorithm

If the goal is just to estimate the join size, not uniformly sampling, we can further improve the efficiency of the algorithm, by simply omitting rejection step (2) in Section 2.5. This means that a $y \in L'_t$ is always sampled and there is no rejection step in line 7 of the Generic-Join-Sample algorithm (it may still be rejected in line 2 of the next recursive call, though). This results in non-uniform samples, i.e., the sampling probability of a tuple t in each attempt, denoted as p_t, will be different for different t. We can no longer use the simple estimator as above. Instead, after each sampling attempt, we return the following Horvitz-Thompson estimator:

$$\tilde{X} = \begin{cases} \frac{1}{p_t}, & \text{if the attempt successfully returns } t; \\ 0, & \text{otherwise.} \end{cases}$$

Note that p_t can be computed as t is sampled, which is simply the product of the sampling probabilities used in sampling step (1) in Section 2.5. It can be easily shown that $\mathsf{E}[\tilde{X}] = \mathrm{OUT}$. Thus, we repeat the attempts and return the average of the above estimator.

The variance of the estimator is $\mathsf{Var}[\tilde{X}] = \sum_{t\in Q}\frac{1}{p_t} - \mathrm{OUT}^2$. Since skipping the rejection step only increases p_t, we have $p_t \geq \frac{1}{\mathsf{AGM}(Q)}$ for every $t \in Q$. Thus, $\mathsf{Var}[\tilde{X}]$ is always no larger than $O(\mathrm{IN}^{\rho}\cdot\mathrm{OUT})$, and the same theoretical guarantee from above applies. In practice, $\mathsf{Var}[\tilde{X}]$ can be much smaller, as demonstrated by the experimental results shown in the appendix.

5 Open Questions

The obvious open problem is if it is possible to improve the sampling time to $O(\mathrm{IN}^{\rho}/\mathrm{OUT})$ for non-sequenceable joins. Another intriguing question is whether the bound $O(\mathrm{IN}^{\rho}/\mathrm{OUT})$ is optimal. Note that the $\Omega(\mathrm{IN}^{\rho}/\mathrm{OUT})$ lower bound [2] assumes that the algorithm can only access the database through standard operations, such as sampling a tuple, looking up the degree, sampling a neighbor, etc. Because our algorithm builds auxiliary data structures (still in linear time though), this lower bound does not apply.

References

1 Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1999.

2 Sepehr Assadi, Mikhail Kapralov, and Sanjeev Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In *Proc. Innovations in Theoretical Computer Science*, 2019.

3 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

4 S. K. Bera and A. Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Symposium on Theoretical Aspects of Computer Science*, 2017.

5 Andreas Björklund, Rasmus Pagh, Virginia V. Williams, and Uri Zwick. Listing triangles. In *Proc. International Colloquium on Automata, Languages, and Programming*, 2014.

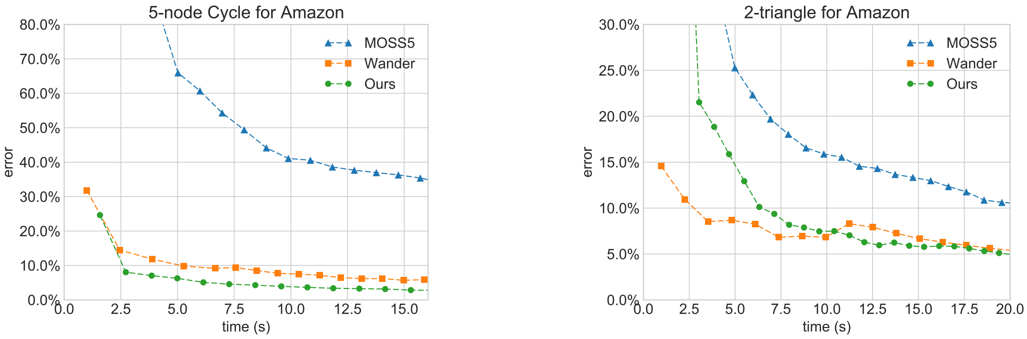
6 P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation.* Springer Verlag, 1983.
7 Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On Random Sampling over Joins. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1999.
8 T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In *Proc. IEEE Symposium on Foundations of Computer Science*, 2015.
9 Talya Eden, Dana Ron, and C. Seshadhri. On Approximating the Number of k-cliques in Sublinear Time. In *Proc. ACM Symposium on Theory of Computing*, 2018.
10 G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. Hypertree decompositions: structure, algorithms, and applications. In *Lecture Notes in Computer Science*, volume 3787, pages 1–15. Springer, 2005.
11 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proc. ACM Symposium on Principles of Database Systems*, 2017.
12 Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander Join: Online Aggregation via Random Walks. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2016.
13 Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *Proc. ACM Symposium on Principles of Database Systems*, pages 37–48, 2012.
14 Hung Q Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: New developments in the theory of join algorithms. *ACM SIGMOD Record*, 42(4):5–16, 2014.
15 C Seshadhri, Ali Pinar, and Tamara G Kolda. Triadic measures on graphs: the power of wedge sampling. In *Proc. SIAM International Conference on Data Mining*, 2013.
16 Todd Veldhuizen. Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proc. International Conference on Database Theory*, 2014.
17 Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John CS Lui, Don Towsley, Jing Tao, and Xiaohong Guan. MOSS-5: A Fast Method of Approximating Counts of 5-Node Graphlets in Large Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2017.
18 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. International Conference on Very Large Data Bases*, pages 82–94, 1981.
19 Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random Sampling over Joins Revisited. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2018.

A Experiments

After skipping the rejection sampling step, the join size estimation algorithm described above is very practical. Here we report some preliminary experimental results, comparing it with Wander Join [12] and MOSS-5 [17], two best heuristic algorithms for estimating the size of a cyclic join. We have also implemented and tested the other algorithm [2] with the same $O\left(\frac{\mathrm{IN}^{\rho}}{\mathrm{OUT}}\right)$ running time guarantee, but its performance is far worse than the other three. We used 4 real-world graph data sets and tested 2 cyclic queries as self-joins on each graph: a length-5 cycle and the two-triangle query as shown in Figure 1.

We used a machine with an Intel Xeon E5-2650 v4 2.20GHz CPU and 256G main memory for our experiments. To see how fast the estimator converges, when running an algorithm, we collect the estimated counts reported by the algorithm at regular intervals, say, every 1 second. Then we repeat the algorithm 100 times and compute the relative RMSE (root-mean-square error) to the true count at each time interval. We ran all algorithms in single-thread mode, but all algorithms can be easily parallelized as they all repeatedly take independent samples.

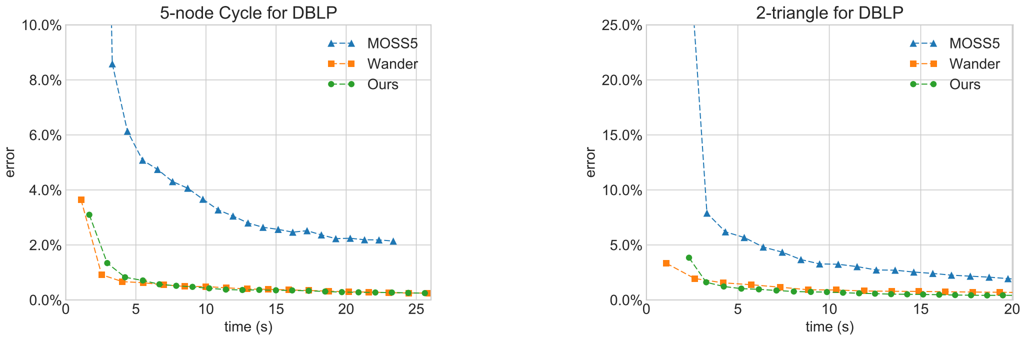
The experimental results are given in Figures 4–7. Note that the preprocessing time is included, which is why the curves do not start from time 0. We see that Wander Join has the shortest preprocessing time, as it only needs hash tables to be built. Our algorithm needs more preprocessing time to build weighted sampling data structures. MOSS-5 needs the longest preprocessing time to prepare the weighted sampling structures for all spanning trees. After preprocessing, the estimates returned by all algorithms converge to the true value as they are all unbiased estimators. The convergence rates are different, though. While Wander Join and MOSS-5 seem to behave differently on different data sets, our algorithm consistently performs on par with the better of the two, probably due to the theoretical guarantee it enjoys. On the other hand, the other $O\left(\frac{\mathrm{IN}^\rho}{\mathrm{OUT}}\right)$ algorithm [2] performs extremely poorly in the experiments, with no reasonable estimates returned after 60 seconds.



(a) Query: Cycle-5.

(b) Query: 2-triangle.

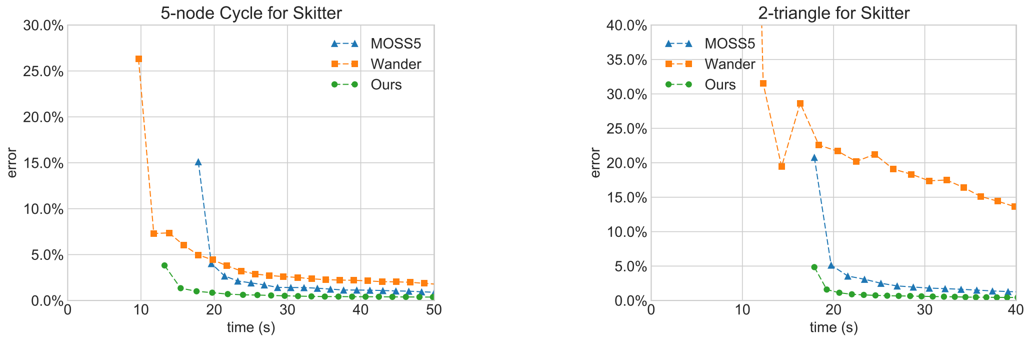
Figure 4 Experimental results on `amazon`.



(a) Query: Cycle-5

(b) Query: 2-triangle.

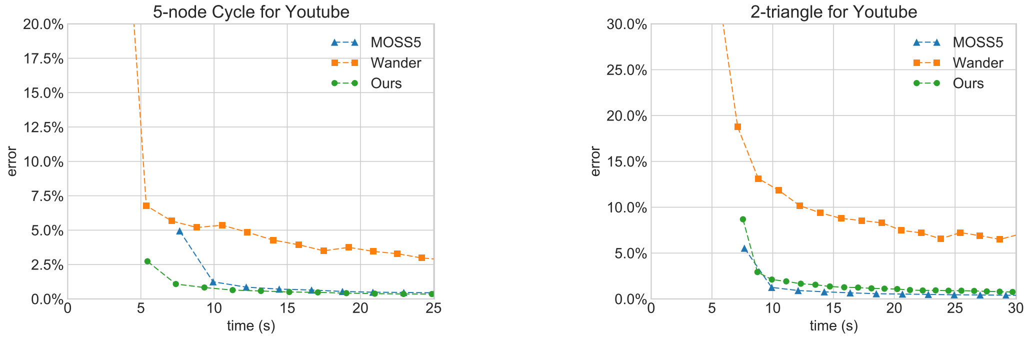
Figure 5 Experimental results on `dblp`.



(a) Query: Cycle-5.

(b) Query: 2-triangle.

Figure 6 Experimental results on `skitter`.



(a) Query: Cycle-5.

(b) Query: 2-triangle.

Figure 7 Experimental results on `youtube`.

Weight Annotation in Information Extraction

Johannes Doleschal
University of Bayreuth, Germany
Hasselt University, Belgium

Benny Kimelfeld
Technion – Israel Institute of Technology, Haifa, Israel

Wim Martens
University of Bayreuth, Germany

Liat Peterfreund
CNRS, IRIF – Université de Paris, France
University of Edinburgh, United Kingdom

Abstract

The framework of document spanners abstracts the task of information extraction from text as a function that maps every document (a string) into a relation over the document's spans (intervals identified by their start and end indices). For instance, the regular spanners are the closure under the Relational Algebra (RA) of the regular expressions with capture variables, and the expressive power of the regular spanners is precisely captured by the class of vset-automata – a restricted class of transducers that mark the endpoints of selected spans.

In this work, we embark on the investigation of document spanners that can annotate extractions with auxiliary information such as confidence, support, and confidentiality measures. To this end, we adopt the abstraction of provenance semirings by Green et al., where tuples of a relation are annotated with the elements of a commutative semiring, and where the annotation propagates through the (positive) RA operators via the semiring operators. Hence, the proposed spanner extension, referred to as an *annotator*, maps every string into an annotated relation over the spans. As a specific instantiation, we explore weighted vset-automata that, similarly to weighted automata and transducers, attach semiring elements to transitions. We investigate key aspects of expressiveness, such as the closure under the positive RA, and key aspects of computational complexity, such as the enumeration of annotated answers and their ranked enumeration in the case of numeric semirings. For a number of these problems, fundamental properties of the underlying semiring, such as positivity, are crucial for establishing tractability.

2012 ACM Subject Classification Information systems → Information extraction; Theory of computation → Transducers; Theory of computation → Problems, reductions and completeness; Theory of computation → Data provenance

Keywords and phrases Information extraction, regular document spanners, weighted automata, provenance semirings, K-relations

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.8

Related Version A full version of the paper is available at `https://arxiv.org/abs/1908.11642`.

Funding This work was supported by the German-Israeli Foundation for Scientific Research and Development (GIF), grant I-1502-407.6/2019. The work of Johannes Doleschal and Wim Martens was also supported by the Deutsche Forschungsgemeinschaft (DFG), grant MA 4938/4-1. The work of Benny Kimelfeld and Liat Peterfreund was also supported by the Israel Science Foundation (ISF), grants 1295/15 and 768/19, and the DFG project 412400621 (DIP program).
Liat Peterfreund: A part of the work was donewhile the author was affiliated with the Technion.

Acknowledgements We are grateful to Matthias Niewerth for many useful discussions and his help regarding Theorem 7.1 and Shaull Almagor for many helpful comments regarding weighted automata. Furthermore, we thank the anonymous reviewers for ICDT 2020 for many helpful remarks.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 8; pp. 8:1–8:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A plethora of paradigms have been developed over the past decades towards the challenge of extracting structured information from text – a task generally referred to as Information Extraction (IE). Common textual sources include natural language from a variety of sources such as scientific publications, customer input and social media, as well as machine-generated activity logs. Instantiations of IE are central components in text analytics and include tasks such as segmentation, named-entity recognition, relation extraction, and coreference resolution [38]. Rules and rule systems have consistently been key components in such paradigms, yet their roles have varied and evolved over time. Systems such as Xlog [42] and SystemT [4] use IE rules for materializing relations inside *relational query languages*. Machine-learning classifiers and probabilistic graphical models (e.g., Conditional Random Fields) use rules for *feature generation* [24, 44]. Rules serve as *weak constraints* (later translated into probabilistic graphical models) in Markov Logic Networks [32] and in the DeepDive system [43]. Rules are also used for generating *noisy training data* ("labeling functions") in the Snorkel system [34].

The framework of *document spanners* (*spanners* for short) provides a theoretical basis for investigating the principles of relational rule systems for IE [13]. Specifically, a spanner extracts from a document a relation over text intervals, called *spans*, using either atomic extractors or a relational query on top of the atomic extractors. More formally, by a *document* we refer to a string $\mathbf{d}$ over a finite alphabet, a *span* of $\mathbf{d}$ represents a substring of $\mathbf{d}$ by its start and end positions, and a *spanner* is a function that maps every document $\mathbf{d}$ into a relation over the spans of $\mathbf{d}$. The most studied spanner language is that of the *regular* spanners: atomic extraction is via *regex formulas*, which are regular expressions with capture variables, and relational manipulation is via the relational algebra: projection, natural join, union, and difference. Equivalently, the regular spanners are the ones expressible as *variable-set automata* (vset-automata for short), which are nondeterministic finite-state automata that can open and close variables (playing the role of the attributes of the extracted relation). Interestingly, there has been an independent recent effort to express artificial neural networks for natural language processing by means of finite-state automata [26, 27, 47].

To date, the research on spanners has focused on their expressive power [13, 17, 31], their computational complexity [2, 3, 15, 18], incompleteness [25, 30], and other system aspects such as cleaning [14] and distributed query planning [8]. That research has exclusively adopted a Boolean approach: *a tuple is either extracted or not.* Nevertheless, when applied to noisy or fuzzy domains such as natural language, modern approaches in artificial intelligence adopt a *quantitative* approach where each extracted tuple is associated with a level of confidence that the tuple coincides with the intent. When used within an end-to-end IE system, such confidence can be used as a principled way of tuning the balance between precision and recall. For instance, in probabilistic IE models (e.g., CRF), each extraction has an associated probability. In systems of weak constraints (e.g., MLN), every rule has a numerical weight, and the confidence in an extraction is an aggregation of the weights of the invoked rules that lead to the extraction. IE via artificial neural networks typically involves thresholding over a produced score or confidence value [5, 29]. Numerical scores in extraction are also used for quantifying the similarity between associated substrings, as done with sequence alignment and edit distance in the analysis of biological sequences such as DNA and RNA [45, 46].

In this work, we embark on the investigation of spanners that quantify the extracted tuples. We do so by adopting the concept of *annotated relations* from the framework of *provenance semirings* by Green et al. [20]. In essence, every tuple of the database is annotated with an element of a commutative semiring, and the positive relational algebra manipulates both the tuples and their annotations by translating relational operators into semiring operators (e.g., product for natural join and sum for union). An annotated relation is referred to as a $\mathbb{K}$-relation, where $\mathbb{K}$ is the domain of the semiring. The conceptual extension of the spanner model is straightforward: instead of a function (i.e., spanner) that maps every document $\mathbf{d}$ into a relation over the spans of $\mathbf{d}$, we consider a function that maps every $\mathbf{d}$ into a $\mathbb{K}$-relation over the spans of $\mathbf{d}$. We refer to such a function as a $\mathbb{K}$-*annotator*. Interestingly, as in the relational case, we can vary the meaning of the annotation by varying the semiring:

- Confidence via the *probability* (a.k.a. *inside*) semiring and the *Viterbi* (best derivation) semiring [19];
- Support (i.e., number of derivations) via the *counting* semiring [19];
- Access control via the semiring of the *confidentiality policies* [16] (e.g., does the extracted tuple require reading top-secret sections? which level suffices for the tuple?);
- The traditional spanners via the *Boolean* semiring.

As a specific instantiation of $\mathbb{K}$-annotators, we study the class of $\mathbb{K}$-*weighted vset-automata*. Such automata generalize vset-automata in the same manner as weighted automata and weighted transducers (cf., e.g., the Handbook of Weighted Automata [10]): transitions are weighted by semiring elements, the cost of a run is the product of the weights along the run, and the weight (annotation) of a tuple is the sum of costs of all the runs that produce the tuple. Again, there has been recent research that studies the connection between models of artificial neural networks in natural language processing and weighted automata [39]. Our investigation answers several fundamental questions about $\mathbb{K}$-weighted vset-automata:

1. Is this class closed under the positive relational algebra (according to the semantics of provenance semirings [20])?
2. What is the computational complexity of computing the annotation of a tuple?
3. Can we enumerate the annotated tuples as efficiently as we can do so for ordinary vset-automata (i.e., regular document spanners)?
4. In cases of numerical semirings (i.e., when $\mathbb{K}$ is a set of numbers), what is the complexity of enumerating the answers in ranked order by decreasing weight?

Our answers are mostly positive, put the last question aside, and show that $\mathbb{K}$-weighted vset-automata possess appropriate expressivity and tractability properties. As for the last question, we show that ranked enumeration is intractable and inapproximable for some of the aforementioned semirings (e.g., the probability and counting semirings), but tractable for positively ordered and bipotent semirings, such as the Viterbi semiring. Due to space constraints, we sometimes omit proofs or only provide a proof sketch.

2 Preliminaries

Our annotators will read documents and produce *annotated relations* [20], which are relations in which each tuple is annotated with an element from a semiring. In this section we revisit the basic definitions and properties of annotated relations.

Semirings

A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is an algebraic structure consisting of a set $\mathbb{K}$, containing two distinguished elements: the *zero* element $\overline{0}$ and the *unit* element $\overline{1}$, and equipped with two binary operations, namely *addition* $\oplus$ and *multiplication* $\otimes$ such that:

- $(\mathbb{K}, \oplus)$ is a commutative monoid with identity element $\overline{0}$;

- $(\mathbb{K}, \otimes)$ is a monoid with identity element $\overline{1}$;
- multiplication distributes over addition, i.e., $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ and $c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$;
- $\overline{0}$ is absorbing for $\otimes$, i.e., $\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$.

A semiring is called *commutative* if $(\mathbb{K}, \otimes)$ is commutative. We follow Green et al. [20] and assume that a semiring is commutative if not stated otherwise. Furthermore, following Eilenberg [11], a semiring is *positive* if the following conditions hold:

- $\overline{0} \neq \overline{1}$,
- If $a \oplus b = \overline{0}$, then $a = \overline{0} = b$.
- If $a \otimes b = \overline{0}$, then $a = \overline{0}$ or $b = \overline{0}$.

An element $a \in \mathbb{K}$ is a *zero divisor* if $a \neq \overline{0}$ and there is an element $b \in \mathbb{K}$ with $b \neq \overline{0}$ and $a \otimes b = \overline{0}$. Furthermore, an element $a \in \mathbb{K}$ has an *additive inverse*, if there is an element $b \in \mathbb{K}$ such that $a \oplus b = \overline{0}$. In the following, we will also identify a semiring by its domain $\mathbb{K}$ if the rest is clear from the context. When we do this for numeric semirings such as $\mathbb{R}$ and $\mathbb{N}$, we always assume the usual addition and multiplication.

▶ **Example 2.1.** The following are examples for commutative semirings. It is easy to verify that all but the numeric semirings and the Łukasiewcz semiring are positive.

1. The *numeric* semirings $(\mathbb{R}, +, \cdot, 0, 1)$ and $(\mathbb{Z}, +, \cdot, 0, 1)$;
2. The *counting* semiring $(\mathbb{N}, +, \cdot, 0, 1)$;
3. The *Boolean* semiring $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$ where $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$;
4. The *probability* semiring $(\mathbb{R}^+, +, \cdot, 0, 1)$.[1] Rabin [33] and Segala [40] define probabilistic automata over this semiring, where all edge weights must be between 0 and 1 and the sum of all edge weights starting some state, labeled by the same label must be 1;
5. The *Viterbi* semiring $([0, 1], \max, \cdot, 0, 1)$ which is used in probabilistic parsing [9];
6. The *access control semiring* $\mathbb{A} = (\{P < C < S < T < 0\}, \min, \max, 0, P)$, where P is “public”, C is “confidential”, S is “secret”, T is “top secret”, and 0 is “so secret that nobody can access it” [16];
7. The *tropical semiring* $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$ where min stands for the binary minimum function. This semiring is used in optimization problems of networks [9].
8. The Łukasiewcz semiring, whose domain is $[0, 1]$, with addition given by $x \oplus y = \max(x, y)$, with multiplication $x \otimes y = \max(0, x + y - 1)$, zero element 0, and unit 1. This semiring is used in multivalued logics [9].

Complexity-wise, we assume that single semiring elements can be stored in a single register and that addition and multiplication can be carried out in constant time – in similar spirit as the standard assumption for Random Access Machines. We use this assumption to simplify the analysis of algorithms.

2.1 Annotated Relations

We assume infinite and disjoint sets $\mathbf{D}$ and Vars, containing *data values* (or simply *values*) and *variables*, respectively. Let $V \subseteq \mathsf{Vars}$ be a finite set of variables. A *V-tuple* is a function $\mathbf{t} : V \to \mathbf{D}$ that assigns values to variables in V. The *arity* of $\mathbf{t}$ is the cardinality $|V|$ of V. For a subset $X \subseteq \mathsf{Vars}$, we denote the restriction of $\mathbf{t}$ to the variables in X by $\mathbf{t}{\restriction}X$. We denote the set of all the V-tuples by V-Tup. We sometimes leave V implicit when the precise set is

[1] One may expect the domain to be $[0, 1]$, but this is difficult to obtain while maintaining the semiring properties. For instance, defining $a \oplus b$ as $\min\{a + b, 1\}$ would violate distributivity.

not important. Let $\mathbb{K}$ be a set containing a distinguished element $\overline{0}$. A $(\mathbb{K}, \mathbf{D})$-*relation* R over V is a function $R : V\text{-}\mathsf{Tup} \to \mathbb{K}$ such that its *support* defined by $\mathsf{supp}(R) \overset{\text{def}}{=} \{\mathbf{t} \mid R(\mathbf{t}) \neq \overline{0}\}$ is finite. The *arity* of a $(\mathbb{K}, \mathbf{D})$-relation over V is $|V|$. When $\mathbf{D}$ is clear from the context or irrelevant, we also use $\mathbb{K}$-*relations* to refer to $(\mathbb{K}, \mathbf{D})$-relations.

▶ **Example 2.2.** The bottom left table in Figure 1 shows an example $(\mathbb{K}, \mathbf{D})$-relation, where $\mathbb{K}$ is the Viterbi semiring. The variables are x_{pers} and x_{loc}, so the V-tuples are described in the first two columns. The third column contains the element in $\mathbb{K}$ associated to each tuple.

Relational Algebra for Annotated Relations

Green et al. [20] defined a set of operators on $(\mathbb{K}, \mathbf{D})$-relations that naturally correspond to relational algebra operators and map $\mathbb{K}$-relations to $\mathbb{K}$-relations. Let $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ be a commutative semiring. The algebraic operators[2] *union*, *projection*, and *natural join* are defined in the usual way, for all finite sets $V_1, V_2 \subseteq \mathsf{Vars}$ and for all $\mathbb{K}$-relations R_1 over V_1 and R_2 over V_2, as follows.

- **Union**: If $V_1 = V_2$ then the union $R \overset{\text{def}}{=} R_1 \cup R_2$ is a function $R : V_1\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by $R(\mathbf{t}) \overset{\text{def}}{=} R_1(\mathbf{t}) \oplus R_2(\mathbf{t})$. (Otherwise, the union is not defined.)
- **Projection**: For $X \subseteq V_1$, the projection $R \overset{\text{def}}{=} \pi_X R_1$ is a function $R : X\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathbf{t}) \overset{\text{def}}{=} \bigoplus_{\mathbf{t}=\mathbf{t}'\restriction X \text{ and } R_1(\mathbf{t}')\neq\overline{0}} R_1(\mathbf{t}').$$
- **Natural Join:** The natural join $R \overset{\text{def}}{=} R_1 \bowtie R_2$ is a function $R : (V_1 \cup V_2)\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathbf{t}) \overset{\text{def}}{=} R_1(\mathbf{t}_1) \otimes R_2(\mathbf{t}_2)$$
where $\mathbf{t}_1$ and $\mathbf{t}_2$ are the restrictions $\mathbf{t}\restriction V_1$ and $\mathbf{t}\restriction V_2$, respectively.
- **Selection:** If $\mathbf{P}$ is a selection predicate that maps each tuple in $V_1\text{-}\mathsf{Tup}$ to either $\overline{0}$ or $\overline{1}$ then $R \overset{\text{def}}{=} \sigma_{\mathbf{P}}(R_1)$ is a function $R : V_1\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathbf{t}) \overset{\text{def}}{=} R_1(\mathbf{t}) \otimes \mathbf{P}(\mathbf{t}).$$

▶ **Proposition 2.3.** [20] *The above operators preserve the finiteness of the supports and therefore they map $\mathbb{K}$-relations into $\mathbb{K}$-relations.*

Hence, we obtain an algebra on $\mathbb{K}$-relations.

3 K-Annotators

We start by setting the basic terminology. We fix a finite alphabet Σ that is disjoint from Vars. A *document* is a finite sequence $\mathbf{d} = \sigma_1 \cdots \sigma_n$ where $\sigma_i \in \Sigma$ for each $i = 1, \ldots, n$. By Docs we denote the set of all documents. A *(k-ary) string relation* is a subset of Docs^k for some $k \in \mathbb{N}$.

[2] As in much of the work on semirings in provenance, e.g. Green et al. [20], we do not yet consider the *difference* operator (which would require additive inverses).

C	a	r	t	e	r	␣	f	r	o	m	␣	P	l	a	i	n	s	,	␣	G	e	o	r	g	i	a	,	␣	W	a	s	h	i	n	g	t	o	n	␣	f	r	o	m	␣	W	e	s	t	m	o	r	e	l	a	n	d	,	␣	V	i	r	g	i	n	i	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67

x_{pers}	x_{loc}	annotation
Carter	Plains,␣Georgia	0.9
Washington	Westmoreland,␣Virginia	0.9
Carter	Georgia,␣Washington	0.81
Carter	Westmoreland,␣Virginia	0.59049

x_{pers}	x_{loc}	annotation
$[1,7\rangle$	$[13,28\rangle$	0.9
$[30,40\rangle$	$[46,68\rangle$	0.9
$[1,7\rangle$	$[21,40\rangle$	0.81
$[1,7\rangle$	$[46,68\rangle$	0.59049

Figure 1 A document (top), a $(\mathbb{K},\mathbf{D})$-relation (bottom left), and an extracted annotated span relation (bottom right).

A *span* identifies a substring of a document $\mathbf{d}$ by specifying its bounding indices, that is, a span of $\mathbf{d}$ is an expression of the form $[i,j\rangle$ where $1 \leq i \leq j \leq n+1$. By $\mathbf{d}_{[i,j\rangle}$ we denote the substring $\sigma_i \cdots \sigma_{j-1}$. In case $i = j$ it holds that $\mathbf{d}_{[i,j\rangle}$ is the empty string, which we denote by ε. We denote by $\mathsf{Spans}(\mathbf{d})$ the set of all possible spans of a document $\mathbf{d}$ and by Spans the set of all possible spans of all possible documents. Since we will be working with relations over spans, we assume that $\mathbf{D}$ is such that $\mathsf{Spans} \subseteq \mathbf{D}$. A $(\mathbb{K},\mathbf{d})$-*relation* over $V \subseteq \mathsf{Vars}$ is defined analogously to a $(\mathbb{K},\mathbf{D})$-relation over V but only uses V-tuples with values from $\mathsf{Spans}(\mathbf{d})$.

▶ **Definition 3.1.** *Let* $(\mathbb{K},\oplus,\otimes,\overline{0},\overline{1})$ *be a semiring. A* $\mathbb{K}$-annotator *(or* annotator *for short), is a function S that is associated with a finite set $V \subseteq \mathsf{Vars}$ of variables and maps documents* $\mathbf{d}$ *into* $(\mathbb{K},\mathbf{d})$*-relations over V. We denote V by $\mathsf{Vars}(S)$. We sometimes also refer to an annotator as an* annotator over $(\mathbb{K},\oplus,\otimes,\overline{0},\overline{1})$ *when we want to emphasize the semiring.*

Notice that $\mathbb{B}$-annotators, i.e., annotators over $(\mathbb{B},\vee,\wedge,\mathsf{false},\mathsf{true})$ are simply the *document spanners* as defined by Fagin et al. [13].

▶ **Example 3.2.** We provide an example document $\mathbf{d}$ in Figure 1 (top). The table at the bottom right depicts a possible $(\mathbb{K},\mathbf{d})$-relation obtained by a spanner that extracts (person, hometown) pairs from $\mathbf{d}$. Notice that for each span $[i,j\rangle$ occurring in this table, the string $\mathbf{d}_{[i,j\rangle}$ can be found in the table to the left.

In this naïve example, which is just to illustrate the definitions, we used the Viterbi semiring and annotated each tuple with $(0.9)^k$, where k is the number of words between the spans associated to x_{pers} and x_{loc}. The annotations can therefore be interpreted as confidence scores.

Relational Algebra for K-Annotators

We now lift the relational algebra operators on $\mathbb{K}$-relations to the level of $\mathbb{K}$-annotators. For all documents $\mathbf{d}$ and for all annotators S_1 and S_2 associated with V_1 and V_2, respectively, we define the following:

- **Union:** If $V_1 = V_2$ then the union $S \stackrel{\text{def}}{=} S_1 \cup S_2$ is defined by $S(\mathbf{d}) \stackrel{\text{def}}{=} S_1(\mathbf{d}) \cup S_2(\mathbf{d})$.[3]
- **Projection:** For $X \subseteq V_1$, the projection $S \stackrel{\text{def}}{=} \pi_X S_1$ is defined by $S(\mathbf{d}) \stackrel{\text{def}}{=} \pi_X S_1(\mathbf{d})$.
- **Natural Join:** The natural join $S \stackrel{\text{def}}{=} S_1 \bowtie S_2$ is defined by $S(\mathbf{d}) \stackrel{\text{def}}{=} S_1(\mathbf{d}) \bowtie S_2(\mathbf{d})$.

[3] Here, $\cup$ stands for the union of two K-relations as was defined previously. The same is valid also for the other operators.

- **String selection:** Let R be a k-ary string relation. The string-selection operator σ^R is parametrized by k variables $x_1, \ldots, x_k$ in V_1 and may be written as $\sigma^R_{x_1,\ldots,x_k}$. Then the annotator $S \overset{\text{def}}{=} \sigma^R_{x_1,\ldots,x_k} S_1$ is defined as $S(\mathbf{d}) \overset{\text{def}}{=} \sigma_{\mathbf{P}}(S_1(\mathbf{d}))$ where $\mathbf{P}$ is a selection predicate with $\mathbf{P}(\mathbf{t}) = \overline{1}$ if $(\mathbf{d}_{\mathbf{t}(x_1)}, \ldots, \mathbf{d}_{\mathbf{t}(x_k)}) \in R$; and $\mathbf{P}(\mathbf{t}) = \overline{0}$ otherwise.

Due to Proposition 2.3 it follows that the above operators form an algebra on $\mathbb{K}$-annotators.

4 Weighted Variable-Set Automata

In this section, we define the concept of a *weighted vset-automaton* as a formalism to represent $\mathbb{K}$-annotators. This formalism is the natural generalization of vset-automata [13] and weighted automata [10]. Later in this section, we also present a formalism that is based on parametric factors, and a specification can be translated into a weighted vset-automaton (Section 4.1).

Let $V \in \mathsf{Vars}$ be a finite set of variables. Furthermore, let $\Gamma_V = \{v\vdash, \dashv v \mid v \in V\}$ be the set of *variable operations*.[4] Let $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring. A *weighted variable-set automaton over semiring* $\mathbb{K}$ (alternatively, a *weighted vset automaton* or a $\mathbb{K}$*-weighted vset-automaton*) is a tuple $A \overset{\text{def}}{=} (V, Q, I, F, \delta)$ where $V \subseteq \mathsf{Vars}$ is a finite set of variables; Q is a finite set of *states*; $I : Q \to \mathbb{K}$ is the *initial weight function*; $F : Q \to \mathbb{K}$ is the *final weight function*; and $\delta : Q \times (\Sigma \cup \{\varepsilon\} \cup \Gamma_V) \times Q \to \mathbb{K}$ is a *(*$\mathbb{K}$*-weighted) transition* function.

We define the *transitions* of A as the set of triples (p, o, q) with $\delta(p, o, q) \neq \overline{0}$. Likewise, the *initial* (resp., *accepting*) states are those states q with $I(q) \neq \overline{0}$ (resp., $F(q) \neq \overline{0}$). A *run* ρ of A over a document $\mathbf{d} \overset{\text{def}}{=} d_1 \cdots d_n$ is a sequence

$$(q_0, i_0) \overset{o_1}{\to} \cdots (q_{m-1}, i_{m-1}) \overset{o_{m-1}}{\to} (q_m, i_m)$$

where

- $i_0 = 1$, $i_m = n + 1$, and $i_j \in \{1, \ldots, n\}$ for each $j \in \{1, \ldots, m-1\}$;
- each o_j is in $\Sigma \cup \{\varepsilon\} \cup \Gamma_V$;
- $i_{j+1} = i_j$ whenever $o_j \in \{\varepsilon\} \cup \Gamma_V$ and $i_{j+1} = i_j + 1$, otherwise;
- $\delta(q_j, o_j, q_{j+1}) \neq \overline{0}$ for all $j \geq 0$.

The *weight* of a run is obtained by $\otimes$-multiplying the weights of its constituent transitions. Formally, the weight w_ρ of ρ is an element in $\mathbb{K}$ given by the expression

$$I(q_0) \otimes \delta(q_0, o_1, q_1) \otimes \cdots \otimes \delta(q_{m-1}, o_{m-1}, q_m) \otimes F(q_m) .$$

We call ρ *nonzero* if $\mathsf{w}_\rho \neq \overline{0}$. Notice that ρ is nonzero only if q_0 and q_m are initial and final, respectively. A run is called *valid* if for every variable $v \in V$ the following hold: there is exactly one index i for which $o_i = v\vdash$ and exactly one index $j \geq i$ for which $o_j = \dashv v$.

For a nonzero and valid run ρ, we define $\mathbf{t}_\rho$ as the V-tuple that maps each variable $v \in V$ to the span $[i_j, i_{j'}\rangle$ where $o_{i_j} = v\vdash$ and $o_{i_{j'}} = \dashv v$. We denote the set of all valid and nonzero runs of A on $\mathbf{d}$ by $P(A, \mathbf{d})$. We naturally extend the notion of functionality to apply also to general (not necessarily Boolean) weighted vset-automata. A *weighted functional vset-automaton* is a weighted vset-automaton whose runs are all valid.[5]

Notice that there may be infinitely many nonzero and valid runs of a weighted vset-automaton on a given document, due to ε*-cycles*, which are sets of states $\{q_1, \ldots, q_k\}$ such that $(q_i, \varepsilon, q_{i+1})$ is a transition for every $i \in \{1, \ldots, k-1\}$. Similar to much of the standard

[4] The operation $v\vdash$ represents opening variable v and $\dashv v$ represents closing v.

[5] Notice that, while our notion of functionality indeed generalizes the notion on $\mathbb{B}$-weighted vset-automata [13], one needs positivity of $\mathbb{K}$ to ensure that a functional automaton has an output tuple for every valid run.

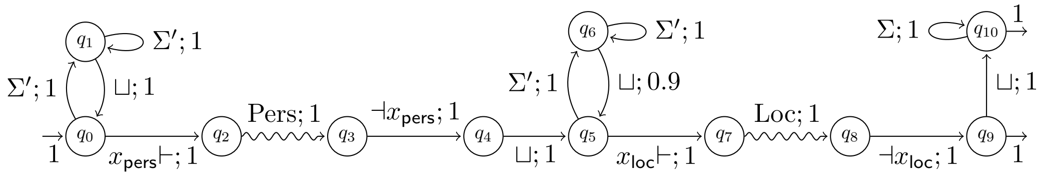


Figure 2 An example weighted vset-automaton over the Viterbi semiring with initial state q_0 and two final states q_9, q_{10}. $\Sigma' = \Sigma \setminus \{\sqcup\}$, Pers and Loc are sub-automata matching person and location names respectively. All edges, including the edges of the sub-automata, have the weight 1 besides the transition from q_6 to q_5 with weight 0.9.

literature on weighted automata (see, e.g., [12]) we will assume that weighted vset-automata do not have ε-cycles, unless mentioned otherwise. The reason for this restriction is that automata with such cycles need $\mathbb{K}$ to be closed under infinite sums for their semantics to be well-defined.[6]

As such, if A does not have ε-cycles, then the result of applying A on a document $\mathbf{d}$, denoted $[\![A]\!]^{\mathbb{K}}(\mathbf{d})$, is the $(\mathbb{K}, \mathbf{d})$-relation R for which

$$R(\mathbf{t}) \overset{\text{def}}{=} \bigoplus_{\rho \in P(A,\mathbf{d}) \text{ and } \mathbf{t}=\mathbf{t}_\rho} \mathsf{w}_\rho .$$

Note that we only use runs ρ that are valid and nonzero here. Observe that if $\mathbf{t}$ is a V'-tuple with $V' \neq V$ then $R(\mathbf{t}) = \overline{0}$. In addition, $[\![A]\!]^{\mathbb{K}}$ is well defined since every V-tuple in the support of $[\![A]\!]^{\mathbb{K}}(\mathbf{d})$ is a V-tuple over $\mathsf{Spans}(\mathbf{d})$. The *size* $|A|$ of a weighted vset-automaton A is its number of states plus its number of transitions.

We say that a $\mathbb{K}$-annotator S is *regular* if there exists a weighted vset-automaton A such that $S = [\![A]\!]^{\mathbb{K}}$. Similar to our terminology on annotators, we use the term $\mathbb{B}$-*weighted vset-automata* to refer to the "classical" vset-automata of Fagin et al. [13], which are indeed weighted vset-automata over the Boolean semiring.

▶ **Example 4.1.** Figure 2 shows an example weighted vset-automaton over the Viterbi semiring, which is intended to extract (person, hometown)-tuples from a document. Here, "Pers" and "Loc" should be interpreted as sub-automata that test if a string could be a person name or a location. (Such automata can be compiled from publicly available regular expressions[7] and from deterministic rules and dictionaries as illustrated in SystemT [4].)

The relation extracted by this automaton from the document in Figure 1 is exactly the annotated span relation of the same figure. The weight of a tuple $\mathbf{t}$ depends on the number of spaces occurring between the span captured by x_{pers} and the span captured by x_{loc}. More specifically the automaton assigns the weight $(0.9)^k$ to each tuple, where k is the number of words between the two variables.

4.1 Annotators via Parametric Factors

We now describe another way of introducing weights (or *softness*) in document spanners. This section can also be seen as an additional motivation for $\mathbb{K}$-annotators. Indeed, we will show that, if softness is introduced in document spanners [13] (i.e., $\mathbb{B}$-annotators) in

[6] The semirings need to fulfill additional properties as well such as distributivity, commutativity and associativity must also hold for infinite sums. Such semirings are called *complete* [28].

[7] For example, `http://regexlib.com/`.

the standard manner that we recall here, the resulting annotators can be captured in our framework.

Softness can be introduced in document spanners via the concept of *parametric factors*, which is a very common concept that is used in a wide range of contexts. Examples are the *soft keys* of Jha et al. [21], the *PrDB* model of Sen et al. [41], the *probabilistic unclean databases* of De Sa et al. [36] which can be viewed as a special case of the *Markov Logic Network* (MLN) [35]. Intuitively, a parametric factor is a succinct expression of numerical factors of a probability via weighted rules: whenever the rule *fires*, a corresponding factor (determined by the weight) is added to the product that constitutes the probability. What we want to show in this section is that, if one has rules that involve $\mathbb{B}$-annotators, and one adds uncertainty or softness to these rules in this standard way – using parametric factors – then the obtained formalism naturally leads to $\mathbb{K}$-annotators.

Next, we give the precise definition of a soft spanner and show that, when the factors are regular, a soft spanner can be translated into a weighted vset-automaton.

Formally, a *soft spanner* is a triple $Q = (P, \mathcal{S}, w)$, where:

- P is a document spanner, i.e., a $\mathbb{B}$-annotator,
- $\mathcal{S}$ is a finite set of document spanners referred to as the *factor spanners, and*
- $w : \mathcal{S} \to \mathbb{R}$ assigns a (positive or negative) numerical value to each factor spanner.

Given a document $\mathbf{d}$, the soft spanner Q assigns to each $\mathbf{t} \in P(\mathbf{d})$ a probability as follows:

$$\hat{Q}(\mathbf{d}, \mathbf{t}) \overset{\text{def}}{=} \exp\left(\sum_{S\in\mathcal{S}} \sum_{\mathbf{u}\in\{\mathbf{t}\}\bowtie S(\mathbf{d})} w(S)\right) = \prod_{S\in\mathcal{S}} e^{w(S)\cdot|\{\mathbf{t}\}\bowtie S(\mathbf{d})|},$$

$$Q(\mathbf{d}, \mathbf{t}) \overset{\text{def}}{=} \hat{Q}(\mathbf{d}, \mathbf{t})/Z(\mathbf{d}),$$

where $Z(\mathbf{d})$ is a normalization factor (or the *partition function*) defined in the usual way:

$$Z(\mathbf{d}) = \sum_{\mathbf{t}\in P(\mathbf{d})} \hat{Q}(\mathbf{d}, \mathbf{t})$$

Note that $\{\mathbf{t}\} \bowtie S(\mathbf{d})$ is the join of the relation $S(\mathbf{d})$ with the relation that consists of the single tuple $\mathbf{t}$. Hence, $|\{\mathbf{t}\} \bowtie S(\mathbf{d})|$ is the number of tuples $\mathbf{t}' \in S(\mathbf{d})$ that are *compatible* (*joinable*) with $\mathbf{t}$, that is, $\mathbf{t}(x) = \mathbf{t}'(x)$ whenever x is in the domain of both $\mathbf{t}$ and $\mathbf{t}'$.

▶ **Example 4.2.** The same relation as discussed in Example 4.1 can also be extracted using a soft spanner $Q = (P, \{S\}, w)$. To this end, P is a boolean spanner extracting (person, hometown)-tuples; S is the spanner, extracting $(x_{\mathsf{pers}}, y, x_{\mathsf{loc}})$-triples of words, where y matches a word between x_{pers} and x_{loc}; and the weight function w is the function assigning $w(S) = \log(0.9)$. Note that S simply extracts words and does not test whether the words matched by x_{pers} or x_{loc} correspond to a person or location.

We therefore see that $\mathbb{K}$-annotators can also be defined by applying the standard technique of parametric factors to document spanners. In fact, as we will see next, soft spanners can be compiled into weighted vset-automata, which serves as an additional motivation for weighted vset-automata. To prove the result, we use closure properties of weighted vset-automata that we will obtain further in the paper (so the proof can be seen as a motivation for the closure- and computational properties of weighted vset-automata as well).

For the following result, we say that a $\mathbb{K}$-weighted vset-automaton A is *unambiguous* if, for every document $\mathbf{d}$ and every tuple $\mathbf{t} \in [\![A]\!]^{\mathbb{K}}(\mathbf{d})$, there exists exactly one valid and nonzero run ρ of A on $\mathbf{d}$ such that $\mathbf{t} = \mathbf{t}_\rho$.

▶ **Theorem 4.3.** *Let $Q = (P, \mathcal{S}, w)$ be a soft spanner such that P and every $S \in \mathcal{S}$ is regular. There exists an $\mathbb{R}$-weighted vset-automaton A such that $[\![A]\!]^{\mathbb{R}}(\mathbf{d})(\mathbf{t}) = \log(\hat{Q}(\mathbf{d}, \mathbf{t}))$ for all documents $\mathbf{d}$ and tuples $\mathbf{t}$; Moreover, if the spanners of Q are represented as unambiguous functional vset-automata, then A can be constructed in polynomial time in the size of Q.*

Proof sketch. Let P_u be an unambiguous version of P, interpreted as an $\mathbb{R}$-weighted vset-automaton where true is associated with 1 and false with 0 and let V_P be the variables of P. Let S_u be an unambiguous version of S. From S_u we compute a weighted vset-automaton S_u^w by interpreting it as an $\mathbb{R}$-weighted vset-automaton and assigning to each accepting state q of S_u the weight $F(q) = w(S)$. Then the automaton we need for computing $\log(\hat{Q}(\mathbf{d}, \mathbf{t}))$ is

$$A = \bigcup_{S \in \mathcal{S}} \pi_{V_P}(P_u \bowtie S_u^w) \ .$$

We show correctness, i.e., $\log(\hat{Q}(\mathbf{d}, \mathbf{t})) = [\![A]\!]^{\mathbb{R}}(\mathbf{d})(\mathbf{t})$. Due to P_u and S_u^w being unambiguous, it follows directly that $P_u \bowtie S_u^w$ has exactly one accepting run with weight $w(S)$ for every tuple $\mathbf{t} \in [\![P_u \bowtie S_u^w]\!]^{\mathbb{R}}(\mathbf{d})$. Per definition of union and projection, it follows that $[\![A]\!]^{\mathbb{R}}(\mathbf{d})(\mathbf{t}) = \sum_{S \in \mathcal{S}} \sum_{\mathbf{u} \in \{\mathbf{t}\} \bowtie S(\mathbf{d})} w(S) = \log(\hat{Q}(\mathbf{d}, \mathbf{t}))$. As we will obtain in Theorem 5.5, automaton A can be represented as an $\mathbb{R}$-weighted vset-automaton and can be constructed in PTIME, which concludes the proof. ◀

5 Fundamental Properties

We now study fundamental properties of annotators. Specifically, we will show that regular annotators are closed under union, projection, and join. Furthermore, annotators over positive semirings are closed under exactly the same string relations as document spanners. We begin the section by showing that every regular $\mathbb{K}$-annotator can be transformed into an equivalent functional regular $\mathbb{K}$-annotator without ε-transitions. We say that two vset-automata A and A' are equivalent if $[\![A]\!]^{\mathbb{K}} = [\![A']\!]^{\mathbb{K}}$.

▶ **Proposition 5.1.** *For every weighted vset-automaton A there is an equivalent weighted vset-automaton A' that has no ε-transitions. This automaton A' can be constructed from A in PTIME. Furthermore, A is functional if and only if A' is functional.*

Notice that non-functional vset-automata can be inconvenient to work with, since some of their nonzero runs are not valid and therefore do not contribute to the weight of a tuple. It is therefore desirable to be able to automatically convert weighted vset-automata into functional weighted vset-automata.

▶ **Proposition 5.2.** *Let A be a weighted vset-automaton. Then there is a functional weighted vset-automaton A_{fun} that is equivalent to A. If A has n states and uses k variables, then A_{fun} can be constructed in time polynomial in n and exponential in k.*

The exponential blow-up in Proposition 5.2 cannot be avoided, since Freydenberger [17, Proposition 3.9] showed that there is a vset-automaton A (over $\mathbb{B}$) with one state and k variables, such that every equivalent functional vset-automaton has at least 3^k states. Functionality of vset-automata can be checked efficiently, as we have the following result.

▶ **Proposition 5.3.** *Given a weighted vset-automaton A with m transitions and k variables, it can be decided whether A is functional in time $O(km)$.*

▶ **Observation 5.4** ((Similar to Freydenberger et al. [18])). Let $A \overset{\text{def}}{=} (V, Q, I, F, \delta)$ be a $\mathbb{K}$-weighted functional vset-automaton. Then there exists a function $C : Q \times V \mapsto \{w, o, c\}$ that maps every state to its variable configuration, i.e., $C(q, x) \in \{w, o, c\}$ depending on whether x is *waiting, open,* or *closed* in state q. More formally, the function

$$C(q,v) = \begin{cases} w & \text{there is a nonzero run where } v\vdash \text{ does not occur before reaching } q, \\ o & \text{there is a nonzero run where } v\vdash \text{ but not } \dashv v \text{ occur before reaching } q, \\ c & \text{there is a nonzero run where } v\vdash \text{ and } \dashv v \text{ occur before reaching } q. \end{cases}$$

is well-defined. Indeed, if C would not be well-defined, then two conflicting runs would contradict the functionality of A.

5.1 Closure Under Join, Union, and Projection

Here we obtain the following result.

▶ **Theorem 5.5.** *Regular annotators are closed under union, projection, and natural join.*

Whereas union and projection are fairly standard, the case of join needs some care in the case that the two automata A_1 and A_2 process variable operations in different orders. (I.e., if A_1 processes $x\vdash y\vdash a \dashv y \dashv x$ and A_2 processes $y\vdash x\vdash a \dashv x \dashv y$, then these two different sequences produce the same result. The automata construction has to deal with this.) One can also show that, if the annotators are given as functional weighted vset-automata, then the construction for a single union, projection, and join can be done in polynomial time. Furthermore, the constructions preserve functionality.

5.2 Closure under String Selection

A k-ary string relation is *recognizable* if it is a finite union of Cartesian products of regular string languages [37]. Let $\mathsf{REG}_{\mathbb{K}}$ be the set of regular $\mathbb{K}$-annotators. We say that a k-ary string relation R is *selectable by regular* $\mathbb{K}$*-annotators* if the following equivalence holds:

$$\mathsf{REG}_{\mathbb{K}} = \{\sigma^{R}_{x_1,\ldots,x_k}(S) \mid S \in \mathsf{REG}_{\mathbb{K}} \text{ and } x_i \in \mathsf{Vars}(S) \text{ for all } 1 \le i \le k\} ,$$

that is, the class of $\mathbb{K}$-annotators is closed under selection using R. If $\mathbb{K} = \mathbb{B}$, we say that R is *selectable by document spanners.* Fagin et al. [13] proved that a string relation is recognizable *if and only if* it is selectable by document spanners. Here, we generalize this result in the context of weights and annotation. Indeed, it turns out that the equivalence is maintained for all positive semirings.

▶ **Theorem 5.6.** *Let* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *be a positive semiring and* R *be a string relation. The following are equivalent:*

(1) *R is recognizable.*

(2) *R is selectable by document spanners.*

(3) *R is selectable by $\mathbb{K}$-annotators.*

Proof sketch. The equivalence between (1) and (2) is known [13, Theorem 4.16]. The proof (2) ⇒ (3) is heavily based on the closure properties from Theorem 5.5 and does not use positivity of the semiring. For (3) ⇒ (2) we use semiring morphisms to turn $\mathbb{K}$-weighted vset-automata into $\mathbb{B}$-weighted vset-automata and need positivity of the semiring. ◀

Since the implication from (2) to (3) does not assume positivity of the semiring, it raises the question if the equivalence can be generalized even further. One can show that this is indeed the case, such as for the Łukasiewicz semiring, which is not positive.

6 Evaluation Problems

We consider two types of evaluation problems in this section: *answer testing* and *best weight evaluation*. The former is given an annotator, document $\mathbf{d}$, and tuple $\mathbf{t}$; and computes the annotation of $\mathbf{t}$ in $\mathbf{d}$ according to the annotator. The latter does not receive the tuple as input, but recieves a weight threshold and is asked whether there exists a tuple that is returned with a weight that is at least the threshold.

6.1 Answer Testing

It follows from Freydenberger [17, Lemma 3.1] that answer testing is NP-complete for $\mathbb{B}$-weighted vset-automata in general. Indeed, he showed that, given a $\mathbb{B}$-weighted vset-automaton A, it is NP-complete to check if A returns any output on the empty document ε, so it is even NP-complete to check if the tuple of empty spans is returned or not. However, the proof makes extensive use of non-functionality of the automaton. Indeed, we can prove that answer testing is tractable for functional weighted vset-automata.

▶ **Theorem 6.1.** *Given a functional weighted vset-automaton A, a document $\mathbf{d}$, and a tuple $\mathbf{t}$, the weight $[\![A]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t})$ assigned to $\mathbf{t}$ by A on $\mathbf{d}$ can be computed in PTIME.*

Proof sketch. Let A, $\mathbf{d}$, and $\mathbf{t}$ be as stated. Per definition, the weight assigned to $\mathbf{t}$ by A is

$$[\![A]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t}) \overset{\text{def}}{=} \bigoplus_{\rho \in P(A,\mathbf{d}) \text{ and } \mathbf{t}=\mathbf{t}_\rho} \mathsf{w}_\rho .$$

Therefore, in order to compute the weight $[\![A]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t})$, we need to consider the weights of all runs ρ for which $\mathbf{t} = \mathbf{t}_\rho$. Furthermore, multiple runs can select the same tuple $\mathbf{t}$ but assign variables in a different order.[8]

We first define an automaton $A_{\mathbf{t}}$, such that $[\![A_{\mathbf{t}}]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t}') = \overline{1}$ if $\mathbf{t} = \mathbf{t}'$ and $[\![A_{\mathbf{t}}]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t}') = \overline{0}$ otherwise. Such an automaton $A_{\mathbf{t}}$ can be defined using a chain of $|\mathbf{d}| + 2|V| + 1$ states, which checks that the input document is $\mathbf{d}$ and which has exactly one nonzero run ρ, with $\mathsf{w}_\rho = \overline{1}$ and $\mathbf{t}_\rho = \mathbf{t}$.

By Theorem 5.5 there is a weighted vset-automaton A' such that $[\![A']\!]^{\mathbb{K}} = [\![A \bowtie A_{\mathbf{t}}]\!]^{\mathbb{K}}$. It follows directly from the definition of A' that $[\![A]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t}) = [\![A']\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t})$. Furthermore, all accepting runs $\rho \in P(A', \mathbf{d})$ have length $|\mathbf{d}| + 2|V|$. Therefore, the weight $[\![A']\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t})$ can be obtained by taking the sum of the weights of all accepting runs of A'. If we assume w.l.o.g. that the states of A' are $\{1, \ldots, n\}$ for some $n \in \mathbb{N}$, then this sum can be computed as

$$[\![A']\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t}) = v_I \times (M_\delta)^{|\mathbf{d}|+2|V|} \times (v_F)^T ,$$

where

- v_I is the vector $(I(1), \ldots, I(n))$,
- M_δ is the $n \times n$ matrix with $M_\delta(i,j) = \bigoplus_{a \in \Sigma} \delta(i,a,j)$, and
- $(v_F)^T$ is the transpose of vector $v_F = (F(1), \ldots, F(n))$.

Since n is polynomial in the input, this product can be computed in polynomial time. ◀

[8] This may happen when variable operations occur consecutively, i.e., without reading a symbol in between.

6.2 Best Weight Evaluation

In many semirings, the domain is naturally ordered by some relation. For instance, the domain of the probability semiring is $\mathbb{R}^+$, which is ordered by the $\leq$-relation. This motivates evaluation problems where we are interested in some kind of optimization of the weight, which we will look into in this section.

▶ **Definition 6.2** ((Dorste and Kuich [9])). *A commutative monoid* $(\mathbb{K}, \oplus, \overline{0})$ *is* ordered *if it is equipped with a partial order* $\leqslant$ *preserved by the* $\oplus$ *operation. An ordered monoid is* positively ordered *if* $\overline{0} \leqslant a$ *for all* $a \in \mathbb{K}$*. A semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is* (positively) ordered *if the additive monoid is (positively) ordered and multiplication with elements* $\overline{0} \leqslant a$ *preserves the order.*

We consider the following two problems.

Threshold

Given: Regular annotator A over an ordered semiring, document $\mathbf{d} \in \mathsf{Docs}$, and a weight $w \in \mathbb{K}$.

Question: Is there a tuple $\mathbf{t}$ with $w \leqslant [\![A]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t})$?

MaxTuple

Given: Regular annotator A over an ordered semiring and a document $\mathbf{d} \in \mathsf{Docs}$.

Task: Compute a tuple with maximal weight, if it exists.

Notice that, if MaxTuple is efficiently solvable, then so is Threshold. We therefore prove upper bounds for MaxTuple and lower bounds for Threshold. The Threshold problem is sometimes also called the *emptiness problem* in the weighted automata literature. It turns out that, for positively ordered semirings that are bipotent (that is, $a \oplus b \in \{a, b\}$), both problems are tractable.

▶ **Theorem 6.3.** *Let* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *be a positively ordered, bipotent semiring. Furthermore, let* A *be a functional* $\mathbb{K}$*-weighted vset-automaton and let* $\mathbf{d} \in \mathsf{Docs}$ *be a document. Then* MaxTuple *for* A *and* $\mathbf{d}$ *can be solved in PTIME.*

Proof sketch. Since $a \oplus b \in \{a, b\}$ for every $a, b \in \mathbb{K}$, the weight of a tuple $t \in [\![A]\!]^{\mathbb{K}}(\mathbf{d})$ is always equal to the weight of one of the accepting runs ρ with $\mathbf{t} = \mathbf{t}_\rho$. Thus in order to find the tuple with maximal weight, we need to find the run of A on $\mathbf{d}$ with maximal weight. This boils down to finding a maximal weight path in a DAG, which is obtained by taking a "product" between A and $\mathbf{d}$. ◀

If the semiring is not bipotent, however, the Threshold and MaxTuple problems become intractable quickly.

▶ **Theorem 6.4.** *Let* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *be a semiring such that* $\bigoplus_{i=1}^{m} \overline{1}$ *is strictly monotonously increasing for increasing values of* m*. Futhermore let* A *be a functional* $\mathbb{K}$*-weighted vset-automaton, let* $\mathbf{d} \in \mathsf{Docs}$ *be a document, and* $k \in \mathbb{K}$ *be a weight threshold. Then* Threshold *for such inputs is NP-complete.*

Proof sketch. It is obvious that Threshold is in NP, as one can guess a tuple $\mathbf{t}$ and and test in PTIME whether $w \leqslant [\![A]\!]^{\mathbb{K}}(\mathbf{d})(\mathbf{t})$ using Theorem 6.1.

For the NP-hardness, we will reduce from MAX-3SAT. To this end, let $\psi = C_1 \wedge \cdots \wedge C_m$ be a boolean formula in 3CNF over variables $x_1, \ldots, x_n$ such that each clause $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ is a disjunction of exactly three literals $\ell_{i,j} \in \{x_c, \neg x_c \mid 1 \leq c \leq n\}$, $1 \leq i \leq k, 1 \leq j \leq 3$.

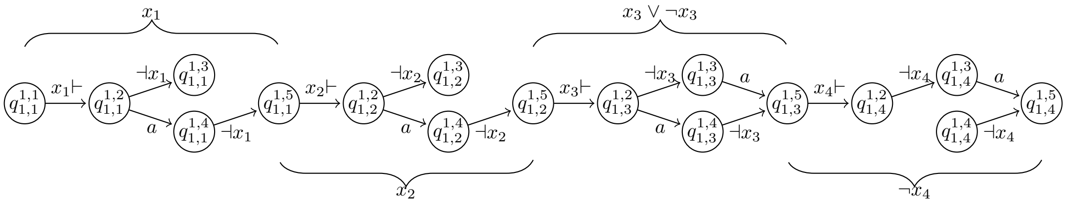


Figure 3 The sub-branch of A_ψ corresponding to C_1 and $x_1 = x_2 = 1, x_4 = 0$.

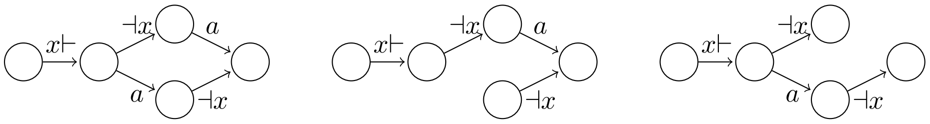


Figure 4 Example gadgets for variable x.

W.l.o.g., we can assume that no clause has two literals corresponding to the same variable. Observe that for each clause C_i there are $2^3 = 8$ assignments of the variables corresponding to the literals of C_i of which exactly 7 satisfy the clause C_i. Formally, let f_{C_i} be the function that maps a variable assignment τ to a number between 1 and 8, depending on the assignments of the literals of the clause C_i. W.l.o.g., we can assume that $f_{C_i}(\tau) = 8$ iff C_i is not satisfied by τ.

We will define a functional weighted automaton automaton A_ψ over the unary alphabet $\Sigma = \{a\}$ such that $[\![A_\psi]\!]^{\mathbb{K}}(a^n)(\mathbf{t}) = \bigoplus_{i=1}^{m} \overline{1}$ if and only if the assignment corresponding to $\mathbf{t}$ satisfies exactly m clauses in ψ and $[\![A_\psi]\!]^{\mathbb{K}}(\mathbf{d}) = \emptyset$ if $\mathbf{d} \neq a^n$.

To this end, each variable x_i of ψ is associated with a corresponding capture variable x_i of A_ψ. We associate a tuple $\mathbf{t}_\tau$ with every assignment τ such that

$$\mathbf{t}_\tau(x_i) = \begin{cases} [i, i\rangle & \text{if } \tau(x_i) = 0, \text{ and} \\ [i, i+1\rangle & \text{if } \tau(x_i) = 1. \end{cases}$$

The automaton $A_\psi \overset{\text{def}}{=} (V, Q, I, F, \delta)$ consists of m disjoint branches, where each branch corresponds to a clause of ψ; we call these *clause branches.* Each clause branch is divided into 7 sub-branches, such that a path in the sub-branch j corresponds to a variable assignment τ if $f_{C_i}(\tau) = j$. Thus, each clause branch has exactly one run ρ with weight $\overline{1}$ for each tuple $\mathbf{t}_\tau$ associated to a satisfying assignment τ of C_i.

More formally, the set of states $Q = \{q_{i,j}^{a,b} \mid 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq a \leq 7, 1 \leq b \leq 5\}$ contains $5n$ states for every of the 7 sub-branches of each clause branch. Intuitively, A_ψ has a gadget, consisting of 5 states, for each variable and each of the 7 satisfying assignments of each clause. Figure 4 depicts the three types of gadgets we use here. Note that the weights of the drawn edges are all $\overline{1}$. We use the left gadget if x does not occur in the relevant clause and the middle (resp., right) gadget if the literal $\neg x$ (resp., x) occurs. Furthermore, within the same sub-branch of A_ψ, the last state of each gadget is the same state as the start state of the next variable, i.e., $q_{i,j}^{a,5} = q_{i,j+1}^{a,1}$ for all $1 \leq i \leq k, 1 \leq j < n, 1 \leq a \leq 7$.

We illustrate the crucial part of the construction on an example. Let $\psi = (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$. The corresponding weighted vset-automaton A_ψ therefore has $14 = 2 \times 7$ disjoint branches. Figure 3 depicts the sub-branch for clause C_1 that corresponds to all assignments with $x_1 = x_2 = 1$ and $x_4 = 0$. ◀

We note that Theorem 6.3 and Theorem 6.4 give us tight bounds for all semirings we defined in Example 2.1.

Since MAX-3SAT is hard to approximate, we can turn Theorem 6.4 into an even stronger inapproximability result for semirings where approximation makes sense. To this end, we focus on semirings that contain $(\mathbb{N}, +, \cdot, 0, 1)$ (as a sub-semiring) in the following result.

▶ **Theorem 6.5.** *Let $\mathbb{K}$ be a semiring that contains $(\mathbb{N}, +, \cdot, 0, 1)$ and let A be a weighted vset-automaton over $\mathbb{K}$. Unless PTIME = NP, there is no algorithm that approximates the tuple with the best weight within a sub-exponential factor in PTIME.*

7 Enumeration Problems

In this section we consider computing the output of annotators from the perspective of enumeration problems, where we try to enumerate all tuples with nonzero weight, possibly from large to small. Such problems are highly relevant for (variants of) vset-automata, as witnessed by the recent literature on the topic [2,15]. We assume familiarity with terminology in enumeration algorithms such as *preprocessing time* and *delay*. If the order of the answers does not matter and the semiring is positive, we can guarantee a constant delay enumeration algorithm with linear preprocessing time.

▶ **Theorem 7.1.** *Given a weighted functional vset-automaton A over a positive semiring $\mathbb{K}$, and a document $\mathbf{d}$, the $\mathbb{K}$-Relation $[\![A]\!]^{\mathbb{K}}(\mathbf{d})$ can be enumerated with preprocessing linear in $|\mathbf{d}|$ and polynomial in $|A|$ and delay constant in $|\mathbf{d}|$ and polynomial in $|A|$.*

Note that the proof of the theorem essentially requires to go through the entire proof of the main result of Amarilli et al. [2, Theorem 1.1].

We now consider cases in which answers are required to arrive in a certain ordering.

Ranked Annotator Enumeration (RA-Enum)

Given: Regular functional annotator A over an ordered semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ and a document $\mathbf{d}$.

Task: Enumerate all tuples $\mathbf{t} \in [\![A]\!]^{\mathbb{K}}(\mathbf{d})$ in descending order on $\mathbb{K}$.

▶ **Theorem 7.2.** *Let $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ be an positively ordered, bipotent semiring, let A be a functional $\mathbb{K}$-weighted vset-automaton, and let $\mathbf{d} \in$ Docs be a document. Then RA-Enum can be solved with polynomial delay and preprocessing.*

Proof sketch. Our algorithm is a slight adaptation of Yen's algorithm [48]. To this end, we will use the DAG we defined in the proof of Theorem 6.3, but invest a bit more preprocessing. In particular, we change the DAG so that it has a one-to-one correspondence between output tuples and some of its paths. Using this correspondence, we can then revert to Yen's algorithm for enumerating simple paths in graphs. ◀

8 Concluding Remarks

We embarked on a study that incorporates *annotations* or *weights* in information extraction and propose $\mathbb{K}$-annotators as a candidate formalism to study this problem. The $\mathbb{K}$-annotators can be instantiated with *weighted vset-automata*, thereby obtaining *regular $\mathbb{K}$-annotators*, which are powerful enough to capture the extension of the traditional spanner framework with parametric factors. Furthermore, the regular $\mathbb{K}$-annotators have favorable closure properties, such as closure under union, projection, natural join, and string selection using regular relations. The first complexity results on evaluation problems are encouraging: answer testing is tractable and, depending on the semiring, problems such as the threshold problem, the max tuple problem, and enumeration of answers are tractable too.

We note that the addition of weights to vset-automata also introduces new challenges. For instance, some typical questions that we study in database theory are not yet fully understood for weighted automata, which are the basis of weighted vset-automata. Examples are equivalence and emptiness. Concerning equivalence, one can show that equivalence is undecidable for weighted vset-automata over the tropical semiring, using techniques from Krob [23] or Almagor et al. [1]. In general, however, it is not completely clear for which semirings equivalence is decidable or not.

The emptiness problem that is usually studied in the weighted automata literature does not ask if there exists a document **d** such that the automaton returns at least one tuple with nonzero weight on **d**, but is additionally given a threshold (as in our THRESHOLD problem) and asks if the automaton returns a tuple with at least the threshold weight (which requires an order on the semiring). It is not yet clear how much this threshold influences the complexity of the problem.

An additional challenge is that *determinization* of weighted automata is a complex matter and not always possible. It is well-known to be possible for the Boolean semiring but, for the tropical semiring, i.e., $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$, deterministic weighted automata are strictly less expressive than unambiguous weighted automata, which are strictly less expressive than general weighted automata, cf. Klimann et al. [22].

A possible direction for further exploration could be the study of annotators which use regular cost functions (cf. Colcombet [6]) instead of weighted automata. Since regular cost functions are restructed to the domain of the natural numbers, this would probably be most interesting in the case where the semiring domain is (a subset of) the natural numbers. Indeed, in this case, it is known that regular cost functions are strictly more expressive than weighted automata over the tropical semiring (cf. Colcombet et al. [7]) and therefore could provide a useful tool to annotate document spanners. On the other hand, it is not yet clear to us how to associate regular cost functions in a natural way to annotated relations, which require semirings.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's Decidable about Weighted Automata? In *Automated Technology for Verification and Analysis*, pages 482–491, 2011.

2 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. In *ICDT*, pages 22:1–22:19, 2019.

3 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient Logspace Classes for Enumeration, Counting, and Uniform Generation. In *PODS*, pages 59–73, 2019.

4 Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. SystemT: An Algebraic Approach to Declarative Information Extraction. In *ACL*, pages 128–137, 2010. URL: `http://www.aclweb.org/anthology/P10-1014`.

5 Jason P. C. Chiu and Eric Nichols. Named Entity Recognition with Bidirectional LSTM-CNNs. *TACL*, 4:357–370, 2016.

6 T. Colcombet. Logic and regular cost functions. In *LICS*, pages 1–4, 2017. `doi:10.1109/LICS.2017.8005061`.

7 Thomas Colcombet, Denis Kuperberg, Amaldev Manuel, and Szymon Torunczyk. Cost Functions Definable by Min/Max Automata. In *STACS)*, volume 47, pages 29:1–29:13, 2016. `doi:10.4230/LIPIcs.STACS.2016.29`.

8 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-Correctness in Information Extraction. In *PODS*, pages 149–163, 2019. doi:10.1145/3294052.3319684.

9 Manfred Droste and Werner Kuich. *Semirings and Formal Power Series*, pages 3–28. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-01492-5_1.

10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.

11 Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, Inc., Orlando, FL, USA, 1974.

12 Zoltán Ésik and Werner Kuich. *Finite Automata*, pages 69–104. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-01492-5_3.

13 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. *J. ACM*, 62(2):12, 2015. doi:10.1145/2699442.

14 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative Cleaning of Inconsistencies in Information Extraction. *ACM Trans. Database Syst.*, 41(1):6:1–6:44, 2016. doi:10.1145/2877202.

15 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant Delay Algorithms for Regular Document Spanners. In *PODS*, pages 165–177, 2018.

16 J. Nathan Foster, Todd J. Green, and Val Tannen. Annotated XML: queries and provenance. In *PODS*, pages 271–280, 2008.

17 Dominik D. Freydenberger. A Logic for Document Spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019.

18 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining Extractions of Regular Expressions. In *PODS*, pages 137–149, 2018.

19 Joshua Goodman. Semiring Parsing. *Computational Linguistics*, 25(4):573–605, 1999.

20 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40, 2007. doi:10.1145/1265530.1265535.

21 Abhay Kumar Jha, Vibhor Rastogi, and Dan Suciu. Query evaluation with soft-key constraints. In *PODS*, pages 119–128, 2008.

22 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004. doi:10.1016/j.tcs.2004.02.049.

23 Daniel Krob. The Equality Problem for Rational Series with Multiplicities in the tropical Semiring is Undecidable. *IJAC*, 4(3):405–426, 1994. doi:10.1142/S0218196794000063.

24 Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning*, volume 3635 of *Lecture Notes in Computer Science*, pages 319–339, 2004.

25 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document Spanners for Extracting Incomplete Information: Expressiveness and Complexity. In *PODS*, pages 125–136, 2018.

26 Franz Mayr and Sergio Yovine. Regular Inference on Artificial Neural Networks. In *CD-MAKE*, volume 11015 of *Lecture Notes in Computer Science*, pages 350–369, 2018.

27 Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing Formal Languages: A Comparison Between Finite Automata and Recurrent Neural Networks. In *ICLR (Poster)*, 2019.

28 Mehryar Mohri. *Weighted Automata Algorithms*, pages 213–254. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-01492-5_6.

29 Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-Sentence N-ary Relation Extraction with Graph LSTMs. *TACL*, 5:101–115, 2017.

30 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity Bounds for Relational Algebra over Document Spanners. In *PODS*, pages 320–334. ACM, 2019.

31 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive Programs for Document Spanners. In *ICDT*, volume 127, pages 13:1–13:18, 2019.

32 Hoifung Poon and Pedro M. Domingos. Joint Inference in Information Extraction. In *AAAI*, pages 913–918, 2007.

33 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. doi:10.1016/S0019-9958(63)90290-0.

34 Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid Training Data Creation with Weak Supervision. *PVLDB*, 11(3):269–282, 2017.

35 Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Mach. Learn.*, 62(1-2):107–136, 2006. doi:10.1007/s10994-006-5833-1.

36 Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A Formal Framework for Probabilistic Unclean Databases. In *ICDT*, volume 127, pages 6:1–6:18, 2019.

37 Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009. doi:10.1017/CBO9781139195218.

38 Sunita Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008. doi:10.1561/1900000003.

39 Roy Schwartz, Sam Thomson, and Noah A. Smith. Bridging CNNs, RNNs, and Weighted Finite-State Machines. In *ACL*, pages 295–305, 2018.

40 Roberto Segala. Probability and Nondeterminism in Operational Models of Concurrency. In *CONCUR*, pages 64–78, 2006.

41 Prithviraj Sen, Amol Deshpande, and Lise Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

42 Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative Information Extraction Using Datalog with Embedded Extraction Predicates. In *VLDB*, pages 1033–1044, 2007. URL: http://www.vldb.org/conf/2007/papers/research/p1033-shen.pdf.

43 Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental Knowledge Base Construction Using DeepDive. *PVLDB*, 8(11):1310–1321, 2015. URL: http://www.vldb.org/pvldb/vol8/p1310-shin.pdf.

44 Charles A. Sutton and Andrew McCallum. An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

45 David Torrents, Mikita Suyama, Evgeny Zdobnov, and Peer Bork. A genome-wide survey of human pseudogenes. *Genome research*, 13(12):2559–2567, 2003.

46 Lusheng Wang and Tao Jiang. On the Complexity of Multiple Sequence Alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

47 Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. In *ICML*, volume 80, pages 5244–5253, 2018.

48 Jin Y. Yen. Finding the k Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, 1971. URL: http://www.jstor.org/stable/2629312.

Containment of UC2RPQ: The Hard and Easy Cases

Diego Figueira
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France
https://www.labri.fr/~dfigueir
diego.figueira@labri.fr

Abstract

We study the containment problem for UC2RPQ, that is, two-way Regular Path Queries, closed under conjunction, projection and union. We show a dichotomy property between PSpace-c and ExpSpace-c based on a property on the underlying graph of queries. We show that for any class $\mathscr{C}$ of graphs, the containment problem for queries whose underlying graph is in $\mathscr{C}$ is in PSpace if and only if $\mathscr{C}$ has bounded *bridgewidth*. Bridgewidth is a graph measure we introduce to this end, defined as the maximum size of a minimal edge separator of a graph.

2012 ACM Subject Classification Information systems → Graph-based database models; Information systems → Resource Description Framework (RDF); Mathematics of computing → Graph theory; Theory of computation → Formal languages and automata theory

Keywords and phrases Regular Path Queries (RPQ), 2RPQ, CRPQ, C2RPQ, UC2RPQ, graph databases, containment, inclusion, equivalence, dichotomy, graph measure, bridge-width (bridgewidth), minimal edge separator, minimal cut-set, max-cut, tree-width (treewidth)

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.9

Related Version https://hal.archives-ouvertes.fr/hal-02291888

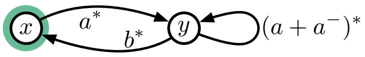
Funding ANR DéLTA (grant ANR-16-CE40-0007) and ANR QUID (grant ANR-18-CE40-0031)

Acknowledgements Thanks to Guillaume Lagarde, Matthias Niewerth and anonymous reviewers for helpful comments.

1 Introduction

Graph databases is a prominent area of study within database theory, in which the use of recursive queries is crucial [4, 2]. A graph database is a finite edge-labeled directed graph. The most basic navigational querying mechanism for graph databases corresponds to the class of *regular path queries* (RPQs), which test whether two nodes of the graph are connected by a path whose label belongs to a given regular language. RPQs are often extended with the ability to traverse edges in both directions, giving rise to the class of *two-way* RPQs, or 2RPQs [9]. For example, an regular expression like $(a + a^-)^*$ states that there is a path that can traverse a-labelled edges either in the forward or reverse direction. The core of the most popular recursive query languages for graph databases is defined by *conjunctive* 2RPQs, or C2RPQs, which are the closure of 2RPQs under conjunction and existential quantifications [7]. As an example, a C2RPQ could be described as a formula γ like

$$\gamma(x) = (\exists y) \quad x \xrightarrow{a^*} y \quad \wedge \quad y \xrightarrow{(a+a^-)^*} y \quad \wedge \quad y \xrightarrow{b^*} x.$$



Here γ is a unary formula that outputs all vertices v_x of the graph database from which there is an a labelled path to some vertex v_y contained in some undirected a-cycle, and so that there is a directed b-path from v_y to v_x. Note that the query can also be depicted as a graph (on the right), edge-labelled with languages and with some highlighted vertices representing free (output) variables. We also consider *unions* of C2RPQs, or UC2RPQs.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 9; pp. 9:1–9:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *containment problem* is arguably the most basic static analysis problem on monotone query languages. In our case it is the problem of, given two UC2RPQ γ, γ', whether $\gamma(\mathcal{G}) \subseteq \gamma'(\mathcal{G})$ for all graph databases $\mathcal{G}$.

The "four Italians" [7] have long ago shown that the containment problem for UC2RPQ is decidable, ExpSpace-complete (in particular generalizing a prior ExpSpace upper bound for CRPQ [11]). On the other hand, Barceló *et al.* [6] have shown that on the class $\mathscr{A}$ of acyclic multi-graphs[1], there is a measure of "*width*" so that for every acyclic class $\mathscr{C} \subseteq \mathscr{A}$ of bounded *width*, the containment problem on UC2RPQ whose underlying graph is in $\mathscr{C}$ is in PSpace.

Contribution. Here we lift their notion of *width* to arbitrary multi-graphs. We call our measure "*bridgewidth*" and we obtain that for every class $\mathscr{C}$ of multi-graphs:

- If $\mathscr{C}$ has bounded bridgewidth, the containment problem for UC2RPQ whose underlying graph is in $\mathscr{C}$ is in PSpace;
- If $\mathscr{C}$ has unbounded bridgewidth, the containment problem for UC2RPQ whose underlying graph is in $\mathscr{C}$ is ExpSpace-complete.

Further, the lower bound hold also if we replace UC2RPQ with CRPQ or even with Boolean CRPQ, and the upper bounds hold also if we allow any arbitrary UC2RPQ in the left-hand side of the containment problem.

But what is *bridgewidth*? It is a rather intuitive measure of graphs. A *bridge* is a minimal edge separator, that is, a minimal set of edges whose removal increases the number of connected components of the graph. The bridgewidth of a graph is the maximum size of a bridge therein. As it turns out, on the class of acyclic graphs $\mathscr{A}$, bridgewidth and [6]'s *width* measure coincide.

Another related static analysis problem is the *boundedness problem* (*i.e.*, whether a UC2RPQ is equivalent to one whose languages are all finite). We have recently shown that, similarly to the containment problem, this problem is ExpSpace-complete and that on acyclic UCRPQ queries of bounded width it is in PSpace [5]. We conjecture that an adaptation of bridgewidth may also characterize the complexity for this problem for UCRPQ (see Section 7).

Organization. We start with necessary basic definitions of graphs, bridges, graph databases, automata, and UC2RPQ in Section 2, and we formally state the main result in Section 3. Section 4 contains some technical lemmas needed for the upper bound algorithm. Finally, Sections 5 and 6 spell out the details of the upper and lower bounds of the main theorem, respectively. We conclude with some remarks in Section 7.

2 Preliminaries

Graphs

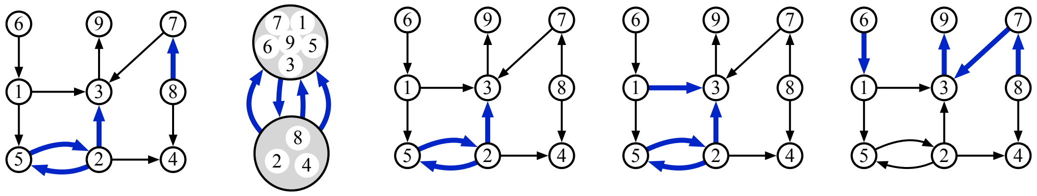
A **multi-graph** $M = (V, E, \eta)$ is a finite set of vertices V and edges E together with a function $\eta : E \to V \times V$ associating each edge with the source and target vertices. For convenience we will sometimes use $\eta_1(e)$ and $\eta_2(e)$ to denote the vertex in the first and second components of $\eta(e)$. For economy, we will henceforth write "graph" to denote a multi-graph. For any set

[1] In this context, by acyclic we mean any directed multi-graph such that every cycle of its underlying undirected simple graph is a self-loop.

of edges $E' \subseteq E$ and set of vertices $V' \subseteq V$, we write $E'[V']$ to denote the set of all edges of E' **incident** to V', that is $E'[V'] = \{e \in E' : \eta_i(e) \in V' \text{ for some } i \in \{1,2\}\}$; for a vertex $v \in V$, we write $E'[v]$ as short for $E'[\{v\}]$. A **connected component** of M is a non-empty minimal set of vertices $V' \subseteq V$ so that $\eta(e) \in (V' \times V') \dot\cup ((V \setminus V') \times (V \setminus V'))$ for every $e \in E$. A graph is **connected** if it has only one connected component. Henceforward, whenever we write "minimal" it is with respect to set-containment. An **isomorphism** between graphs $M = (V, E, \eta)$ and $M' = (V', E', \eta')$, noted $M \cong M'$, is a pair of bijections $\nu_V : V \to V'$ and $\nu_E : E \to E'$ such that $\eta(e) = (u, v)$ if and only if $\eta'(\nu_E(e)) = (\nu_V(u), \nu_V(v))$. We will henceforth always work modulo graph isomorphism. Given a set $W \subseteq V$, the **graph induced** by W, written $M|_W$, is $\langle W, \{e \in E : \eta(e) \in W \times W\}, \{e \mapsto \eta(e) : \eta(e) \in W \times W\}\rangle$. Similarly, for $E' \subseteq E$, $M|_{E'} = \langle V, E', \{e \mapsto \eta(e) : e \in E'\}\rangle$. Given an equivalence relation $\sim \subseteq V \times V$, the **quotient graph** $M/\sim$ is defined as (V', E, η') where V' has one vertex $[v]_\sim$ for every $\sim$ equivalence class, and $\eta'(e) = ([v]_\sim, [v']_\sim)$ for every $e \in E$ such that $\eta(e) = (v, v')$.

Bridges

A **bridge** (*a.k.a.* minimal edge separator) of a graph M is a minimal set $B \subseteq E$ of edges (minimal in the sense of set-containment) whose deletion induces an increase on the number of connected components of M. The **bridgewidth** of M, which we note $bw(M)$, is the maximum size of a bridge of M, or 0 if M has no edges. The bridgewidth of a class of graphs $\mathscr{C}$, which we write $bw(\mathscr{C})$, is defined as $\sup_{M \in \mathscr{C}} bw(M)$. If $bw(\mathscr{C}) < \infty$, we say that $\mathscr{C}$ has **bounded bridgewidth**; otherwise, it has **unbounded bridgewidth**.



bridge of size 4

associated $\mathcal{E}_4$ minor

not a bridge (not a separator)

not a bridge (not minimal)

not a bridge (not minimal)

Figure 1 Examples of bridges.

Two-way alternating automata

The upper bound algorithm makes use of automata, we give here our *sui generis* definition of 2AFA: a simplified weaker version of the usual 2AFA which fulfills our needs. A **2-way alternating finite state automaton (2AFA)** is a tuple $\mathcal{A} = \langle \mathbb{A}, Q, I, F, \delta\rangle$ where $\mathbb{A}$ is a finite alphabet of letters; Q is a finite set of states; $I, F \subseteq Q$ are sets of initial and final states respectively; and $\delta : Q \to \mathcal{B}_\vee(\mathcal{B}_\wedge(\{+1,-1\} \times \mathbb{A} \times Q))$ is the transition function, where $\mathcal{B}_\vee(\mathcal{B}_\wedge(X))$ stands for a disjunction of conjunction of elements from X (*e.g.*, "$(+1, a, q) \vee ((+1, a, p) \wedge (-1, b, p)) \vee (-1, a, p)$"). A **non-deterministic two-way finite automaton (2NFA)** is a 2AFA that has no conjunctions "$\wedge$" in δ, that is, $\delta : Q \to \mathcal{B}_\vee(\{+1,-1\} \times \mathbb{A} \times Q)$. If further δ has no "-1" elements (*i.e.*, if $\delta : Q \to \mathcal{B}_\vee(\{+1\} \times \mathbb{A} \times Q)$), it is a **non-deterministic finite automaton (NFA)**. An **run from position** i on a word $w \in \mathbb{A}^*$ (where $0 \leq i \leq |w|$) is a finite non-empty tree whose vertices are labelled by elements from $\{0, \ldots, |w|\} \times Q$ such that the root is labeled by some element from $\{i\} \times I$, and for every vertex x labelled (j, q) there is a disjunct of $\delta(q)$ such that for every conjunct (n, a, p)

thereof we have: (i) $0 \leq j+n \leq |w|$, (ii) $(n,a) \in \{(+1, w[j+1]), (-1, w[j])\}$, and (iii) there is a child of x labelled $(j+n, p)$. If the automaton is an NFA, note that the run has only one leaf (j, q_f), in which case we say that the run starts at position i and *ends* at position j. A run is **accepting** if every leaf is labeled by an element from $\{|w|\} \times F$. In the sequel it will be convenient to use these automata as *unary querying devices* on finite words; that is, for any given word $w \in \mathbb{A}^*$, the evaluation $\mathcal{A}(w)$ of the automaton on the word, outputs the set of positions $\mathcal{A}(w) \subseteq \{0, \dots, |w|\}$ of w from which there is an accepting run. The **language** of $\mathcal{A}$ is the set of all words w such that $0 \in \mathcal{A}(w)$. For a word w, we will write $w \in \mathcal{A}$ to denote that w is in the language recognized by $\mathcal{A}$. For any given $\delta : Q \to \mathcal{B}_\vee(\mathcal{B}_\wedge(\{+1,-1\} \times \mathbb{A} \times Q))$ and $I, F \subseteq Q$ we denote by $\delta[I, F]$ the 2AFA having I and F as initial and final state sets and δ as transition function (the alphabet and statespace being implicit).

Oftentimes 2-way automata are defined to have also "word delimiters", in order to recognize when we are at the leftmost or rightmost position of the word. We do not need this feature in our construction (and its absence simplifies, albeit slightly, some definitions); we therefore prefer to leave the definition as simple as possible.

Graph databases and UC2RPQ

A graph database is an edge-labelled finite graph, where labels come from some fix, finite alphabet $\mathbb{A}$. Formally, a **graph database** over an alphabet $\mathbb{A}$ (henceforth graph db) $\mathcal{G} = \langle V, E, \eta, \lambda \rangle$ is a graph $\langle V, E, \eta \rangle$ equipped with a function $\lambda : E \to \mathbb{A}$.

We work with query languages that can traverse edges in both directions: in the direction of the edge as represented in the graph db (*i.e.*, in the *forward* direction), or in the opposite direction (*i.e.*, in the *reverse* direction). Given a finite alphabet $\mathbb{A}$ we represent the instruction of traversing an a-labelled edge in the forward direction by reading the letter $a \in \mathbb{A}$, and the instruction of traversing an a-labelled edge in the reverse direction by reading a^-. Hence, let $\mathbb{A}^-$ be the set of all letters a^- where $a \in \mathbb{A}$, and let $\mathbb{A}^\pm$ be $\mathbb{A} \dot\cup \mathbb{A}^-$. For every $a \in \mathbb{A}$, let $(a^-)^\neg = a$ and $a^\neg = a^-$. A C2RPQ is the closure under conjunction and existential quantification of 2RPQ queries, which are of the form $L(x,y)$ where L is any regular language over $\mathbb{A}^\pm$ and x, y are free variables ranging over vertices of graph databases. Here, we prefer to define C2RPQ directly in a graph form. Let us first define informally what a C2RPQ is – we will later deal with all boring details. A C2RPQ γ is a graph whose edges are labelled with regular languages over $\mathbb{A}^\pm$, equipped with a vector of "output" vertices. An expansion of a C2RPQ is the result of replacing every edge from x to y labelled by L with a path corresponding to some word $w \in L$, respecting the directions imposed by the alphabet $\mathbb{A}^\pm$. As a result, we obtain a graph db and a vector of vertices. For example if $w = a \cdot b^- \cdot b$, then the path looks like $x \xrightarrow{a} \bullet \xleftarrow{b} \bullet \xrightarrow{b} y$; and if $w = \varepsilon$ then the path is empty, and it forces the collapse of x and y. Some examples are shown in Figure 2. Observe that each expansion is

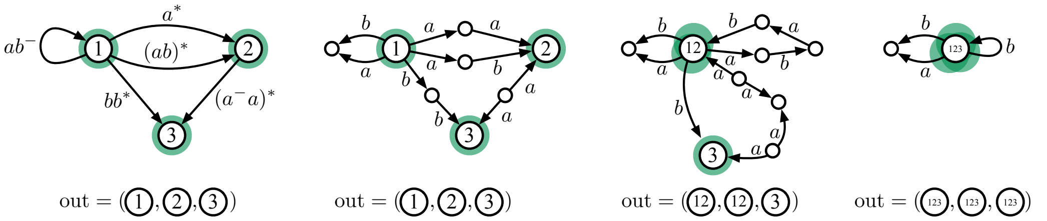


Figure 2 A C2RPQ (left) and three possible expansions. Highlighted vertices are output vertices.

essentially a Conjunctive Query (CQ). A tuple of vertices of a graph db $\mathcal{G}$ is in the output $\gamma(\mathcal{G})$ of a UC2RPQ γ evaluated at $\mathcal{G}$ iff it is in the output $Q(\mathcal{G})$ of the CQ Q corresponding to some expansion of γ.

Formal details follow. A **C2RPQ** over the alphabet $\mathbb{A}$ is represented as

$$\gamma = \langle V, E, \eta, Q, \delta, I, F, \bar{o}\rangle$$

where

(i) $\langle V, E, \eta\rangle$ is a graph,
(ii) Q is a finite set of states,
(iii) $\delta : Q \to \mathcal{B}_\vee(\{+1\} \times \mathbb{A}^\pm \times Q)$ is the transition relation of an NFA over $\mathbb{A}^\pm$,
(iv) $I, F : E \to 2^Q$ are functions indicating the set of initial and final state for each edge, and
(v) $\bar{o}$ is a (possibly null-ary) vector over V, called the **output vector**.

We henceforth write $\bar{\emptyset}$ to denote the null-ary vector. We often write $\gamma(\bar{o})$, whenever $\bar{o} \neq \bar{\emptyset}$, to make explicit the output vector of γ. If $\bar{o} = \bar{\emptyset}$, we say that γ is a **Boolean** C2RPQ. The **arity** of γ is the dimension of $\bar{o}$ (hence, 0 if $\bar{o} = \bar{\emptyset}$). The **underlying graph** of γ is $\langle V, E, \eta\rangle$. We denote by $|\gamma|$ the size of any reasonable representation of γ.

For an equivalence relation $\sim \subseteq V \times V$, the quotient $(\mathcal{G}, \bar{o})/\sim$ of a C2RPQ is defined as for graphs, that is, the result of replacing every vertex v by a representative element $[v]_\sim$ from its equivalence class.

For every word $w \in (\mathbb{A}^\pm)^*$ of length n, let $w[i]$ denote its i-th letter. The graph db associated to w contains $n+1$ distinct vertices $v_0, \dots, v_n$ and n edges $e_1, \dots, e_n$ so that for every i: if $w[i] \in \mathbb{A}$, then $\lambda(e_i) = w[i]$ and $\eta(e_i) = (v_{i-1}, v_i)$; if $w[i] \in \mathbb{A}^-$, then $\lambda(e_i) = (w[i])^\neg$ and $\eta(e_i) = (v_i, v_{i-1})$. We call this graph db the **semipath for the word w that starts in v_0 and ends in v_n**. We will also refer to the ***i*-th vertex of the semipath for w** to denote the vertex v_i. Note that if $w = \varepsilon$ then $v_0 = v_n$, and in this case we say that the semipath is **empty**. By $\overleftrightarrow{\delta}$ we denote transition function of a 2-way NFA having Q as statespace, defined as $q \mapsto \bigvee\{(+1, \alpha, q'), (-1, \alpha^\neg, q') : (+1, \alpha, q') \text{ in } \delta(q)\}$. The intuition is that $\overleftrightarrow{\delta}$ "implements" the notion that by reading a^- we traverse a-edges in the *reverse* direction. More concretely, for a semipath $\mathcal{G}_w$ for $w \in (\mathbb{A}^\pm)^*$, consider $\mathcal{P}_w$ the graph db obtained by adding to $\mathcal{G}_w$ an edge labelled a^- from u to v for every edge labelled a from v to u. Observe then that a run of $\overleftrightarrow{\delta}$ between states p and q on w corresponds to a run of δ between p and q on a directed path of $\mathcal{P}_w$, and vice-versa. A **semipath for an edge** e of a C2RPQ as above is any non-empty semipath for a word $w \in \delta[I(e), F(e)]$ which starts in $\eta_1(e)$ and ends in $\eta_2(e)$ (note that, in particular, $w \neq \varepsilon$ and $\eta_1(e) \neq \eta_2(e)$).

$(\mathcal{X}, \bar{x})$ is an **expansion** of $\gamma(\bar{o})$ (*a.k.a.* a canonical database for $\gamma(\bar{o})$) if $\mathcal{X}$ is a graph db over $\mathbb{A}$, $\bar{x}$ is a vector of vertices of $\mathcal{X}$ of the same arity as $\bar{o}$, and there exists a partition of E into E_0, E_1(*i.e.*, $E = E_0 \mathbin{\dot\cup} E_1$) so that

- for each $e \in E_0$, $I(e) \cap F(e) \neq \emptyset$; and
- for each $e \in E_1$, there is a semipath π_e for e,

and $\mathcal{X}$ is the union of all these semipaths, collapsing all pairs of vertices of E_0. Formally, $(\mathcal{X}, \bar{x})$ is defined as $((\bigcup_{e \in E_1} \pi_e), \bar{o})/\sim$, where $\sim$ is the equivalence relation induced by the connected components of $\mathcal{X}|_{E_0}$.

A **UC2RPQ** is a finite union $\gamma = \gamma_1 \cup \dots \cup \gamma_n$ of C2RPQ with the same arity. The set of expansions of γ is the union of the sets of expansions of the γ_i's.

A vector $\bar{v}$ of vertices of a graph db $\mathcal{G}$ is in the output $\gamma(\mathcal{G})$ of UC2RPQ γ evaluated on $\mathcal{G}$ iff there is some expansion $(\mathcal{X}, \bar{x})$ of γ and a homomorphism $h : \mathcal{X} \to \mathcal{G}$ such that $h(\bar{x}) = \bar{v}$. If γ is Boolean, we say that it is *true* in $\mathcal{G}$ iff $\bar{\emptyset} \in \gamma(\mathcal{G})$, that is, if there exists a homomorphism $\mathcal{X} \to \mathcal{G}$ for some expansion $\mathcal{X}$.

The bridgewidth of a UC2RPQ is the maximum bridgewidth of its underlying graphs, and the bridgewidth of a graph db is the bridgewidth of its underlying graph.

Containment problem for query fragments

Given two UC2RPQ γ_1, γ_2 we say that γ_1 **is contained in** γ_2 (and we note it $\gamma_1 \subseteq \gamma_2$) if γ_1 and γ_2 have the same arity (possibly 0) and $\gamma_1(\mathcal{G}) \subseteq \gamma_2(\mathcal{G})$ for all graph databases $\mathcal{G}$. For any fragment $\mathcal{F}$ of UC2RPQ, the **$\mathcal{F}$-containment problem** is the problem of deciding, given $\gamma_1, \gamma_2 \in \mathcal{F}$, whether $\gamma_1 \subseteq \gamma_2$. Given a class of graphs $\mathscr{C}$, the query class CRPQ($\mathscr{C}$) [resp. C2RPQ($\mathscr{C}$), UC2RPQ($\mathscr{C}$)] is the set of all CRPQ [resp. C2RPQ, UC2RPQ] whose underlying graph is in $\mathscr{C}$.

3 Main result

Let us say that a class of graphs is **non-trivial**, if it contains at least one graph with at least one edge. The main result is the following.

▶ **Theorem 1** (containment dichotomy). *For every non-trivial class $\mathscr{C}$ of graphs,*

1. *if $\mathscr{C}$ has bounded bridgewidth, the UC2RPQ($\mathscr{C}$)-containment problem is* PSpace*-complete;*
2. *if $\mathscr{C}$ has unbounded bridgewidth, the UC2RPQ($\mathscr{C}$)-containment problem is* ExpSpace*-complete.*

Further, the lower bounds hold also for the fragment of Boolean CRPQ($\mathscr{C}$).

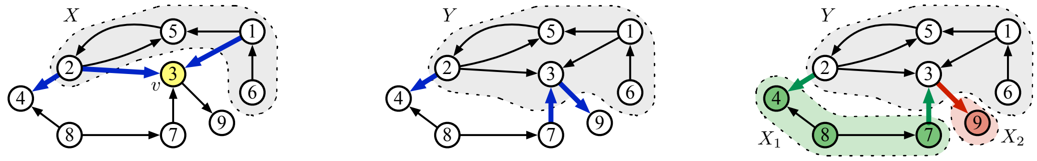
Proofs for lower and upper bounds build upon known results and techniques in the area. Although technical details are somewhat lengthy, they do not bring original ideas other than verifying that bridgewidth is the "right" notion for a dichotomy result. In particular, Barceló *et al.* [6] have already shown the result for the restricted case where $\mathscr{C}$ is any class of "acyclic" graphs (meaning that the only allowed cycles in the underlying undirected simple graphs are self-loops). They use a notion of "*width*" which coincides with bridgewidth on this class of graphs. The lifting of this result to bridgewidth is considerably more involved, but it follows the same philosophy.

4 A bridge maintenance toolkit

We state here some properties of bridges and definitions which shall be of use in the next section.

For a graph $M = \langle V, E, \eta \rangle$, we say that a set of edges $S \subseteq E$ **separates** $X \subseteq V$ and $Y \subseteq V$ if all elements of $X \cup Y$ are in the same connected component of M, and for every $x \in X$ and $y \in Y$, x and y are in different connected components of $M|_{E \setminus S}$. In this case we say that S is a **separator** of X and Y; and if it is minimal with respect to this property, we say that it is a **minimal separator**. Observe that a set of edges is a bridge if and only if it is a minimal separator of two singleton sets $\{x\}, \{y\}$. However, minimal separators need not be bridges in general; for example, the rightmost picture in Figure 1 (page 3) is a minimal separator of {①, ③, ⑧} and {⑥, ⑨, ⑦}.

For a bridge $B \subseteq E$ we say that $X \subseteq V$ is a **side** of B if $X \neq \emptyset$ and there is a connected component Y of $M \setminus B$ such that $X = \{y \in Y : B[y] \neq \emptyset\}$. Note that there are exactly two sides for each bridge, and every bridge separates its sides. For example, the bridge in the leftmost picture of Figure 1 has {⑤, ③, ⑦} and {②, ⑧} as sides. For some set of vertices $Z \subseteq V$ we say **the Z-side of** B to denote the side of B that intersects Z (assuming there is exactly one).



bridge separating X and $V \setminus X$ minimal separator of Y and $V \setminus Y$ separator as union of bridges

Figure 3 Illustration for Lemma 5 (first two pictures) and Lemma 4 (last two pictures).

Bridgewidth can be also understood in terms of graph minors, as we show next. Given an edge $e \in E$ of $M = (V, E, \eta)$ with $\eta(e) = (u, v)$ for some $u \neq v$, an e-**edge contraction** of M is the graph M' obtained by collapsing the endpoints of e. Formally, $M' = (M|_{E \setminus \{e\}})/\sim$ for $\sim$ the finest equivalence relation such that $u \sim v$ (*i.e.*, $\sim = \{(z, z) : z \in V\} \cup \{(u, v), (v, u)\}$). A **minor** of M is any graph M' obtained from M by contracting edges and removing edges and vertices. In particular, minors preserve boundedness of bridgewidth, so do subdivisions and, as a consequence, so do expansions (since graphs corresponding to expansions are subdivisions of minors).

▶ **Lemma 2.**

- *For every graph M and minor M' thereof, $bw(M) \geq bw(M')$.*
- *For every UC2RPQ γ and expansion $\hat{\gamma}$ thereof, $bw(\gamma) \geq bw(\hat{\gamma})$.*

Let us call $\mathcal{E}_k$ the class of graphs containing two distinct nodes u, v and k edges between these nodes. That is, the set of all graphs $M = (\{u, v\}, \{e_1, \dots, e_k\}, \eta)$ for any η such that $\eta(e_i) \in \{(u, v), (v, u)\}$ for every i. Note that $|\mathcal{E}_k| = 2^k$ (actually, there are just $\lceil \frac{n}{2} \rceil$ many graphs up to isomorphism). It is easy to see that the presence of some $\mathcal{E}_k$ minor witnesses a bridge of size at least k and vice-versa (see first two pictures of Figure 1).

▶ **Lemma 3.** *A graph M has bridgewidth at least k if and only if it contains some graph from $\mathcal{E}_k$ as minor.*

▶ **Lemma 4.** *If $S \subseteq E$ is a minimal separator of Y and $V \setminus Y$ in a graph $M = (V, E, \eta)$ such that $M|_Y$ is connected, then there is a partition $\{Y_i\}_{i \in I}$ of $V \setminus Y$ and a pairwise disjoint set of bridges $\{B_i\}_{i \in I}$, computable in polynomial time, so that*

- *$\bigcup_{i \in I} B_i = S$; and*
- *B_i separates Y_i and $V \setminus Y_i$ for every $i \in I$.*

Proof. Let $\{Y_i\}_{i \in I}$ be the partition of $V \setminus Y$ in the connected components of $M|_{V \setminus Y}$. For every i, let $B_i \subseteq S$ be the set of edges between Y and Y_i. It follows that every B_i is a bridge separating Y_i from $V \setminus Y_i$, no edge can belong to two distinct B_i's, and every edge of S is in some B_i. See last two pictures of Figure 3 for an example. ◀

▶ **Lemma 5.** *If $B \subseteq E$ is a bridge of a graph (V, E, η) separating X and $V \setminus X$, and $v \in V \setminus X$ is a vertex incident to B (i.e., so that $B[v] \neq \emptyset$), then there is a (possibly empty) partition $\{X_i\}_{i \in I}$ of $V \setminus (X \cup \{v\})$ and a pairwise disjoint set of bridges $\{B_i\}_{i \in I}$ such that*

- *$\bigcup_{i \in I} B_i = (B \cup E[v]) \setminus B[v]$; and*
- *B_i separates X_i and $V \setminus X_i$ for every $i \in I$.*

Proof. Consider $Y = X \cup \{v\}$ and observe that $M|_Y$ is connected. Then apply the previous Lemma 4 to the minimal separator $S = (B \cup E[v]) \setminus B[v]$ of Y and $V \setminus Y$. See first two pictures of Figure 3. ◀

▶ **Lemma 6.** *For every graph $M = (V, E, \eta)$ there is a vertex $v \in V$ so that $E[v]$ is a bridge.*

Proof. Suppose wlog that M is connected. Observe that if there is a bridge separating X and $V \setminus X$ and $|V \setminus X| = 1$ we are done: the singleton set $V \setminus X$ yields the vertex v to choose. Otherwise, we proceed by induction on the size of $V \setminus X$. Take any bridge B separating X from $V \setminus X$, and take any $v \in V \setminus X$ incident to B. By Lemma 5, $(B \cup E[v]) \setminus B[v]$ is a disjoint union of bridges. Take any such bridge B and observe that it separates X' and $V \setminus X'$ for some $X' \supseteq X \cup \{v\}$. Since $|V \setminus X'| < |V \setminus X|$, we apply inductive hypothesis and we conclude that there must be a vertex in $V \setminus X'$ verifying the property. ◀

▶ **Lemma 7.** *Given a connected graph M and a partition $\tilde{X} \dot\cup X_L \dot\cup X_R$ of the set of vertices therein, and given a bridge B of M separating $\tilde{X}$ and $X_L \dot\cup X_R$; there is a partition $\{X_i\}_{i\in I}$ of $X_L \dot\cup X_R$ and a set of bridges $\{B_i\}_{i\in I}$ of M, computable in polynomial time, such that*

(i) *every B_i separates X_i and $V \setminus X_i$;*

(ii) *for every i there is $Z \in \{X_L, X_R\}$ such that B_i contains only edges between Z and $V \setminus Z$;*

(iii) *for every i, $B_i \cap B[X_L] = \emptyset$ if and only if $B_i \cap B[X_R] \neq \emptyset$;*

(iv) *for every $e \in B$, there is exactly one $i \in I$ with $e \in B_i$.*

Proof. Consider the partition $\{Y_i\}_{i\in J}$ of $X_L \dot\cup X_R$ given by all the connected components of $M|_{X_R}$ and $M|_{X_L}$, and consider the set of sets of edges $\{E_i\}_{i\in J}$ where E_i is the set of edges between Y_i and $V \setminus Y_i$. If follows that $\{Y_i\}_{i\in J}$ and $\{E_i\}_{i\in J}$ satisfy all items above, but some E_i's may not be bridges. We show how to produce a partition into bridges from this initial partition. If there is some Y_i from which there is no edge to $\tilde{X}$ but there are edges to $Y_{i_1}, \dots, Y_{i_\ell}$, then remove all $Y_i, Y_{i_1}, \dots, Y_{i_\ell}$ from the partition and replace them with $Y = Y_i \cup Y_{i_1} \cup \dots \cup Y_{i_\ell}$. Similarly, remove $E_i, E_{i_1}, \dots, E_{i_\ell}$ and add the set of edges between Y and $V \setminus Y$. Note that this results in a strictly coarser partition of $X_L \dot\cup X_R$ which still satisfies all items with respect to its associated set of sets of edges. Repeat this operation until all sets of the partition have at least one edge to $\tilde{X}$. It follows that for each set Z of the partition obtained, the edges between Z and $V \setminus Z$ are a bridge B of M, and that the set of all these bridges verify the conditions with respect to the partition. See Figure 4 for a picture. ◀

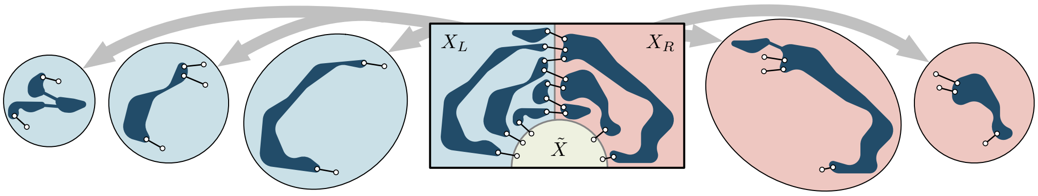


Figure 4 We amalgamate connected components that do not have incident edges with $\tilde{X}$. We end up with a partition whose every element induces a connected subgraph, and a bunch of bridges between partition elements. Each of the ovals shows a bridge and the component that it separates. Left bridges (in blue) use some edge from $B[X_L]$, and right bridges (red) use some edge from $B[X_R]$, but notice that no bridges use both an edge from $B[X_L]$ and an edge from $B[X_R]$.

5 Upper bounds

We show an algorithm to solve the containment problem for UC2RPQ which uses space exponential only in the bridgewidth of the underlying graph. Hence, if the bridgewidth is bounded, the algorithm runs in polynomial space, and otherwise in exponential space.

No self-loops assumption. To simplify some developments, we will assume that the C2RPQs we work with have no self-loops, that is, there are no edges e with $\eta_1(e) = \eta_2(e)$. Any C2RPQ can be transformed into a self-loop-less C2RPQ by adding, for every self-looping edge e on a vertex v a new vertex v_e and edge e', redefining $\eta(e) = (v, v_e)$, $\eta(e') = (v_e, v)$ and $I(e') = F(e') = F(e)$. Note that this is a linear time procedure that does not increase the bridgewidth unless the bridgewidth is 1, in which case it becomes of bridgewidth 2. Hence, this assumption is without loss of generality.

5.1 Proof strategy

We use the same proof strategy as in [6], which we review briefly here:

R1 By a result from [14, Theorem 4], there is a polynomial time reduction of this problem to the containment of a Boolean single-edge C2RPQ γ' into a Boolean UC2RPQ γ, and it is easy to see that this reduction preserves bridgewidth – in fact, $bw(\gamma') = 1$ and the underlying graph of γ is left unaltered in the cited reduction.

R2 We can reduce, in turn, this problem to the non-emptiness of a NFA $\mathcal{A}_{\gamma'\subseteq\gamma}$ of size $O(2^{|\gamma'|+|\gamma|^{c\cdot bw(\gamma)}})$, which can de done in deterministic space $O(|\gamma'| + |\gamma|^{c\cdot bw(\gamma)})$. The NFA runs over the exponential alphabet $\mathbb{A}^{\pm} \dot\cup \text{Loops}$, where $\text{Loops} = 2^{Q\times Q}$ and Q is the statespace of γ. This NFA is obtained from three automata $\mathcal{A}_{cd}, \mathcal{A}_{loop}, \mathcal{A}_\gamma$ as $\mathcal{A}_{\gamma'\subseteq\gamma} \overset{\text{def}}{=} \mathcal{A}_{cd} \cap \mathcal{A}_{loop} \cap \overline{\mathcal{A}_\gamma}$ where:

(i) $\mathcal{A}_{cd}$ is a singly exponential size NFA (with a polynomial number of states) depending on γ', recognizing all words $L_0 a_1 L_1 \cdots a_n L_n \in \text{Loops} \cdot (\mathbb{A}^{\pm} \cdot \text{Loops})^*$ such that $a_1 \cdots a_n \in \delta'[I(e), F(e)]$ for e the sole edge of γ' (*i.e.*, $a_1 \cdots a_n$ represents an expansion of γ').

(ii) $\mathcal{A}_{loop}$ is a singly exponential size NFA depending on γ, recognizing all words of the form $L_0 a_1 L_1 \cdots a_n L_n \in \text{Loops} \cdot (\mathbb{A}^{\pm} \cdot \text{Loops})^*$ such that, for every i, $(q, q') \in L_i$ if and only if there is a 2-way run of $\overleftrightarrow{\delta}[\{q\}, \{q'\}]$ on $a_1 \cdots a_n$ that starts in position i with state q and ends in the same position i with state q'.

(iii) $\mathcal{A}_\gamma$ is a 2AFA of size $O(|\gamma|^{c\cdot bw(\gamma)})$ for some constant c with the property that for every word $w = L_0 a_1 \cdots a_n L_n \in \mathcal{A}_{loop}$, we have that $w \in \mathcal{A}_\gamma$ if and only if γ holds true in $\mathcal{G}_w$, where $\mathcal{G}_w$ is the semipath for $a_1 \cdots a_n$. Remember that this is equivalent to asking whether there is an expansion of γ that can be homomorphically mapped to $\mathcal{G}_w$. $\overline{\mathcal{A}_\gamma}$ is the automaton recognizing the complement language.

Since the complement of a 2AFA can be constructed as a NFA with a single exponential blowup in the statespace [8, Theorem 8], it follows that the resulting NFA $\mathcal{A}_{\gamma'\subseteq\gamma}$ is of size $O(2^{|\gamma'|+|\gamma|^{c\cdot bw(\gamma)}})$. Consequently, it is of single exponential size whenever the bridgewidth is bounded, and thus its emptiness can be checked using polynomial space. We invite the curious reader to read [6, §4.2] for more details on the two reductions **R1** and **R2** above.

The sole contribution of this paper on the proof above lies in the definition of $\mathcal{A}_\gamma$ of item **(iii)**. In [6], $\mathcal{A}_\gamma$ was defined for the case of γ being acyclic, and shown to be exponential in the "width of the acyclic query" (meaning the maximum number of edges between any two distinct vertices), and hence polynomial if the width is bounded. Here we lift this result to *all* queries with respect to the bridgewidth of the query, without assuming any further restriction (such as acyclicity). Bridgewidth is a generalization of their width measure, in the sense that for all acyclic queries, bridgewidth coincides with [6]'s width notion. The price to pay for this generalization is that now the definition of $\mathcal{A}_\gamma$ is considerably more involved. The rest of this section will be devoted to defining $\mathcal{A}_\gamma$ in such a way that it is exponential only in the bridgewidth of γ, and that satisfies the property described in item **(iii)**.

5.2 Definition of $\mathcal{A}_\gamma$

For a UC2RPQ $\gamma = \gamma_1 \cup \cdots \cup \gamma_n$, we define $\mathcal{A}_\gamma$ to be the union $\mathcal{A}_{\gamma_1} \cup \cdots \cup \mathcal{A}_{\gamma_n}$, this is why for simplicity we will henceforth assume that γ is a C2RPQ (*i.e.*, no unions). Let $\gamma = \langle V, E, \eta, Q, \delta, I, F, \bar{\emptyset} \rangle$ (remember, by **R1** γ is Boolean), and let $M = \langle V, E, \eta \rangle$ be the underlying graph of γ. If M is not connected – say γ is equivalent to $\gamma_1 \wedge \cdots \wedge \gamma_n$ for connected C2RPQ's – $\mathcal{A}_\gamma$ is defined as $\mathcal{A}_{\gamma_1} \cap \cdots \cap \mathcal{A}_{\gamma_n}$, which can be done in polynomial time since we are working with *alternating* automata. Therefore, let us also suppose, without loss of generality, that M is connected. For any word $w = L_0 a_1 L_1 \cdots a_n L_n \in \text{Loops} \cdot (\mathbb{A}^{\pm} \cdot \text{Loops})^*$, let $\mathcal{G}_w$ denote the semipath for $a_1 \cdots a_n$.

Let us first refresh what $\mathcal{A}_\gamma$ is supposed to do. Remember that property **(iii)** concerns the case where the input is a word of the form $w = L_0 a_1 L_1 \cdots a_n L_n$ in which each L_i contains the loop information of $\vec{\delta}$ on position i of the word $a_1 \cdots a_n \in (\mathbb{A}^{\pm})^*$. Since γ is Boolean (*i.e.*, a *property* of graph db's), upon reading such a word, $\mathcal{A}_\gamma$ must check whether γ is true on $\mathcal{G}_w$, possibly using the information contained in the labels L_i's. Further, $\mathcal{A}_\gamma$ must use a "small" set of states (polynomial if $bw(\gamma)$ is bounded).

A detour through non-Boolean queries. The definition of $\mathcal{A}_\gamma$ will make use of non-Boolean subqueries of γ. Suppose $B \subseteq E$ is a bridge separating Y from $V \setminus Y$ in M, and let $X = \{v \in Y : B[v] \neq \emptyset\}$ be the Y-side of B. For any given a state assignment $f : B \to Q$ we define the query $\gamma[B, X, f]$ as the result of modifying γ by:

- removing all edges internal to X;
- removing all vertices from $Y \setminus X$ (and the incident edges);
- defining X as output vertices[2]; and
- for every $e \in B$ redefining $I(e)$ [resp. $F(e)$] as $f(e)$ if $\eta_1(e) \in X$ [resp. if $\eta_2(e) \in X$].

See Figure 5 for an example.

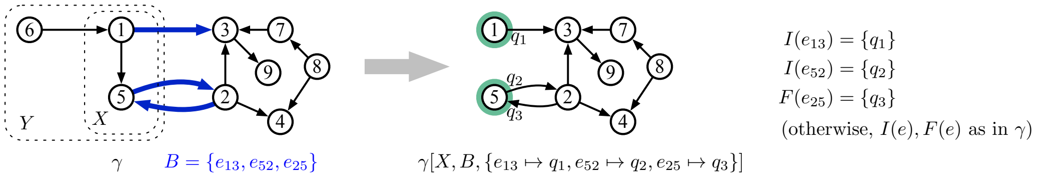

6
1
3
7
9
8
5
2
4
Y
X
γ
$B = \{e_{13}, e_{52}, e_{25}\}$
1
3
7
9
8
5
2
4
q_1
q_2
q_3
$\gamma[X, B, \{e_{13} \mapsto q_1, e_{52} \mapsto q_2, e_{25} \mapsto q_3\}]$
$I(e_{13}) = \{q_1\}$
$I(e_{52}) = \{q_2\}$
$F(e_{25}) = \{q_3\}$
(otherwise, $I(e), F(e)$ as in γ)

Figure 5 Highlighted vertices are output vertices and edges e adorned with a state q means that $I(e)$ is replaced with $\{q\}$ if the edge is outgoing from an output vertex, and that $F(e)$ is replaced with $\{q\}$ if e is incoming to an output vertex.

How many distinct $\gamma[B, X, f]$*'s are there?* The number of such queries is bounded by $(|Q|+1)^{bw(\gamma)} \cdot 2 \cdot |E|^{bw(\gamma)}$, hence a polynomial number if the bridgewidth is bounded by a constant.

[2] That is, defining any vector $(v_1, \dots, v_n)$ as output if $X = \{v_1, \dots, v_n\}$, the order is inconsequential.

For every such non-Boolean $\gamma[B,X,f]$ we define a 2AFA automaton $\mathcal{A}_{\gamma[B,X,f]}$ with the property that, for every $0 \leq i \leq n$ and $w \in \mathcal{A}_{loop}$,

$$2i \in \mathcal{A}_{\gamma[B,X,f]}(w) \quad \text{if and only if} \quad \underbrace{(v_i, \dots, v_i)}_{|X| \text{ times}} \in \gamma[B,X,f](\mathcal{G}_w), \tag{†}$$

where v_i is the i-th vertex of $\mathcal{G}_w$. That is, the automaton $\mathcal{A}_{\gamma[B,X,f]}$ checks whether there is an expansion of $\gamma[B,X,f]$ that can map to $\mathcal{G}_w$ via a homomorphism that assigns the i-th vertex of $\mathcal{G}_w$ to every vertex of X.

Once we know how to define the $\mathcal{A}_{\gamma[B,X,f]}$'s, the definition of $\mathcal{A}_\gamma$ follows easily: Choose any vertex v of γ such that $E[v]$ is a bridge (it exists due to Lemma 6), and guess a function $f : E[v] \to Q$ such that $f(e) \in I(e)$ if $\eta_1(e) = v$, and $f(e) \in F(e)$ if $\eta_2(e) = v$. Then, move non-deterministically to an even position, and run $\mathcal{A}_{\gamma[E[x],\{x\},f]}$. Note that, assuming $\mathcal{A}_{\gamma[E[x],\{x\},f]}$ satisfies (†), property **(iii)** holds on a word w if and only if there is an even position $2i$ of w and a function f with the aforementioned properties such that $2i \in \mathcal{A}_{\gamma[E[x],\{x\},f]}(w)$.

Why do we want to define $\mathcal{A}_\gamma$ in terms of $\mathcal{A}_{\gamma[B,X,f]}$? Because in this way $\mathcal{A}_{\gamma[B,X,f]}$ can be defined in an recursive way: each $\mathcal{A}_{\gamma[B,X,f]}$ is defined using other $\mathcal{A}_{\gamma[B',X',f']}$'s; which is arguably simpler to define and understand, and it has an explicit invariant.

5.3 Definition of $\mathcal{A}_{\gamma[B,X,f]}$

We will show how to construct $\mathcal{A}_{\gamma[B,X,f]}$ satisfying property (†) by possibly "calling", as subroutines, other automata $\mathcal{A}_{\gamma[B',X',f']}$'s. Of course, we adopt this way of defining $\mathcal{A}_{\gamma[B,X,f]}$ just to simplify the description – the formal definition of $\mathcal{A}_{\gamma[B,X,f]}$ will contain one separate statespace for each distinct $\mathcal{A}_{\gamma[B',X',f']}$.

As an example, suppose we have $\gamma[B,X,f]$ with γ is as in Figure 5 (left picture), but with $X = \{⑤\}$ and $B = E[X]$ (also depicted in Figure 8-a). And suppose we have a semipath $\mathcal{G}_w$ for a word $a_1 \cdots a_n \in (\mathbb{A}^{\pm})^*$. What does a mapping from an expansion of γ to $\mathcal{G}_w$ look like? For example, if the word is $abb^- aab^- a^- bb$ and the expansion is obtained by choosing words as in the left of Figure 6, we could obtain a mapping as shown on the right. Note that the automaton has to somehow guess both the expansion and the mapping. Of course, already guessing the order of appearance of the homomorphic images of the vertices of γ (in our case, from left to right: ⑥, ⑦, {⑧, ①}, {⑨, ⑤}, ③, {④, ②}) would already yield an exponential automaton, regardless of bounded bridgewidth, and we therefore need to avoid this kind of brute-force guessing. Nevertheless, the automaton can first guess the non-output vertices that will appear to the left and to the right, as in Figure 7, and then, based on this guessing, find a way of decomposing the query into other, simpler, queries (as shown to the right). While there are exponentially many ways of guessing, the statespace remains polynomial if the bridgewidth is bounded. In the concrete case of the query of Figure 7, the automaton

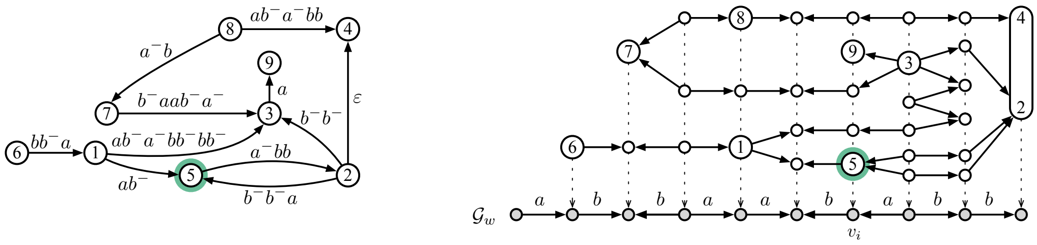


Figure 6 An expansion of $\gamma[B,X,f]$ and its homomorphic mapping into $\mathcal{G}_w$.

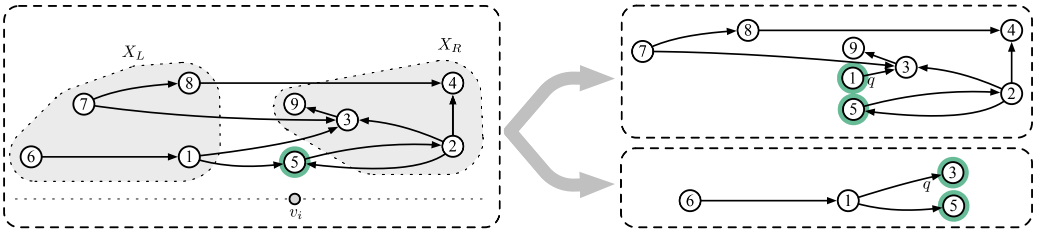


Figure 7 The automaton guesses the vertices mapped to the left (X_L) and to the right (X_R) of the current position v_i as well as a state q, and it decomposes the query into a conjunction of the two smaller queries on the right.

decomposes it by guessing a state $q \in Q$ (intuitively, the state of the automaton for the edge ① → ③ at the position where ⑤ is mapped) and running the conjunction of the two smaller queries on the right. There may be many ways of decomposing them, but not too many, since the simpler queries will still be of the form $\gamma[B, X, f]$.

Suppose that $\mathcal{A}_{\gamma[B,X,f]}$ guesses a set of vertices X_R that will be mapped to the right of v_i, and a set of vertices X_L that will be mapped to the left[3] of v_i (*i.e.*, the gray blobs of Figure 7). For any such given partition X_L, X, X_R of the vertices of $\gamma[B, X, f]$, $\mathcal{A}_{\gamma[B,X,f]}$ proceeds in different ways according to whether some of these sets are empty. In particular, if both X_L and X_R are non-empty, it will need to guess what is the state of some edges that “fly” across the current position, that is, edges with one endpoint in X_L and one endpoint in X_R (in Figure 7, the edge from ① to ③). This is indeed an exponential operation (and hence the construction may take exponential time) but it will not be reflected in the statespace of the automaton if $bw(\gamma)$ is bounded, nor in the space needed to perform this operation.

Now more concretely. Once X_L, X_R are fixed, applying the decomposition of Lemma 7 on the bridge B of M with $\tilde{X} = V \setminus (X_L \cup X_R)$, X_L and X_R, we obtain a set of bridges $\{B_i\}_i$ and a partition $\{X_i\}_i$ of $X_L \cup X_R$. We divide these bridges into left and right bridges: $\mathcal{B}_L = \{B_i : B_i \cap B[X_L] \neq \emptyset\}$ and $\mathcal{B}_R = \{B_i : B_i \cap B[X_R] \neq \emptyset\}$. We guess any function $g : \bigcup(\mathcal{B}_L \cup \mathcal{B}_R) \to Q$ so that $g(e) = f(e)$ for every $e \in B$. For each left-bridge $\hat{B} \in \mathcal{B}_L$ we consider $\gamma[\hat{B}, X^L_{\hat{B}}, g_{\hat{B}}]$ where $X^L_{\hat{B}}$ is the $(X \cup X_R)$-side of $\hat{B}$, and $g_{\hat{B}}$ is g restricted to $\hat{B}$. Similarly, for each right-bridge $\hat{B} \in \mathcal{B}_R$ we consider $\gamma[\hat{B}, X^R_{\hat{B}}, g_{\hat{B}}]$ where $X^R_{\hat{B}}$ is the $(X \cup X_L)$-side of $\hat{B}$, and $g_{\hat{B}}$ is g restricted to $\hat{B}$. Let Z be the $(V \setminus X)$-side of B (*i.e.*, the side which is not X). Note that if $X_L \cap Z = \emptyset$, then $\mathcal{B}_L = \emptyset$ and $\mathcal{B}_R = \{B\}$; and similarly for $X_R \cap Z = \emptyset$.

Thus, if we guess X_L, X_R as in our example of Figure 7, and if g is guessed so that for the edge e_{13} from ① to ③ we have $g(e_{13}) = q$, we generate the two queries on the right picture. Note that there is an expansion and homomorphism in accordance with the guessing X_L, X_R, g and sending ⑤ to v_i if and only if there are homomorphisms from some expansions of the two smaller queries on the right, mapping ① and ⑤ [resp. ③ and ⑤] to v_i. This describes the idea of the most interesting case: how to cover homomorphisms that map some vertices of Z to the left of v_i and some to the right of v_i (*i.e.*, $\mathcal{B}_L \neq \emptyset$ and $\mathcal{B}_R \neq \emptyset$). There are, however, two other remaining cases that need to be treated as well: (1) homomorphisms that map every vertex of Z to the right of v_i (*i.e.*, $Z \subseteq X_R$), which corresponds to simply moving to the “next position” $2(i + 1)$ of w (representing v_{i+1} in $\mathcal{G}_w$) updating the function

[3] The vertices not in X that are mapped *at the same position* as those in X could be placed either in X_L or X_R (*e.g.*, in the picture ⑨ is in X_R).

f accordingly; (2) symmetrically, when all vertices of Z are mapped to the left of v_i; and (3) homomorphisms where at least one vertex of Z is mapped to the current vertex v_i, in which case the query must also be updated. We now describe all these cases formally.

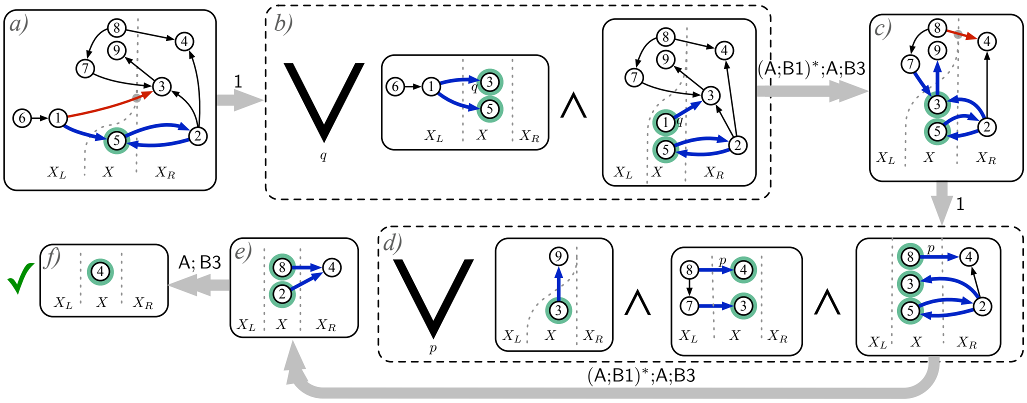


Figure 8 Schematic idea of a branch of an accepting run of $\mathcal{A}_{\gamma[B,X,f]}$ as witnessed by the homomorphism of Figure 6, and how the decomposition is done in terms of the $\gamma[B',X',f']$'s. Arrows are labelled by the type of transitions according to the cases **1**, **2**, **A**, **B1**, **B2**, **B3**; double-arrows stand for a succession of (non-alternating) transitions being applied.

1 $|\mathcal{B}_L \cup \mathcal{B}_R| > 1$. This is the case in our example. In this case, using alternation the automaton verifies that $2i \in (\bigcap_{B\in\mathcal{B}_L} \mathcal{A}_{\gamma[B,X_B^L,g_B]} \cap \bigcap_{B\in\mathcal{B}_R} \mathcal{A}_{\gamma[B,X_B^R,g_B]})(w)$. For example, this is the first kind of transition in our example of Figure 8, both in the transitions $a \to b$ and $c \to d$.

2 $|\mathcal{B}_L \cup \mathcal{B}_R| = 1$. In this case we have necessarily that either $X_L \cap Z$ or $X_R \cap Z$ is empty. Further, $\mathcal{B}_L \cup \mathcal{B}_R = \{B\}$. The automaton proceeds as follows: first it reads the loop information updating the states (case **A** below) and then it either moves right or left to the $2(i+1)$ or $2(i-1)$ position updating the states in f (cases **B1**, **B2**), or it guesses that there is a vertex that is mapped to vertex v_i of $\mathcal{G}_w$ and produces a new decomposition into bridges generating the alternation of smaller queries (**B3**). Here are the details:

A The automaton first reads L_i to the right (*i.e.*, the letter at position $2i$ in w) updating, nondeterministically, the states of f according to the loop type; that is, we replace f with f' for any $f' : B \to Q$ satisfying $(f(e), f'(e)) \in L_i$ for every $e \in B$. Then, it goes back left to the original position $2i$.

B Then it performs one of the three following actions, non-deterministically chosen:

B1 It checks $Z \cap X_R \neq \emptyset$ and moves right to position $2(i+1)$. That is, it reads L_i and then a_{i+1} to the right to end up at position $2(i+1)$ of w. It now updates the states of f' according to the label a_{i+1}; that is, it guesses some f'' where for every $e \in B$ we have $f''(e) = q$ for some $(+1, a_{i+1}, q)$ in $\overleftrightarrow{\delta}(f'(e))$. It finally verifies $2(i+1) \in \mathcal{A}_{\gamma[B,X,f'']}(w)$. For example, this is one of the actions implicit in the arrow $b \to c$ of Figure 8.

B2 It checks $Z \cap X_L \neq \emptyset$ and moves left to position $2(i-1)$. That is, it reads a_i and then L_{i-1} to the left to end up in position $2(i-1)$ of w. It now updates the states of f' accordingly; that is, it guesses some f'' where for every $e \in B$ we have $f''(e) = q$ for some $(-1, a_i, q)$ in $\overleftrightarrow{\delta}(f'(e))$. It finally verifies $2(i-1) \in \mathcal{A}_{\gamma[B,X,f'']}(w)$.

B3 It guesses that there is a vertex v in the $(X_L \cup X_R)$-side of B that will be mapped to position i. That is, it non-deterministically chooses some $v \in X_L \cup X_R$ and verifies $I(e) \cap F(e) \neq \emptyset$ for every edge $e \in B[v]$. Consider now the separator $S =$

$(B \cup E[v]) \setminus B[v]$. The automaton chooses, non-deterministically, a state assignment for all new edges $S \setminus B$. That is, it picks some assignment $g : S \to Q$ so that: $g(e) = f(e)$ for every $e \in B$; $g(e) \in I(e)$ if $\eta_1(e) = v$; and $g(e) \in F(e)$ if $\eta_2(e) = v$. Now, using Lemma 5, S can be decomposed into a disjoint union of bridges $S = \bigcup_{i \in I} B_i$ so that each bridge will cover an independent part of the remaining graph of $\gamma[B, X, f']$. Thus, using alternation, the automaton finally verifies $2i \in (\bigcap_{i \in I} \mathcal{A}_{\gamma[B_i, X_{B_i}, g_{B_i}]})(w)$, where X_{B_i} is the $(X \cup \{v\})$-side of B_i, and g_{B_i} is g restricted to B_i (in Figure 8, this case corresponds to transitions $b{\to}c$ and $d{\to}e$). In particular, if $I = \emptyset$, the automaton simply accepts the word (in Figure 8, transition $e{\to}f$).

Figure 8 contains an example of how these cases interact in a run. Summing up, the automaton $\mathcal{A}_{\gamma[B,X,f]}$ can be implemented using 4 states $q_0^{B',X',f'}, q_{\textsf{A1}}^{B',X',f'}, q_{\textsf{A2}}^{B',X',f'}, q_{\textsf{B}}^{B',X',f'}$ for each possible $\gamma[B', X', f']$, plus one global final state $F = \{q_f\}$. The initial set of states is $I = \{q_0^{B,X,f}\}$. For every possible $\mathcal{B}_L$, $\mathcal{B}_R$, if condition **1** holds, it uses alternation on states $q_0^{B',X',f'}$ for suitable B', X', f'. Otherwise, it uses: states $q_{\textsf{A1}}^{B,X,f'}, q_{\textsf{A2}}^{B,X,f'}$ to perform the transitions described in **A** (*i.e.*, move to $2i + 1$ with state $q_{\textsf{A1}}^{B,X,f'}$ for some f', and then back to $2i$ with state $q_{\textsf{A2}}^{B,X,f'}$); states $q_{\textsf{B}}^{B,X,f'}, q_0^{B,X,f''}$ to move finally to position $2(i + 1)$ or $2(i - 1)$ as in **B1** or **B2** with the updated f''; and states $q_0^{B',X',f'}$'s to perform **B3** if possible. Whenever it accepts, it shifts to state q_f, from which it moves to the rightmost position to accept the word.

5.4 Correctness

For any subquery $\gamma[B, X, f]$ we prove, by induction on the size of $\gamma[B, X, f]$, that the automaton $\mathcal{A}_{\gamma[B,X,f]}$ satisfies (†). Suppose B separates $\tilde{X}$ and $V \setminus \tilde{X}$ in M, for $X \subseteq \tilde{X}$.

The *base case* is when $|V \setminus \tilde{X}| = 1$. Note that in this case, by construction, $\mathcal{A}_{\gamma[B,X,f]}$ works as a non-deterministic two-way automaton (that is, there is no alternation).
$\Rightarrow$) Suppose $2i \in \mathcal{A}_{\gamma[B,X,f]}(w)$. An accepting run consists basically on a number of applications of: the pair of back-and-forth transitions described in **A**, plus a transition from **B1** or **B2**, until by **B3** the automaton accepts. Assuming the word w is correctly labeled (*i.e.*, $w \in \mathcal{A}_{loop}$), such a run induces a homomorphism from an expansion of $\gamma[B, X, f]$ (given by the letters and loops read along) into $\mathcal{G}_w$ that maps every $x \in X$ to v_i. Notice that the last step in the accepting run is a collapse of the vertex of $V \setminus \tilde{X}$ by condition **B3** where $I = \emptyset$.
$\Leftarrow$) On the other hand, if there is an expansion of $\gamma[B, X, f]$ with a homomorphism mapping every $x \in X$ to v_i, then the sole vertex of $V \setminus \tilde{X}$ is mapped either to the right or to the left of i. In the former case the automaton can perform a number of **A**,**B2** steps until reaching the desired position and finishing with an application of **A**,**B3** steps accepting, and in the latter case it performs **A**,**B1** a number of times and then accepting with **A**,**B3**. Observe that all the loops can be "factored away" by using the information in the L_i's. Hence, if there is such a homomorphism, the automaton has an accepting run from position $2i$ on w.

The *inductive case* is when $|V \setminus \tilde{X}| > 1$.
$\Rightarrow$) Suppose $2i \in \mathcal{A}_{\gamma[B,X,f]}(w)$. If the first transition of the accepting run comes from **1**, then by the inductive hypothesis we obtain $(v_i, \ldots, v_i) \in \gamma[B, X, f](\mathcal{G}_w)$ using the following claim.

▷ Claim. If there are expansions of $\gamma[B, X_B^L, g_B]$ and $\gamma[B, X_B^R, g_B]$ with homomorphisms to $\mathcal{G}_w$ mapping every vertex of $\bigcup_{B \in \mathcal{B}_L \cup \mathcal{B}_R} X_B^L \cup X_B^R$ to v_i, then there is an expansion of $\gamma[B, X, g]$ mapping every vertex of X to v_i. In fact, the desired homomorphism and expansion is simply obtained as the union of the homomorphisms and expansions. This is because Lemma 7 partitions the graph of $\gamma[B, X, f]$ using pairwise disjoint sets of bridges $\mathcal{B}_L$ and $\mathcal{B}_R$.

Otherwise, the accepting run of the automaton consists of a number of applications of steps **A**,**B1** or **A**,**B2**, followed by the application of steps **A**,**B3** (this last one using alternation). Again, since the alternation is performed on disjoint sets of bridges and graph connected components by Lemma 5, once we have expansions and homomorphisms for these by inductive hypothesis, we can build an expansion of $\gamma[B, X, f]$ using the information on the applications of **A**,**B1** or **A**,**B2** that preceded this last **B3** step.
$\Leftarrow$) If there is an expansion of $\gamma[B, X, f]$ with a homomorphism h mapping every $x \in X$ to v_i, then the accepting run will depend on how the vertices are mapped to $\mathcal{G}_w$. If the $(V \setminus X)$-side of B has vertices that map through h both to the left and to the right of v_i in $\mathcal{G}_w$, then we follow **1** and we decompose into the conjunction of some $\gamma[B', X', f']$'s as directed by Lemma 7. Again, since the existence of h implies the existence of homomorphisms from expansions of the simpler $\gamma[B', X', f']$'s, it follows, by inductive hypothesis, that there are accepting runs for each $\mathcal{A}_{\gamma[B',X',f']}$ starting in $2i$, which in turns means that there is an accepting run of $\mathcal{A}_{\gamma[B,X,f]}$ starting in $2i$. If, on the other hand, all vertices from the $(V \setminus X)$-side of B are mapped through h to the right of v_i (or all to the left) in $\mathcal{G}_w$, we follow the strategy to either go right (or left) by repeated applications of the transitions **A**,**B1** (or **A**,**B2**) until we arrive at some position $2(i + \ell)$ (or $2(i - \ell)$) to which some other vertex is mapped, and by **A**,**B3** we use alternation on some $\gamma[B', X', f']$ as directed by Lemma 5. Similarly as before, the homomorphism h and expansion implies the existence of homomorphisms from expansions of the $\gamma[B', X', f']$'s from $2(i + \ell)$, which in turns means that there is an accepting run from $2i$ for $\mathcal{A}_{\gamma[B,X,f]}$.

6 Lower bounds

The lower bounds of Theorem 1 are straightforward from known results. We give the proof ideas here.

The PSPACE-hardness of point 1 of Theorem 1 is a consequence of a straightforward reduction from the containment problem for regular languages. Given two regular languages L_1, L_2 over an alphabet $\mathbb{A}$, given any two distinct symbols $\#, \perp$ not in $\mathbb{A}$, and given a graph $M \in \mathscr{C}$ with some edge e between two vertices u_1, u_2, consider the Boolean queries $\gamma_1, \gamma_2 \in$ CRPQ($\mathscr{C}$) whose underlying graph is M over the alphabet $\mathbb{A} \dot\cup \{\perp, \#\}$, where γ_i has the language $\# \cdot L_i \cdot \#$ at edge e and the language $\{\perp\}$ at every other edge. It follows that γ_1 is contained in γ_2 if and only if $L_1 \subseteq L_2$.

The EXPSPACE-hardness of point 2 follows from a reduction from the following containment problem restricted to two-vertex Boolean CRPQ, which is EXPSPACE-hard.

▶ **Lemma 8.** *The problem of deciding, given a Boolean CRPQ γ_1 with two vertices and one edge and another Boolean CRPQ γ_2 with two vertices and arbitrarily many edges in the same direction, whether γ_1 is contained in γ_2, is* EXPSPACE*-hard.*

We show this lemma by an easy adaptation of the proofs of [7, Theorem 6] and [5, Lemma 14].

Proof. We reduce from the following 2^n-tiling problem, which is EXPSPACE-complete (see, *e.g.*, [10, Theorem 6.1]). An input instance consists of a number $n \in \mathbb{N}$ written in unary, a finite set Δ of tiles, two relations $H, V \subseteq \Delta \times \Delta$ specifying constraints on how tiles should be placed horizontally and vertically, and the starting and final tiles $t_S, t_F \in \Delta$. A solution to the input instance is a "consistent" assignment of tiles to a finite rectangle having 2^n columns. Concretely, a solution is a function $f : \{1, \dots, 2^n\} \times \{1, \dots, k\} \to \Delta$, for some $k \in \mathbb{N}$, such that $f(1, 1) = t_S$, $f(2^n, k) = t_F$, and $f((i, j), f(i + 1, j)) \in H$ and $f((i, j), f(i, j + 1)) \in V$ for every i, j in range. We now show the following.

▷ Claim 9. For every 2^n-tiling problem T there are Boolean CRPQ γ_1, γ_2, computable in polynomial time from T, such that $\gamma_1 \subseteq \gamma_2$ if, and only if, T has no solution. Further, γ_1 is of the form $\exists x, y\ x \xrightarrow{L} y$ and γ_2 is of the form $\exists x, y\ \bigwedge_{0 \le i \le n}(x \xrightarrow{L_i} y)$, where each L_i is given as a regular expression.

For any tiling instance as above, we show how to define the two CRPQ over the alphabet $\mathbb{A} := \Delta \cup \{0, 1, \#\}$ so that containment fails if and only if there is a solution to T. We will encode a solution of a tiling as a word of $\#((0+1)^n \cdot \Delta)^*\#$, where the rectangle of tiles is read left-to-right and top-to-bottom, and each block (*i.e.*, each element of $(0+1)^n\ \Delta$) represents the column number (in binary) and the tile. The symbols # at the beginning and end of the word are used for technical reasons.

For enforcing this encoding, we define regular languages E, F_C, F_H and G_i for each $i \le n$ over $\mathbb{A}$.

The language E gives the general shape of the encoding of solutions,

$$E = \#\, 0^n\ t_S\ ((0+1)^n \Delta)^*\ 1^n\ t_F\, \#,$$

in particular it enforces that it starts and ends with the correct tiles. The language F_C detects adjacent blocks with an error in the column number bit, which can be easily defined with a polynomial size NFA (*e.g.*, if $n = 3$ in particular F_C contains every word having "$001\,t\,011$", "0000" or "$t\,t'$" as factor, for $t, t' \in \Delta$). The language F_H checks that there are adjacent blocks in which the tiles do not respect the horizontal adjacency relation H,

$$F_H = \bigcup_{(t_1,t_2) \in \Delta^2 \setminus H} t_1\ \overline{0^n}\ t_2,$$

where $\overline{0^n} = (0+1)^n \setminus \{0^n\}$. Finally, $G_0, \dots, G_n$ are used to check that there are two blocks at distance 2^n which do not respect the vertical adjacency relation V; in other words, there is a factor of the word whose first and last blocks have the same column number, it contains not more than one block with column number 1^n (otherwise we would be skipping a row), and its first and last tiles are not V-related. First, G_0 checks that the first and last blocks of the factor we are interested in do not conform to V, and furthermore that there is exactly one column number 1^n in between

$$G_0 = \bigcup_{(t_1,t_2) \in \Delta^2 \setminus V} (0+1)^n\, t_1\, (\overline{1^n}\,\Delta)^*\ 1^n\ (\Delta\,\overline{1^n})^*\, t_2,$$

where $\overline{1^n} = (0+1)^n \setminus \{1^n\}$. For each $b \in \{0, 1\}$ and $i \in \{1, \dots, n\}$ we define G_i^b to check that the i-th bit of the address of both the first and last tile is set to b,

$$G_i^b = (0+1)^{i-1}\, b\, (0+1)^{n-i}\, \Delta\, \left((0+1)^n\, \Delta\right)^*\ (0+1)^{i-1}\, b\, (0+1)^{n-i}\, \Delta,$$

and we define G_i as $G_i^0 + G_i^1$. For each one of these languages one can produce a regular expression recognizing the language in polynomial time. Finally, the Boolean CRPQ are

$$\gamma_1 = \exists x, y\ \ x \xrightarrow{E} y \qquad\qquad \gamma_2 = \exists x, y \bigwedge_{0 \le i \le n} x \xrightarrow{G_i \cup F_C \cup F_H} y.$$

Observe that $\gamma_1 \subseteq \gamma_2$ iff every word of E contains an error in the encoding (*i.e.*, it contains a word from F_C as factor), or it contains a pair of horizontally adjacent tiles which do not respect the horizontal constraints (*i.e.*, it contains a word from F_H as factor) or, otherwise, it contains a pair of vertically adjacent tiles which do not respect the vertical constraints (*i.e.*, it contains a word from $\bigcap_{0 \le i \le n} G_i$ as factor). Thus, $\gamma_1 \subseteq \gamma_2$ if, and only if, T has no solution. ◀

Now, in view of Lemma 8, suppose γ_1 reads language $L \subseteq \mathbb{A}^*$ at its only edge, and γ_2 has ℓ edges with languages $L_1, \dots, L_\ell \subseteq \mathbb{A}^*$. By the characterization of Lemma 3, it follows that for every k, $\mathscr{C}$ contains a graph M with a minor from $\mathcal{E}_k$. Hence, there is some graph $M \in \mathscr{C}$ and two sets of vertices V_1, V_2 therein so that $M|_{V_1}$ and $M|_{V_2}$ are connected and M has ℓ distinct edges $e_1, \dots, e_\ell$ with source in V_1 and target in V_2. Consider the following two Boolean CRPQ($\mathscr{C}$) γ'_1, γ'_2 having M as underlying graph over the alphabet $\mathbb{A} \dot\cup \{\bot\}$, where $\bot$ is some symbol not in $\mathbb{A}$. First, γ'_1 is defined as follows: every edge in $M|_{V_1}$ or $M|_{V_2}$ is labelled with $\{\varepsilon\}$; every edge e_i is labelled with L; and every other edge is labelled with $\{\bot\}$. Second, γ'_2 is defined as follows: every edge in $M|_{V_1}$ or $M|_{V_2}$ is labelled with $\{\varepsilon\}$; every edge e_i is labelled with L_i; and every other edge is labelled with $(\mathbb{A} \cup \{\bot\})^*$. It is not hard to see that $\gamma'_1 \subseteq \gamma'_2$ if and only if $\gamma_1 \subseteq \gamma_2$. See Figure 9 for an example. It then follows that containment of Boolean CRPQ($\mathscr{C}$) is ExpSpace-hard.

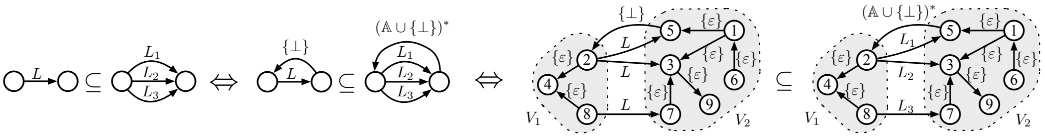


Figure 9 Reduction from two vertices Boolean CRPQ containment (left) to Boolean CRPQ($\mathscr{C}$) containment (right).

7 Final remarks

Observe that the following graph measures are all lower bounds for bridgewidth: maximum vertex degree, maximum number of pairwise edge-independent paths between two vertices (and hence also the size of a minimum cut set by Menger's theorem), and the graph's *treewidth*.[4] In particular, bounded bridgewidth implies bounded treewidth, and thus classes of UC2RPQ of bounded bridgewidth can be evaluated in polynomial time. Classes of UC2RPQ of bounded bridgewidth are somewhat more robust than those of bounded treewidth, in the sense that both the evaluation and containment problems are "efficient". This is what happens for classes of Conjunctive Queries of bounded treewidth, where both these problems – which in this case are essentially the same – are polynomial-time computable [13, 12]. On the other hand, bounded treewidth does not imply bounded bridgewidth (take for instance $\bigcup_i \mathcal{E}_i$). As for treewidth [3], the problem of whether a graph has bridgewidth k (where both the graph and k are input) is NP-complete, by a simple reduction from the MAX-CUT problem (see, *e.g.*, [1]).

What about boundedness? We conjecture that an adaptation of the bridgewidth measure may also yield a similar result for the boundedness problem for UCRPQ (note: no 2-wayness). More concretely, let the **scc-minor** of a graph M be the result of contracting all edges belonging to the same strongly connected component (scc). Note that the resulting graph is a directed acyclic graph (possibly with self-loops). The **scc-bridgewidth** of M is the bridgewidth of the scc-minor of M.

[4] For treewidth, the root of the tree decomposition is a bag containing a vertex whose set of incident edges is a bridge (it exists due to Lemma 6), together with all its neighbors. We then apply iteratively Lemma 4 to decompose the remaining graph into subtrees.

▶ **Conjecture.** *For every class $\mathscr{C}$ of graphs,*

1. *if $\mathscr{C}$ has bounded scc-bridgewidth, the UCRPQ($\mathscr{C}$)-boundedness problem is in* PSPACE*;*
2. *if $\mathscr{C}$ has unbounded scc-bridgewidth, the UCRPQ($\mathscr{C}$)-boundedness problem is* EXPSPACE-*complete.*

References

1 Max cut problem between two connected subgraphs. Stack Exchange. Version: 2018-07-26 21:13. URL: `https://cstheory.stackexchange.com/q/41267/49964`.

2 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoč. Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017. `doi:10.1145/3104031`.

3 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

4 Pablo Barceló. Querying graph databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 175–188, 2013. `doi:10.1145/2463664.2465216`.

5 Pablo Barceló, Diego Figueira, and Miguel Romero. Boundedness of Conjunctive Regular Path Queries. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 104:1–104:15. Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.104`.

6 Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. Semantic Acyclicity on Graph Databases. *SIAM J. Comput.*, 45(4):1339–1376, 2016. `doi:10.1137/15M1034714`.

7 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of Conjunctive Regular Path Queries with Inverse. In *Principles of Knowledge Representation and Reasoning (KR)*, pages 176–185, 2000.

8 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-Based Query Answering and Query Containment over Semistructured Data. In *International Symposium on Database Programming Languages (DBPL)*, volume 2397 of *Lecture Notes in Computer Science*, pages 40–61. Springer, 2001. `doi:10.1007/3-540-46093-4_3`.

9 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. *Journal of Computer and System Sciences (JCSS)*, 64(3):443–465, 2002. `doi:10.1006/jcss.2001.1805`.

10 Bogdan S. Chlebus. Domino-Tiling Games. *Journal of Computer and System Sciences (JCSS)*, 32(3):374–392, 1986. `doi:10.1016/0022-0000(86)90036-X`.

11 Daniela Florescu, Alon Levy, and Dan Suciu. Query Containment for Conjunctive Queries with Regular Expressions. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 139–148. ACM Press, 1998. `doi:10.1145/275487.275503`.

12 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1:1–1:24, 2007. `doi:10.1145/1206035.1206036`.

13 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Symposium on Theory of Computing (STOC)*, pages 657–666. ACM Press, 2001. `doi:10.1145/380752.380867`.

14 Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. Regular Queries on Graph Databases. *Theoretical Computer Science*, 61(1):31–83, 2017. `doi:10.1007/s00224-016-9676-2`.

On Equivalence and Cores for Incomplete Databases in Open and Closed Worlds

Henrik Forssell
University of Oslo, Norway
University of South East Norway, Drammen, Norway
jonf@ifi.uio.no

Evgeny Kharlamov
Bosch Center for Artificial Intelligence, Renningen, Germany
University of Oslo, Norway
evgeny.kharlamov@de.bosch.com

Evgenij Thorstensen
University of Oslo, Norway
evgenit@ifi.uio.no

Abstract

Data exchange heavily relies on the notion of incomplete database instances. Several semantics for such instances have been proposed and include open (OWA), closed (CWA), and open-closed (OCWA) world. For all these semantics important questions are: whether one incomplete instance semantically implies another; when two are semantically equivalent; and whether a smaller or smallest semantically equivalent instance exists. For OWA and CWA these questions are fully answered. For several variants of OCWA, however, they remain open. In this work we adress these questions for Closed Powerset semantics and the OCWA semantics of [24]. We define a new OCWA semantics, called OCWA*, in terms of homomorphic covers that subsumes both semantics, and characterize semantic implication and equivalence in terms of such covers. This characterization yields a guess-and-check algorithm to decide equivalence, and shows that the problem is NP-complete. For the minimization problem we show that for several common notions of minimality there is in general no unique minimal equivalent instance for Closed Powerset semantics, and consequently not for the more expressive OCWA* either. However, for Closed Powerset semantics we show that one can find, for any incomplete database, a unique finite set of its subinstances which are subinstances (up to renaming of nulls) of all instances semantically equivalent to the original incomplete one. We study properties of this set, and extend the analysis to OCWA*.

2012 ACM Subject Classification Theory of computation → Incomplete, inconsistent, and uncertain databases; Theory of computation

Keywords and phrases Incomplete Databases, Cores, Semantics, Open and Closed Worlds

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.10

Related Version A full version of the paper is available at `https://arxiv.org/abs/2001.04757`.

Funding This work was partially supported by the SIRIUS Centre, Norwegian Research Council project number 237898 and project number 230525.

1 Introduction

Data Exchange

Data exchange is the problem of translating information structured under a *source schema* into a *target schema*, given a source data set and a set of declarative *schema mappings* between the source and target schemata. This problem has originally been studied for traditional relational databases where a decade of intensive research brought up a number of foundational and system oriented work [2, 5, 7, 21, 26]. More recently research in data exchange changed its focus in various directions that include non-relational [4] and temporal data [13], knowledge bases [3], mapping discovery [27, 28], and probabilistic settings [19, 25].

In relational data exchange, a set of schema mappings M is defined as a set of source-to-target tuple generating dependences [1] of the form $\phi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{z})$, where $\phi(\bar{x}, \bar{y})$ (resp. $\psi(\bar{x}, \bar{z})$) is a query over the source (resp. target) schema with its variables. In general such mappings only partially specify how to populate attributes of the target schema with data from a given *source instance* [8], i.e., due to existential variables $\bar{z}$ in $\psi(\bar{x}, \bar{z})$. Therefore, data exchange can result in possibly multiple incomplete *target instances* A. Each such A represents a set of possible complete target instances and there are several options on how such correspondence, or *semantics of incomplete instances*, can be defined, including Open World (OWA) [8, 9], Closed World (CWA) [18], Open and Closed World (OCWA, with annotated instances) [24], and Powerset Closed World (PCWA) [12], which we discuss in detail in Section 2.

Problems for Data Exchange

In the context of data exchange the following questions have attracted considerable attention: given a semantics for incomplete database instances, decide:

- [**Semantic Implication:**] whether one incomplete instance semantically implies another;
- [**Equivalence:**] whether two incomplete instances are semantically equivalent; and
- [**Minimality, Core:**] whether a smaller or smallest (core) semantically equivalent incomplete instance exists.

These questions form a natural progression, in that a characterization of semantic implication leads to one for equivalence, which in turn allows the study of minimal equivalent instances. The latter is important since, e.g., in some cases one can use the smallest minimal instance for computing certain answers by naively' evaluating queries directly on this instance.

How These Problems Have Been Addressed So Far

These three questions are the focus of this paper since they have only partially been answered. Indeed, for OWA and CWA, these questions have been fully answered. For OWA, semantic implication corresponds to the existence of a database homomorphism from one instance into another, and a unique smallest equivalent instance (the core [9]) always exists, and is minimal for several natural notions of minimality. Likewise, for CWA semantic implication corresponds to the existence of a strongly surjective homomorphism from one instance to another [18]. This implies that equivalence corresponds to isomorphism, rendering the question of smallest equivalent instance moot. For PCWA, semantic implication corresponds to the existence of a *homomorphic cover* from one instance to another [12], while the question of smallest equivalent instance remains open. For OCWA with annotated instances, both questions are open, although preliminary results were previously presented by the authors [11]. Finally, we are not aware that the question of semantic implication between PCWA and OCWA with annotated instances has previously been considered.

Our Approach to Implication and Equivalence

Therefore, in this paper we address the questions of Semantic Implication, Equivalence, and Minimality for PCWA and OCWA semantics. To this end we introduce a novel open-and-closed-world semantics, OCWA*, based purely on the notion of homomorphic cover. We show how both PCWA and OCWA semantics with annotated instances can be defined as special cases of OCWA*. This subsumption property allows us to characterize semantic implication and equivalence for all three semantics using homomorphic covers, and thus also semantic implication and equivalence between PCWA and OCWA with annotated instances.

Our Approach to Minimality and Cores

We study several natural notions of minimality, and show for all of them that there is in general no unique minimal equivalent instance for PCWA nor, consequently, for the more expressive OCWA*. This raises the question: *How can one find a smaller or "better" equivalent instance?* Indeed, even if one can find all equivalent subinstances of a given incomplete instance A and compare them using the characterization of equivalence, one still does not know whether a there exists a smaller equivalent instance that is not a subinstance of A.

We address this challenge as follows. Focusing first on PCWA, we show that for all instances A there exists a finite set $\varpi(A)$ of "PCWA-cores" which serves to determine all minimal instances that are equivalent to A. More precisely, this set has the following properties:

1. each member of $\varpi(A)$ is minimal (for all notions of minimality that we consider in this paper) and a subinstance of A,
2. the union of the members of $\varpi(A)$ is equivalent to A,
3. A and B are equivalent if and only if $\varpi(A) = \varpi(B)$, up to renaming of nulls, and
4. any instance which is equivalent to A and which is minimal in the sense of having no equivalent subinstance must be an image of the union of the members of $\varpi(A)$. In particular, all such instances can be found, up to renaming of nulls, from (the union of) $\varpi(A)$.

We also apply the analysis of naïve evaluation of existential positive queries with Boolean universal guards from [12] and show that such queries can be evaluated on the smaller members in $\varpi(A)$ rather than on A itself. Finally, we extend the analysis to OCWA* and show that, by resolving a question of "redundant annotation", the function $\varpi(A)$ can be extended also to annotated instances, yielding similar properties for OCWA*. In summary, the contributions of this paper are:

- A new semantics OCWA* which properly extends PCWA and OCWA with annotated instances.
- Characterization and analysis of semantic implication and equivalence for PCWA, OCWA with annotated nulls, and OCWA*.
- Negative results for the existence of unique minimal instances in PCWA and OCWA*.
- A new concept of "PCWA-core" for PCWA; and in terms of it,
- a new "powerset canonical representative function" $\varpi(-)$ for PCWA and OCWA*, with the properties listed above.
- An analysis of "annotation redundancy" in OCWA*.

The paper is organised as follows. In Section 2 we give preliminaries and introduce known semantics for incomplete DBs. In Section 3 we present our OCWA* semantics and give its basic properties. In Section 4 we study semantic implication and equivalence for OCWA*. In Section 5 we show the non-existence of a subinstance minimal representative function for PCWA and, consequently, for OCWA*. In Section 6 we move to positive results for PCWA and then extend them in Section 7 for the general case of OCWA*.

2 Preliminaries

We use boldface for lists and tuples; thus $\mathbf{x}$ instead of $\bar{x}$ or $\overrightarrow{x}$. $\mathbb{N}^+$ is the set of positive (non-zero) natural numbers. $\mathcal{P}^+(A)$ is the set of non-empty subsets of A. $\mathcal{P}^{\text{fin}}(A)$ is the set of finite subsets of A. If S is a set of instances then $\overline{S}^{\cup}$ denotes the closure of S under binary unions.

2.1 Incomplete Databases

We assume that we are working with a fixed database schema. Let Const and Null be countable sets of constants and labeled nulls. For the sake of readability, we will use lower case letters late in the alphabet for nulls instead of the more common $\perp$. Lower case letters a, b, c, d will be used for constants. An (incomplete) *instance* A is a database instance whose (active) domain is a subset of $\mathsf{Const} \cup \mathsf{Null}$. A *complete instance* I is an instance without nulls. (This is also known as a *ground* instance.) We write $\mathcal{D}$ for the set of all instances and $\mathcal{C}$ for the set of all complete instances. We use upper case letters A, B, etc. from the beginning of the alphabet for instances in general, and upper case letters I, J, etc. from the middle of the alphabet for instances that are explicitly assumed to be complete.

Following [24] an *annotated instance* is an instance where each *occurrence* of a constant or null is annotated with either o, standing for *open*, or c, standing for *closed*. The added expressivity is used to define more fine-grained semantics for incomplete databases.

2.2 Homomorphisms and Disjoint Unions

We use the terms "homomorphism" and "isomorphism" to mean database homomorphism and database isomorphism, respectively, and we distinguish these from "structure" homomorphisms. Explicitly, if A and B are instances – whether incomplete or complete, annotated or not – a *structure homomorphism* $h : A \to B$ is a function from the active domain of A to the active domain of B such that for every relation symbol R, if a tuple $\mathbf{u}$ is in the relation R in A then the tuple $h(\mathbf{u})$ is in the relation R in B. We write $\mathsf{Str}(A, B)$ for the set of structure homomorphisms from A to B. A *structure isomorphism* is an invertible structure homomorphism.

If $P \subseteq \mathsf{Const} \cup \mathsf{Null}$ and h is a structure homomorphism we say that h *fixes* P *pointwise* if $h(p) = p$ for all $p \in P$ on which h is defined. We say that h *fixes* P *setwise* if it restricts to a bijection on the subset of P on which it is defined.

A *(database) homomorphism* from A to B is a structure homomorphism that fixes Const pointwise. We write $\mathsf{Hom}(A, B)$ for the set of homomorphisms from A to B. A *(database) isomorphism* is an invertible homomorphism.

A *subinstance* of A is an instance B with an inclusion homomorphism $B \hookrightarrow A$ – that is, with a homomorphism that fixes $\mathsf{Const} \cup \mathsf{Null}$ pointwise. B is a *proper* subinstance if $A \neq B$. We shall often be somewhat lax with the notion of a subinstance and regard B as a subinstance if it is so up to renaming of nulls, that is to say, up to (database) isomorphism. If we need to insist that the homomorphism $B \hookrightarrow A$ is an inclusion we say that B is a *strict* subinstance.

If $h : A \to B$ is a structure homomorphism then the *image* $h(A)$ of h is the subinstance of B defined by the condition that $\mathbf{v}$ is in the relation R in $h(A)$ if there exists $\mathbf{u}$ in R in A so that $h(\mathbf{u}) = \mathbf{v}$. If $h(A) = B$ we say that h is *strongly surjective* and write $h : A \twoheadrightarrow B$. If h is not a structure isomorphism we say that $h(A)$ is a *proper* image.

A *reflective subinstance* of A is an instance B with an inclusion homomorphism $m : B \hookrightarrow A$ and a strongly surjective homomorphism $q : A \twoheadrightarrow B$ such that $q \circ m$ is the identity on B. Again, we often say that B is a reflective subinstance if it is so up to renaming of nulls, and say that it is a strict reflective subinstance if we want to insist that m is an inclusion, rather than just an injective homomorphism.

If $H = \{h_i : A \to B \mid i \in S\}$ is a family of homomorphisms we say that H is a *covering family*, or simply a *cover*, if $B = \bigcup_{i \in S} h_i(A)$. We say that A *covers* B if $\mathsf{Hom}(A, B)$ is a cover. If $H = \{h_i : A_i \to B \mid i \in S\}$ is a family of homomorphisms with the same codomain we say that H *jointly covers* B if $B = \bigcup_{i \in S} h_i(A_i)$

If A is an incomplete instance, a *freeze* of A is, as usual, a complete instance $\overline{A}$ together with a structure isomorphism between A and $\overline{A}$ that fixes the constants in A. Whenever we take a freeze of an instance, we tacitly assume that it is "fresh", in the sense that the new constants in it do not occur in any other instances currently under consideration (that is, usually, that have been introduced so far in the proof).

We define the *null-disjoint* union $A \sqcup_{\mathsf{Null}} B$ of two instances A and B to be the instance obtained by renaming whatever nulls necessary to make sure that A and B have no nulls in common, and then taking the union of the result. As such, the null-disjoint union is only defined up to isomorphism. The key property of the null-disjoint union is the 1-1 correspondence $\mathsf{Hom}(A \sqcup_{\mathsf{Null}} B, C) \cong \mathsf{Hom}(A, C) \times \mathsf{Hom}(B, C)$ between homomorphisms from $A \sqcup_{\mathsf{Null}} B$ and pairs of homomorphisms from A and B.

The definition extends to n-ary and infinitary null-disjoint unions. (Infinitary null-disjoint unions are, strictly speaking, not database instances in so far as they are not finite, but they are an occasionally useful technical extrapolation, and we trust that they will cause no confusion in the places where we make use of them.) We shall mostly be considering the null-disjoint union of an instance with itself. For $n \in \mathbb{N}^+ \cup \{\infty\}$, we abuse notation and simply write A^n for the null-disjoint union of A with itself n times, with the property that $\mathsf{Hom}(A^n, C) \cong \prod_{i=1}^{n} \mathsf{Hom}(A, C)$. We denote by $\pi_m : A \to A^n$, for $m \in \mathbb{N}^+$ smaller or equal to n, the homomorphism that sends A to the mth copy of it in A^n. If $f : A^n \to C$ is a homomorphism we write $f = \langle f_1, \ldots, f_n \rangle$ where $f_i = f \circ \pi_i : A \to C$. We denote by $\nabla : A^n \to A$ the homomorphism that corresponds to the n-tuple of identity homomorphisms $A \to A$. That is to say, ∇ identifies all copies in A^n of a null in A with that null.

2.3 Semantics of Incomplete Databases

A *semantics* is a function $[\![-]\!] : \mathcal{D} \to \mathcal{P}^+(\mathcal{C})$ which assigns a non-empty set $[\![A]\!]$ of complete instances to every instance A. We say that A *represents* $[\![A]\!]$.

A semantics $[\![-]\!]$ induces a preordering on $\mathcal{D}$ by $A \leq B \Leftrightarrow [\![A]\!] \subseteq [\![B]\!]$[1]. We say that A and B are *semantically equivalent*, and write $A \equiv B$, if $A \leq B$ and $B \leq A$. Accordingly, $A \equiv B \Leftrightarrow [\![A]\!] = [\![B]\!]$. The semantic equivalence class of an instance is denoted using square brackets: $[A] := \{B \in \mathcal{D} \mid A \equiv B\}$.

A *representative function* (*cf. representative set, canonical function* in [12]) is a function $\chi : \mathcal{D} \to \mathcal{D}$ which picks a representative of each semantic equivalence class. We shall be content with $\chi(A)$ being defined up to isomorphism. A representative function χ is *subinstance minimal* if $\chi(A)$ is a subinstance of all members of $[A]$.

Next, we briefly recall the established semantics OWA, CWA, the Closed Powerset semantics of [12], and the Open and Closed World Assumption as defined by Libkin and Sirangelo [24].

[1] Note that this is the opposite of the standard order as defined in e.g. [12]

2.3.1 Open World Approach: OWA

Under OWA (Open World Assumption) an instance A represents the set of complete instances to which A has a (database) homomorphism; $[\![A]\!]_{\mathsf{OWA}} = \{I \in \mathcal{C} \mid \mathsf{Hom}(A, I) \neq \emptyset\}$.

Consequently, $[\![A]\!]_{\mathsf{OWA}}$ is closed under structure homomorphisms that fix the constants in A pointwise, in the sense that if $I \in [\![A]\!]_{\mathsf{OWA}}$ and $I \to J$ is a structure homomorphism that fixes the constants in A, then $J \in [\![A]\!]_{\mathsf{OWA}}$. It is well known (see e.g. [9]) that the function $\mathsf{Core}(-)$ that maps each instance to its core is a subinstance minimal representative function.

2.3.2 Closed World Approach: CWA

Under CWA (Closed World Assumption) an instance A represesents the set of its images; $[\![A]\!]_{\mathsf{CWA}} = \{I \in \mathcal{C} \mid \text{there exists } h : A \twoheadrightarrow I\}$

Note that $[\![A]\!]_{\mathsf{CWA}}$ is closed under strongly surjective structure homomorphisms that fix the constants in A pointwise. Clearly, the only possible representative function (up to isomorphism, as usual) is the identity.

2.3.3 Closed Powerset: PCWA

Under *Closed Powerset semantics* (PCWA) [12], A represents the set of its CWA-interpretations closed under union; $[\![A]\!]_{\mathsf{PCWA}} = \overline{[\![A]\!]_{\mathsf{CWA}}}^{\cup} = \{I_1 \cup \ldots \cup I_n | n \in \mathbb{N}^+,\ I_1, \ldots, I_n \in [\![A]\!]_{\mathsf{CWA}}\}$ Consequently, $[\![A]\!]_{\mathsf{PCWA}}$ is closed under unions and under strongly surjective homomorphisms that fix the constants in A pointwise. Note that in [12] this semantics is denoted $(|A|)_{\mathsf{CWA}}$ We recall the following from [12, Thm 10.1];

▶ **Proposition 1.** *$A \leq_{\mathsf{PCWA}} B$ iff there exists a cover from B to A.*

Thus, $A \equiv_{\mathsf{PCWA}} B$ iff there exists a cover both from B to A and from A to B. The existence of minimal representative functions for PCWA is the subject of Section 5 and 6.

▶ Remark 2. The semantics GCWA introduced in [17] defines $[\![A]\!]_{\mathsf{GCWA}}$ as the set of unions of *minimal* images of A. In [12] $[\![A]\!]_{\mathsf{GCWA}}$ is denoted by $(|A|)^{\min}_{\mathsf{CWA}}$. As with OWA, $\mathsf{Core}(-)$ is a minimal representative function for GCWA (see [12])

$[\![A]\!]_{\mathsf{GCWA}}$ is not in general closed under strong surjections preserving the constants in A (cf. [12, 9.1]). It therefore cannot be represented in the semantics introduced in Section 3 below.

2.3.4 Mixed Approach: OCWA$^{\mathrm{LS}}$

Let A be an annotated instance, i.e. such that each occurrence of a constant or null is annotated as open or closed. Under OCWA$^{\mathrm{LS}}$ (Open and Closed World Assumption - Libkin/Sirangelo) the set of complete instances represented by A is defined in two steps as follows [24]: for all complete instances I, $I \in [\![A]\!]_{\mathsf{OCWA}^{\mathrm{LS}}}$ if

(i) there exists a homomorphism $h : A \to I$; and

(ii) for every $R(\mathbf{t})$ in I there exists a $R(\mathbf{t}')$ in A such that $h(\mathbf{t}')$ and $\mathbf{t}$ agree on all positions annotated as closed in $\mathbf{t}'$.

OCWA^{LS} is subsumed by a more expressive semantics which we define next.

3 Our Semantics: OCWA*

In this section we propose the semantics OCWA* for annotated instances as a properly more expressive version of both OCWA$^{\mathrm{LS}}$and PCWA. The semantics OCWA* presupposes that instances are annotated according to certain conditions, which we define first:

▶ **Definition 3.** *We say that an annotated instance A is in* normal form *if:*

1. *all occurrences of constants in A are annotated as closed; and*
2. *all occurrences in A of a null agree on the annotation of that null.*

The following then allows us to restrict attention to instances in normal form without loss of generality with respect to OCWA^{LS}.

▶ **Proposition 4.** *Let A be an annotated instance. Then there exists an annotated instance A' in normal form such that $[\![A]\!]_{\mathsf{OCWA^{LS}}} = [\![A']\!]_{\mathsf{OCWA^{LS}}}$.*

Proof. For any atoms that contain open constants or open nulls annotated as closed elsewhere, change the annotation to "closed" and add a copy of the atom where those terms are replaced by fresh open nulls. ◀

▶ **Definition 5.** *If A is a normal form annotated instance and B is an instance, an* RCN-cover *$H : A \overset{\mathrm{RCN}}{\rightrightarrows} B$ is a set $H \subseteq \mathsf{Hom}(A, B)$ such that the homomorphisms in H are jointly strongly surjective and agree on the closed nulls of A.*

▶ **Definition 6** (OCWA*)**.** *Let A be a annotated instance in normal form. Then A represents those complete instances under OCWA* that it RCN-covers; $[\![A]\!]_{\mathsf{OCWA^*}} = \{I \in \mathcal{C} \,|\, \exists H : A \overset{\mathrm{RCN}}{\rightrightarrows} I\}$.*

▶ Remark 7. The definition of $[\![A]\!]_{\mathsf{OCWA^*}}$ could equivalently be given as the set of finite unions $h_1(A) \cup \ldots \cup h_n(A)$ of complete images of A such that the homomorphisms $h_1, \ldots, h_n$ agree on the closed nulls of A. Thus OCWA* lies within what [12] call *Powerset semantics*; that is, semantics that are defined in terms of a relation from instances to sets of complete instances (certain finite sets of valuations, in this case) and a relation from sets of complete instances to complete instances (unions, in this case).

OCWA* properly extends OCWA$^{\mathrm{LS}}$in the following sense:

▶ **Theorem 8.** 1. *For every normal form annotated instance A one can compute in time linear in $|A|$ a normal form annotated instance A' such that $[\![A]\!]_{\mathsf{OCWA^{LS}}} = [\![A']\!]_{\mathsf{OCWA^{LS}}} = [\![A']\!]_{\mathsf{OCWA^*}}$.*

2. *There is a normal form annotated instance A such that for every A' it holds that $[\![A]\!]_{\mathsf{OCWA^*}} \neq [\![A']\!]_{\mathsf{OCWA^{LS}}}$.*

Proof. (1) Given A, extend it to a new instance A' by: for each atom $R(\mathbf{t})$ in A add an atom $R(\mathbf{t}')$ where $\mathbf{t}'$ has every *occurrence* of an open null in $\mathbf{t}$ replaced by a fresh open null. It is then straightforwardly verified that $[\![A]\!]_{\mathsf{OCWA^{LS}}} = [\![A']\!]_{\mathsf{OCWA^{LS}}} = [\![A']\!]_{\mathsf{OCWA^*}}$.

(2) Consider the annotated instance $A = \{R(a^c, x^o, x^o)\}$. The instances in $[\![A]\!]_{\mathsf{OCWA^*}}$ contain only tuples where the second and third coordinate are equal. However, the definition of OCWA^{LS} requires only that one tuple in each instance from $[\![A]\!]_{\mathsf{OCWA^{LS}}}$ respects this equality. Since there is no bound on the size of instances in $[\![A]\!]_{\mathsf{OCWA^*}}$, there is no A' such that $[\![A']\!]_{\mathsf{OCWA^{LS}}} = [\![A]\!]_{\mathsf{OCWA^*}}$. ◀

Regarding PCWA, if A is a normal form annotated instance without any closed nulls, then an RCN-cover $A \overset{\text{RCN}}{\rightrightarrows} C$ is simply a cover, since there are no closed nulls to agree upon. Thus PCWA is OCWA* restricted to instances without closed nulls. Explicitly, let A be an un-annotated instance, and let its *canonical annotation* be that which annotates each constant as closed and each null as open. Then we have:

▶ **Proposition 9.** *Let A be an (un-annotated) instance and let $A[]$ be that instance with canonical annotation. Then $[\![A[]]\!]_{\mathsf{OCWA}^*} = [\![A]\!]_{\mathsf{PCWA}}$.*

For the rest of this paper we assume that all annotated instances are in normal form. This allows us to introduce some notational conventions that simplify the study of RCN-covers on such instances. We also switch to annotating nulls by using lower and upper case instead of superscripts, since this allows us to more clearly emphasize the distinguished status of the closed nulls. We introduce the following conventions:

– Open nulls are written in lower case, x, y, z. Closed nulls are written in upper case, X, Y, Z. (All instances are in normal form, so no null may occur both in lower and upper case in an instance.)

– We display the closed nulls of an instance together with the instance; so that $A[\mathbf{X}]$ is an annotated instance where $\mathbf{X}$ is a listing of the closed nulls of the instance. Thus $\mathbf{X}$ can be the empty list. We allow ourselves to treat $\mathbf{X}$ as the set of closed nulls of A when convenient. It is a list for purposes of substitution. In particular:

– If $\mathbf{t}$ is a list of constants or nulls, $A[\mathbf{t}/\mathbf{X}]$ is the instance obtained by replacing $\mathbf{X}$ with $\mathbf{t}$. If clear from context, we use $A[\mathbf{t}]$ as shorthand.

– Let $n \in \mathbb{N}^+ \cup \{\infty\}$. Recall from Section 2.2 that we, for an un-annotated instance A, write A^n as a shorthand for the n-ary null-disjoint union of A with itself. For an annotated instance $A[\mathbf{X}]$ with closed nulls $\mathbf{X}$, we extend this notation and write $A^n[\mathbf{X}]$ for the n-ary *open-null-disjoint* union; that is, the result of taking the union of n copies of $A[\mathbf{X}]$ where the *open* nulls have been renamed so that no two copies have any open nulls in common. Accordingly, a homomorphism $A^n[\mathbf{X}] \to C$ corresponds to an n-tuple of homomorphisms $A[\mathbf{X}] \to C$ that agree on the closed nulls $\mathbf{X}$.

We close this section by displaying some equivalent definitions of $[\![A[\mathbf{X}]]\!]_{\mathsf{OCWA}^*}$, including in terms of CWA and PCWA, which will be made use of in the sequel. Note that for $n \in \mathbb{N}^+ \cup \{\infty\}$, the family $\{\pi_m : A[\mathbf{X}] \to A^n[\mathbf{X}] \mid m \leq n,\ m \in \mathbb{N}^+\}$ forms a RCN cover from $A[\mathbf{X}]$ to $A^n[\mathbf{X}]$.

▶ **Theorem 10.** *Let $A[\mathbf{X}]$ be an annotated instance and I a complete instance. The following are equivalent:*

1. *$I \in [\![A[\mathbf{X}]]\!]_{\mathsf{OCWA}^*}$, i.e there exist an RCN-cover $A[\mathbf{X}] \overset{\text{RCN}}{\rightrightarrows} I$;*
2. *$I \in \bigcup_{n \in \mathbb{N}^+} [\![A^n[\mathbf{X}]]\!]_{\mathsf{CWA}}$;*
3. *$I \in [\![A^\infty[\mathbf{X}]]\!]_{\mathsf{CWA}}$; and*
4. *$I \in \bigcup_{\mathbf{d} \in \mathsf{Const}^k} [\![A[\mathbf{d}/\mathbf{X}]]\!]_{\mathsf{PCWA}}$ where k is the length of $\mathbf{X}$.*

▶ **Corollary 11.** *$[\![A[\mathbf{X}]]\!]_{\mathsf{OCWA}^*}$ is closed under strongly surjective structure homomorphisms that fix the constants in $A[\mathbf{X}]$ pointwise.*

We now proceed to the study of implication and equivalence OCWA*.

4 OCWA*: Implication, Equivalence

Since RCN-covers are closed under left composition with strong surjections, we have (by Theorem 10) that $[\![A[\mathbf{X}]]\!] \subseteq [\![B[\mathbf{Y}]]\!]$ iff there is an RCN-cover from $B[\mathbf{Y}]$ to $A^n[\mathbf{X}]$, for all $n \in \mathbb{N}^+$, or, equivalently, that there is an RCN-cover from $B[\mathbf{Y}]$ to $A^\infty[\mathbf{X}]$. We display this and show that n can be bounded by a number depending on B, or indeed that n can be bounded by 2 if one considers RCN-covers of a particular form. Note that the following theorem can also be applied to OCWA^{LS} via the translations of Proposition 4 and Theorem 8.

▶ **Theorem 12.** *Let $A[\mathbf{X}]$ and $B[\mathbf{Y}]$ be annotated instances. The following are equivalent:*
- **(i)** $[\![A[\mathbf{X}]]\!]_{\mathsf{OCWA}^*} \subseteq [\![B[\mathbf{Y}]]\!]_{\mathsf{OCWA}^*}$.
- **(ii)** *There is an RCN-cover from $B[\mathbf{Y}]$ to $A^n[\mathbf{X}]$, for all $n \in \mathbb{N}^+$.*
- **(iii)** *There is an RCN-cover from $B[\mathbf{Y}]$ to $A^\infty[\mathbf{X}]$*
- **(iv)** *There is a strongly surjective homomorphism from $B^\infty[\mathbf{Y}]$ to $A^\infty[\mathbf{X}]$.*
- **(v)** *There is an RCN-cover from $B[\mathbf{Y}]$ to $A^{n+1}[\mathbf{X}]$ where n is the number of closed nulls in $B[\mathbf{Y}]$, i.e. the length of $\mathbf{Y}$.*
- **(vi)** *There exists a RCN-cover H from $B[\mathbf{Y}]$ to $A^2[\mathbf{X}]$ such that H contains at least one homomorphism h which factors through $\pi_1 : A[\mathbf{X}] \to A^2[\mathbf{X}]$.*

Proof. v⇒vi : Let n be the length of $\mathbf{Y}$, and let H be an RCN-cover from $B[\mathbf{Y}]$ to $A^{n+1}[\mathbf{X}]$. Choose an h in H. There are more copies of $A[\mathbf{X}]$ in $A^{n+1}[\mathbf{X}]$ than there are $\mathbf{Y}$s, so we can assume that for all closed nulls Y_i in $B[\mathbf{Y}]$, if $h(Y_i)$ is in the $n+1$th copy, then $h(Y_i)$ is either a closed null or a constant. Then the composite $h' = \langle \pi_1, \dots, \pi_n, \pi_n \rangle \circ h : B[\mathbf{Y}] \to A^{n+1}[\mathbf{X}] \to A^{n+1}[\mathbf{X}]$ agrees with H on all $\mathbf{Y}$, so $H' = H \cup \{h'\}$ is an RCN-cover. Now, if we compose H' with the strong surjection $\langle \pi_1, \dots, \pi_1, \pi_2 \rangle : A^{n+1}[\mathbf{X}] \to A^2[\mathbf{X}]$ which sends the n first copies of $A^{n+1}[\mathbf{X}]$ to the first copy in $A^2[\mathbf{X}]$ and the $n+1$th copy of $A^{n+1}[\mathbf{X}]$ to the second in $A^2[\mathbf{X}]$, we obtain an RCN-cover of $A[\mathbf{X}] \to A^2[\mathbf{X}]$ in which the map $\langle \pi_1, \dots, \pi_1, \pi_2 \rangle \circ h'$ factors through $\pi_1 : A[\mathbf{X}] \to A^2[\mathbf{X}]$.

vi⇒ii: Let n be given, and let H be an RCN-cover from $B[\mathbf{Y}]$ to $A^2[\mathbf{X}]$ such that $h \in H$ factors through $\pi_1 : A[\mathbf{X}] \to A^2[\mathbf{X}]$. For $1 \leq i \leq n$, $\pi_1 : A[\mathbf{X}] \to A^n[\mathbf{X}]$ and $\pi_i : A \to A^n[\mathbf{X}]$ is a pair of homomorphisms that agree on closed nulls, so correspond to a homomorphism $\langle \pi_1, \pi_i \rangle : A^2[\mathbf{X}] \to A^n[\mathbf{X}]$. The family $\{\langle \pi_1, \pi_i \rangle \mid 1 \leq i \leq n\}$ of such homomorphisms is an RCN-cover from $A^2[\mathbf{X}]$ to $A^n[\mathbf{X}]$. The composite of this cover with H is RCN, since for any closed null Y_i in $B[\mathbf{Y}]$, $h' \in H$, $1 \leq i \leq n$, we have that $\langle \pi_1, \pi_i \rangle (h'(Y_i)) = \langle \pi_1, \pi_i \rangle (h(Y_i)) = \pi_1(h(Y_i))$.

The remaining implications are straightforward. ◀

From Theorem 12 we can derive two guess-and-check algorithms to decide containment between annotated instances. On the one hand, we may construct $A^{n+1}[\mathbf{X}]$, where n is the length of $\mathbf{Y}$, guess a set of homomorphisms from $B[\mathbf{Y}]$ to this instance, and check that it is an RCN-cover. Alternatively, we may avoid this blowup of $A[\mathbf{X}]$ by constructing $A^2[\mathbf{X}]$, guessing a homomorphism h from $B[\mathbf{Y}]$ to $A[\mathbf{X}]$ as well as a set of homomorphisms H from $B[\mathbf{Y}]$ to $A^2[\mathbf{X}]$, and checking that $\{h\} \cup H$ is an RCN-cover.

Complexity analysis

Since the instance $A^{n+1}[\mathbf{X}]$ has size at most $|A[\mathbf{X}]| \times (|\mathbf{Y}| + 1)$, and the number of homomorphisms in any non-redundant cover is bounded by the number of tuples in the target instance, the complexity of this problem stays in NP. For NP-hardness, we adapt the reduction of 3-colourability for graphs to the problem of deciding whether a given graph has a homomorphism into K_3, the complete graph on three vertices. It is easy to see that any homomorphism from a graph with at least one edge into K_3 extends to a cover of K_3. Therefore, the problem of deciding $\leq_{\mathsf{PCWA}}$, and consequently $\leq_{\mathsf{OCWA*}}$, is likewise NP-complete. It follows that the problem of deciding, given two instances A and B, whether A is a minimal equivalent instance for B given a partial order among instances, belongs to the class DP, as it involves checking the non-existence of a smaller instance. In other words, deciding semantic implication and equivalence for annotated instances has the same complexity as the homomorphism problem.

5 Issues with Minimality in OCWA*

In this section and the next we study the notion of OCWA* semantic equivalence and the question of whether, or to what extent, there exists a unique "best" annotated instance to choose among those that are semantically equivalent. For motivation and illustration, we first recall the situation in OWA in some more detail. It is well known that $A \equiv_{\mathsf{OWA}} B$ if and only if A and B are "homomorphically equivalent", that is, if there exists a homomorphism both from A to B and from B to A. Furthermore, there is, up to isomorphism, a least subinstance of A to which it is homomorphically equivalent, known as the *core* of A. Instances A and B are homomorphically equivalent if and only if their cores are isomorphic. Moreover, as a consequence of being the least homomorphically equivalent subinstance of A, the core of A is also the least reflective subinstance of A, and the least homomorphically equivalent image of A. Thus there are three quite natural notions of minimality to which the core is the answer in OWA. We say that an instance is *a core* if it is its own core, i.e. if it has no homomorphically equivalent subinstances. Cores can be characterized as those instances C with the property that any homomorphism $C \to C$ must be an isomorphism. (See [9, 10, 16] for more about cores.)

We show now that for OCWA* there does not in general exist least semantically equivalent instances in any of the three senses above. We then turn to the question of whether a "good" representative function can nevertheless be found, first for PCWA and then for OCWA* in general. We begin by fixing some terminology.

▶ **Definition 13.** *Let A and B be instances. In the context of a given semantics, we say that:*

1. *B is* sub-minimal *(subinstance minimal) if there are no proper semantically equivalent subinstances of B;*
2. *B is* rfl-minimal *(reflective subinstance minimal) if there are no proper semantically equivalent reflective subinstances of B;*
3. *B is a* least *semantically equivalent (reflective) subinstance of A if $B \equiv A$ and B is a (reflective) subinstance of all semantically equivalent (reflective) subinstances of A;*
4. *B is* img-minimal *(image minimal) if there are no proper semantically equivalent images of B, and finally;*
5. *B is a* least *semantically equivalent image of A if $B \equiv A$ and for all semantically equivalent images C of A, B is an image of C.*

We show by the examples that follow that in PCWA, and hence in OCWA*, least semantically equivalent subinstances, reflective subinstances, and images do not in general exist, and that when they do, they need not coincide. In the examples all instances consist of nulls only.

▶ **Example 14.** B_1 and C_1 are non-isomorphic PCWA-equivalent reflective subinstances of A_1. Both B_1 and C_1 are sub-minimal and rfl-minimal.

A_1	R		
	x	x	y
	x	x	x
	v	w	w
	v	v	v
	z	z	r
	z	s	s
	z	z	z

B_1	R		
	x	x	y
	x	x	x
	v	w	w
	v	v	v

C_1	R		
	z	z	r
	z	s	s
	z	z	z

▶ **Example 15.** The instances B_2 and C_2 are non-isomorphic PCWA-equivalent images of the instance A_2. Both B_2 and C_2 are img-minimal.

A_2	R				
	x	x	u	y	z
	x	x	x	x	z
	x	x	x	y	x
	x	x	x	x	x
	v	p	p	r	s
	p	p	p	p	s
	p	p	p	r	p
	p	p	p	p	p

B_2	R				
	p	p	u	r	z
	p	p	p	p	z
	v	p	p	r	s
	p	p	p	p	s
	p	p	p	r	p
	p	p	p	p	p

C_2	R				
	p	p	u	y	s
	p	p	p	y	p
	v	p	p	r	s
	p	p	p	p	s
	p	p	p	r	p
	p	p	p	p	p

▶ **Example 16.** The instance A_3 has a least PCWA-equivalent reflective subinstance, a least PCWA-equivalent subinstance, and a least PCWA-equivalent image, consisting of the non-isomorphic instances A_3, B_3, and C_3, respectively:

A_3	R				
	x	x’	y	y	z
	v’	v	s	t	s
	x	v	u	u	u
	x	x	x	x	x
	v	v	v	v	v

B_3	R				
	x	x’	y	y	z
	v’	v	s	t	s
	x	x	x	x	x
	v	v	v	v	v

C_3	R				
	w	x’	y	y	z
	v’	w	s	t	s
	w	w	w	w	w

We summarize:

▶ **Theorem 17.** *In PCWA (OCWA*),*

1. *there exists an (annotated) instance A for which there exists two non-isomorphic semantically equivalent sub-minimal subinstances;*
2. *there exists an (annotated) instance A for which there exists two non-isomorphic semantically equivalent img-minimal images; and*
3. *there exists an (annotated) instance A for which there exists two non-isomorphic semantically equivalent rfl-minimal reflective subinstances.*

6 Minimality in PCWA

Recall from Section 2.3 that a representative function for a given semantics is a function $\chi : \mathcal{D} \to \mathcal{D}$ which chooses a representative for each equivalence class. That is to say, $A \equiv B \Leftrightarrow \chi(A) = \chi(B)$, for all $A, B \in \mathcal{D}$, and $\chi(A) \equiv A$, for all $A \in \mathcal{D}$. Again, we only require that $\chi(A)$ is defined up to isomorphism, i.e. up to renaming of nulls. Recall further that a representative function is subinstance minimal if $\chi(A)$ is a subinstance of A (up to isomorphism) for all $A \in \mathcal{D}$. Similarly, we say that a representative function is *image minimal* if $\chi(A)$ is an image of A, and that it is *reflective subinstance minimal* if $\chi(A)$ is a reflective subinstance of A. The canonical example is the Core function, which is a minimal representative function for OWA in all of these three senses.

Theorem 17 showed that there can be no minimal representative function for PCWA, for any of these three senses of "minimal". However, we show that there is a function $\varpi(-) : \mathcal{D} \to \mathcal{P}^{\mathrm{fin}}(\mathcal{D})$ that assigns a finite set $\{E_1, \dots, E_n\}$ to each instance A that is representative in the sense that $A \equiv_{\mathsf{PCWA}} B \Leftrightarrow \varpi(A) = \varpi(B)$, for all $A, B \in \mathcal{D}$, and $\bigcup\limits_{E \in \varpi(A)} E \equiv_{\mathsf{PCWA}} A$, for all $A \in \mathcal{A}$; and "minimal" in the sense that

- E is a reflective subinstance of A, for all $A \in \mathcal{D}$ and all $E \in \varpi(A)$, and
- E is semantically minimal in the strong sense that if $C \equiv_{\mathsf{PCWA}} E$ then E is a reflective subinstance of C, for all $E \in \varpi(A)$.

Thus the members of $\varpi(A)$ are both sub-, img-, and rfl-minimal, in the sense of Definition 13. Furthermore, if $\varpi(A) = \{E_1, \dots, E_n\}$ then

$$[\![A]\!]_{\mathsf{PCWA}} = \overline{[\![E_1]\!]_{\mathsf{CWA}} \cup \dots \cup [\![E_n]\!]_{\mathsf{CWA}}}^{\cup}. \tag{1}$$

We propose $\varpi(-)$ as a form of "power core" or "multi-core" function for PCWA; giving for each A a finite set of PCWA-minimal instances which jointly embody the PCWA-relevant structure of A, analogously to the role that the single instance $\mathsf{Core}(A)$ plays in OWA. In addition to the properties just listed, we show the following as an instance of the usefulness of $\varpi(-)$. For any given instance A, the set of sub-minimal subinstances of A is of course finite. But this set may have no overlap with the set of sub-minimal subinstances of B, even if A and B are semantically equivalent. Thus it is, on the face of it, not obvious that the set $\mathrm{Min}([A]_{\mathsf{PCWA}})$ of sub-minimal members of the whole equivalence class $[A]_{\mathsf{PCWA}}$ must be finite (up to renaming of nulls). However, we show that any sub-minimal member of $[A]_{\mathsf{PCWA}}$ must be an image of $\bigcup_{E \in \varpi(A)} E$, establishing thereby that $\varpi(A)$ both yields a finite bound on the size of $\mathrm{Min}([A]_{\mathsf{PCWA}})$, and a way to compute it.

Moreover, we show in Section 6.3 that for the class of queries known as *existential positive with Boolean universal guards*, the so-called *certain answers* can in fact be computed directly from the elements in $\varpi(A)$, rather than from the larger A.

In the rest of Section 6 we fix the semantics to be PCWA, and thus leave the subscripts implicit.

6.1 PCWA-cores

Recall that A is a core if and only if every homomorphism $A \to A$ is an isomorphism. In analogy, we introduce the notion of PCWA-core as follows.

▶ **Definition 18.** *We say that an instance A is a* PCWA-core *if every self-cover $H \subseteq$ $\mathsf{Hom}(A, A)$ contains an isomorphism.*

▶ **Example 19.** $D = \{R(z, z, r), R(z, z, z)\}$ is a PCWA-core, as the only endomorphism hitting $R(z, z, r)$ is the identity. The core of D is $\{R(z, z, z)\}$.

Accordingly, every core is a PCWA-core. It is also evident that cores have the property that if C is a core and A is any instance, then A and C are OWA semantically equivalent if and only if C is a reflective subinstance of A. For PCWA-cores we have the following:

▶ **Proposition 20.** *Let $A \equiv B$ and assume that A is a PCWA-core. Then A is a reflective subinstance of B.*

Proof. $\mathsf{Hom}(B, A) \circ \mathsf{Hom}(A, B)$ is a cover so it contains an isomorphism. ◀

Consequently, if two PCWA-cores are semantically equivalent, they are isomorphic.

Section 5 introduced three different notions of minimality with respect to semantic equivalence. We relate these to each other and to the property of being a PCWA-core.

▶ **Proposition 21.** *Let A be an instance. The following implications hold and are strict.*

1. *If A is a PCWA-core then A is sub-minimal and img-minimal.*
2. *If A is sub-minimal or img-minimal then it is rfl-minimal.*

Proof. 1) follows from Proposition 20 and 2) is immediate.That the implications are strict is shown in Examples 14 and 16. Specifically, C_1 of Example 14 is both sub-minimal and img-minimal, but it is not a PCWA-core. And A_3 of Example 16 is rfl-minimal, but neither sub- nor img-minimal. ◀

In what follows it is convenient to fix a more compact notation for atoms $R(\mathbf{t})$ that occur in an instance A. We primarily use the variable k for atoms, and write $k : A$ for "k is an atom of A". If $f : A \to B$ is a homomorphism and $k = R(\mathbf{t}) : A$ then $f(k) = R(f(\mathbf{t}))$.

We recall the notion of "core with respect to a tuple":

▶ **Definition 22.** *Let $k : A$. The* core of A with respect to k, *denoted C^A_k, is the least strict reflective subinstance of A containing k.*

The instance C^A_k can be regarded as the "core of A with k frozen", and thus is unique, up to isomorphism. As a reflective subinstance, it comes with an injective homomorphism to A and a strong surjection from A, which we write $m_k : C^A_k \to A$ and $q_k : A \to C^A_k$, respectively. When the instance A is clear from context, we leave the superscript implicit and just write C_k. We display the following for emphasis.

▶ **Lemma 23.** *Any homomorphism $h : C^A_k \to C^A_k$ that fixes k must be an isomorphism.*

▶ **Definition 24.** *We say that two atoms $k, k' : A$ are* endomorphism-equivalent, *and write $k \sim_A k'$, if there exist $f, g \in \mathsf{Hom}(A, A)$ such that $f(k) = k'$ and $g(k') = k$. We say that $k : A$ is* (endomorphism-)maximal *if "only equivalent atoms map to it". That is, for all $k' : A$ and $f \in \mathsf{Hom}(A, A)$, $f(k') = k$ implies that $k \sim_A k'$. If k is maximal we write* $\mathrm{Max}_A(k)$*.*

▶ **Lemma 25.** *Let A be an instance, and $k, k' : A$. If $k \sim_A k'$ then $C_k \cong C_{k'}$.*

Proof. Suppose $f, g \in \mathsf{Hom}(A, A)$ such that $f(k) = k'$ and $g(k') = k$ and consider the diagram

$$\begin{tikzcd}
& A \arrow[loop above, "{f,g}"] \arrow[dl, bend left, "q_{k'}"] \arrow[dr, bend left, "q_k"] & \\
C_{k'} \arrow[ur, "m_{k'}"] \arrow[rr, bend left, "q_k \circ f \circ m_{k'}"] & & C_k \arrow[ul, "m_k"'] \arrow[ll, bend left, "q_{k'} \circ g \circ m_k"]
\end{tikzcd}$$

The homomorphism $h := (q \circ f \circ m') \circ (q' \circ g \circ m : C_k \to C_k)$ fixes k. So h must be an isomorphism. By symmetry, we obtain that $C_k \cong C_{k'}$. ◀

▶ **Lemma 26.** *If $k : A$ is maximal, then C_k is a PCWA-core.*

Proof. First note that for any instance B and any set of homomorphisms $H \subseteq \mathsf{Hom}(B, B)$, if $\overline{H}$ is the closure of H under composition, then: 1) H is a cover if and only if $\overline{H}$ is a cover; and 2) H contains an isomorphism if and only if $\overline{H}$ contains an isomorphism. Let $H \subseteq \mathsf{Hom}(C_k, C_k)$ be a cover, and assume without loss of generality that it is closed under composition. Then we can find $k' : C_k$ and $f \in H$ such that $f(k') = k$. Since k is maximal in A it is maximal in C_k, so there is a homomorphism $h : C_k \to C_k$ such that $h(k) = k'$. But then $f \circ h$ is an isomorphism, so f must be an isomorphism as well. ◀

Thus the maximal atoms of an instance determine a set of reflective subinstances which are PCWA-cores. We show that these are invariant under semantic equivalence.

▶ **Theorem 27.** *Let $H \subseteq \mathsf{Hom}(A, B)$ and $G \subseteq \mathsf{Hom}(B, A)$ be covers. Let $k_B : B$ be maximal. Then there exist $h \in H$ and $k_A : A$ such that k_A is maximal and $h(k_A) = k_B$. Moreover, the homomorphism $q_{k_B} \circ h \circ m_{k_A} : C^A_{k_A} \to C^B_{k_B}$ is an isomorphism.*

Proof. First, we show that, more generally, whenever $A \equiv B$, it is the case that for all $f : A \to B$ and all $k : A$, if $\mathrm{Max}_B(f(k))$ then $\mathrm{Max}_A(k)$.

For suppose $g : A \to A$ and $k' : A$ is such that $g(k') = k$. Choose $f' : B \to A$ and $k'' : B$ such that $f'(k'') = k'$. Then $f \circ g \circ f'(k'') = f(k)$ so there is $f'' : B \to B$ such that $f''(f(k)) = k''$, whence $g \circ f' \circ f'' \circ f(k) = k'$. This establishes the first claim of the theorem.

Next, let $\mathrm{Max}_A(k_A)$, $\mathrm{Max}_B(k_B)$, and $h \in H$ such that $h(k_A) = k_B$. Chose $k' : B$ and $g \in G$ such that $g(k') = k_A$. Then $f \circ g(k') = k_B$, so there exists $f : B \to B$ such that $f(k_B) = k'$.

$$\begin{tikzcd}
A \arrow[r, bend left, "h"] \arrow[d, bend left, "q_{k_A}"] & B \arrow[l, bend left, "g"] \arrow[loop above, "f"] \arrow[d, bend left, "q_{k_B}"] \\
C^A_{k_A} \arrow[u, hook, bend left, "m_{k_A}"] \arrow[r, bend left, "q_{k_B} \circ h \circ m_{k_A}"] & C^B_{k_B} \arrow[u, hook, bend left, "m_{k_B}"] \arrow[l, bend left, "q_{k_A} \circ g \circ f \circ m_{k_B}"]
\end{tikzcd}$$

Then $q_{k_B} \circ h \circ m_{k_A}(k_A) = k_B$ and $q_{k_A} \circ g \circ f \circ m_{k_B}(k_B) = k_A$, whence their composites are isomorphisms. So they must themselves be isomorphisms. ◀

Finally, we note the following property of PCWA-cores which will be used in the next section.

▶ **Lemma 28.** *An instance A is a PCWA-core if and only if there exists $k : A$ with the property that for all $f \in \mathsf{Hom}(A, A)$, if k is in the image of f then f is an isomorphism.*

Proof. Suppose A is a PCWA core. For each maximal k, let $f_k : A \to A$ be the composition of $q_k : A \to C_k$ and $m_k : C_k \to A$. Then $\mathsf{Hom}(A, A) \circ \{f_k \mid \mathrm{Max}_A(k)\}$ is covering, so one of its homomorphisms, and hence one of the f_ks, must be an isomorphism. The converse is immediate. ◀

6.2 PCWA Multicores

Consider the family $\{C_k | \mathrm{Max}_A(k)\}$ of (strict) reflective subinstances of A. From the definition of maximality we have that for any atom $t : A$ there exists a maximal atom $k : A$ and an endomorphism $h : A \to A$ such that $f(k) = t$. Thus the family $\{C_k \mid \mathrm{Max}_A(k)\}$ jointly covers A. Clearly, if we successively remove any member of $\{C_k \mid \mathrm{Max}_A(k)\}$ that is a reflective subinstance of another member, we will retain a subset that still jointly covers A. Thus we can summarize what we have so far with the following.

▶ **Theorem 29.**

1. *For each $A \in \mathcal{D}$ there exists a finite set $\varpi(A) \subseteq \mathcal{D}$ such that:*
   - **a.** *for all $E \in \varpi(A)$, $E \cong C_k^A$ for some maximal $k : A$;*
   - **b.** *for all maximal $k : A$, there exists $E \in \varpi(A)$ such that C_k^A is a reflective subinstance of E; and*
   - **c.** *for all $E, E' \in \varpi(A)$ if E is a reflective subinstance of E' then $E = E'$.*
2. *for given $A \in \mathcal{D}$ the set $\varpi(A)$ is unique with properties 1.(a)–1.(c), up to isomorphisms of its members. That is to say, if X is another set satisfying properties 1.(a)–1.(c), then there exists a bijection $f : \varpi(A) \to X$ such that $f(E) \cong E$.*
3. *$A \equiv B$ if and only if $\varpi(A) = \varpi(B)$, up to isomorphism of the members.*

We refer to $\varpi(A)$ as the *multicore* of A. The multicore of an instance A is only defined up to isomorphisms of its members, so we can assume without loss whenever it is convenient that no nulls are shared between those members; i.e. that for all $E, E' \in \varpi(A)$ we have $\mathrm{dom}(E) \cap \mathrm{dom}(E') \subseteq \mathsf{Cons}$. We also regard multicores as equal when their members are isomorphic.

Before proceeding, we characterize when a set of instances is (up to isomorphism) $\varpi(A)$ for some A. We need the following lemma.

▶ **Lemma 30.** *Let A be an instance and B a reflective subinstance of A. Let $k : A$ and suppose that k is maximal. Then C_k is a reflective subinstance of B if and only if there exists a homomorphism $f : B \to C_k$ and $k' : B$ such that $f(k') = k$.*

Proof. The left-to-right is immediate. Assume that there exists a homomorphism $f : B \to C_k$ and $k' : B$ such that $f(k') = k$. Since k is maximal in A there exists a homomorphism $g : C_k \to B$ such that $g(k) = k'$. But then $f \circ g$ fixes k, so it is an isomorphism. ◀

▶ **Theorem 31.** *Let $\mathfrak{F} = \{C_1, \cdots, C_n\}$ be a family of instances (with no nulls in common). The following are equivalent:*

1. *There exists an instance A such that $\mathfrak{F} = \varpi(A)$ (up to isomorphism of the members).*
2. **a.** *C_i and C_j have the same core (up to isomorphism) for all $i, j \leq n$, and*
   - **b.** *there exists a selection of atoms $k_i : C_i$, $1 \leq i \leq n$, satisfying the condition that if there exists $h : C_j \to C_i$ such that k_i is in the image of h, then $i = j$ and h is an isomorphism.*

Proof. Assume $\mathfrak{F} = \varpi(A)$. Then we can regard $\varpi(A)$ as $\{C_k \mid k \in I\}$ for a set I of maximal $k : A$. Firstly, the core of A is the core of C_k for all $k \in I$. Secondly, by Lemma 30, if $h : C_j \to C_k$ such that k is in the image of h then C_k is a reflective subinstance of C_j, whence by the definition of $\varpi(A)$ we have that $j = k$ and h is an isomorphism.

Assume conditions in a) and b) are satisfied. b) ensures, together with Lemma 28, that C_i is a PCWA core for all $i \in I$. Let $A := \bigcup_{k \in I} C_k$ (relying on the assumption that the members of $\mathfrak{F}$ have no nulls in common). Since the C_is share the same core, C_i is a reflective subinstance of A for all $i \in I$. Specifically, $m_i : C_i \to A$ is the inclusion and $q_i : A \to C_i$ is the homomorphism induced by $\{f_{j,i} : C_j \to C_i \mid j \in I\}$ where $f_{j,i}$ sends C_j to the core if $j \neq i$, and $f_{i,i}$ is the identity. Next, to show that c_i is maximal in A for all $i \in I$: suppose there exists a homomorphism $h : A \to A$ and a $t : A$ such that $h(t) = c_i$. Then t is contained in some C_j. By composing

$$\begin{tikzcd} C_j \arrow[r, "m_j"] & A \arrow[r, "h"] & A \arrow[r, "q_i"] & C_i \end{tikzcd}$$

and by Lemma 30, we see that $i = j$ and $(q_i \circ h \circ m_i)$ is an isomorphism on C_i. Hence $m_i \circ (q_i \circ h \circ m_i)^{-1} \circ q_i(c_i) = t$. Finally, if $k : A$ is maximal, then $k : C_i$ for some $i \in I$, whence C_k is a reflective subinstance of C_i. ◀

Let $\chi_p(A)$ be the union of all members of the multicore, where these are chosen so as to have no nulls in common, $\chi_p(A) := \bigcup_{E \in \varpi(A)} E$. It is now easy to see that $\chi_p(A) \equiv A$, so that $\chi_p(-)$ is a representative function, in the sense of Section 2.3. $\chi_p(A)$ need not be minimal either in terms of subinstances or images. However, an instance is sub-minimal only if it is an image of $\chi_p(A)$, as we show by way of the following lemma.

▶ **Lemma 32.** *Let $A \in \mathcal{D}$. There exists a homomorphism $m : \chi_p(A) \to A$ such that $\chi_p(A) \equiv m(\chi_p(A)) \equiv A$.*

Proof. We can regard $\varpi(A)$ as a set $\{C_k^A \mid k \in S\}$ for a suitable set S. For each $k \in S$ we have an inclusion $i_k : C_k \to A$, and a strong surjection $s_k : A \to C_k$. The family $\{i_k : C_k \to A \mid k \in S\}$ determines a homomorphism $m : \chi_p(A) \to A$. For each k the inclusion $i_k : C_k \to A$ factors through $m(\chi_p(A))$ and the composite $C_k \xrightarrow{i_k} m(\chi_p(A)) \subseteq A \xrightarrow{s_k} C_k$ is the identity. Thus $\chi_p(A) \equiv m(\chi_p(A)) \equiv A$. ◀

▶ **Theorem 33.** *An instance A is sub-minimal only if there exists a strongly surjective homomorphism $m : \chi_p(A) \to A$.*

▶ **Corollary 34.** *Identifying isomorphic instances, the number of sub-minimal instances that are semantically equivalent to A is bounded by the number of (semantically equivalent) images of $\chi_p(A)$.*

▶ Remark 35. We note that there will usually be proper semantically equivalent images of $\chi_p(A)$. In particular, this always exists if the core of A has a null in it and $\varpi(A)$ has more than one member. The reason is that members of $\varpi(A)$ can be "glued" along common reflective subinstances; such subinstances induce a filter which yields a semantically equivalent image of $\chi_p(A)$. Observe that if $\varpi(A)$ has a *single* member, then that member is equivalent to A, and thus $[A]$ has a least element both in terms of subinstances, reflective subinstances, and images.

▶ **Example 36.** Consider Example 15. $\varpi(A_2)$ consists of the two PCWA-cores C_{k_1} and C_{k_5}. In addition to the core, C_{k_1} and C_{k_5} have the reflective subinstances V and W in common.

C_{k_1}	R				
k_1	x	x	u	y	z
k_2	x	x	x	x	z
k_3	x	x	x	y	x
k_4	x	x	x	x	x

C_{k_5}	R				
k_5	v	p	p	r	s
k_6	p	p	p	p	s
k_7	p	p	p	r	p
k_8	p	p	p	p	p

W	R				
	p	p	p	p	s
	p	p	p	p	p

V	R				
	x	x	x	y	x
	x	x	x	x	x

The filter induced on $C_{k_1} \cup C_{k_5} = A_2$ by V identifies x with p and y with r. If we write out the resulting image by overwriting x with p and y with r, we obtain B_2 of Example 15. It follows that B_2 is a semantically equivalent image of A_2. Similarly, from W we see that we can produce a semantically equivalent image of A_2 by overwriting x with p and z with s. This results in C_2.

6.3 Naïve Evaluation of Queries

Before proceeding to the study of minimality for OCWA* in general, we make an example remark on the use of $\varpi(-)$ in the evaluation of queries. The motivation is, briefly, that it may be significantly cheaper to evaluate a query separately on the smaller instances in $\varpi(A)$ than on all of A.

Recall from e.g. [12] that that the *certain answers* of a query Q on an instance A under a semantics $[\![-]\!]$ is the intersection of the answers obtained on $[\![A]\!]$: $\mathsf{certain}(Q, A) := \bigcap\{Q(I) \mid I \in [\![A]\!]\}$. The *naïve evaluation* of Q on A is the result of removing all tuples with nulls from $Q(A)$. Naïve evaluation is said to *work* for Q if it produces precisely the certain answers.

It is shown in [12] that naïve evaluation works for the class $\exists\mathbf{Pos} + \forall\mathbf{G}^{\mathrm{bool}}$ of existential positive queries with Boolean universal guards with respect to PCWA. Here $\exists\mathbf{Pos} + \forall\mathbf{G}^{\mathrm{bool}}$ is the least class of formulas containing all atomic formulas, including equality statements, and closed under conjunction; disjunction; existential quantification; and the following rule: if α is an atomic formula, ϕ a formula in $\exists\mathbf{Pos} + \forall\mathbf{G}^{\mathrm{bool}}$, and $\mathbf{x}$ a list of distinct variables containing all free variables in both α and ϕ, then $\forall\mathbf{x}(\alpha \to \phi)$ is a formula in $\exists\mathbf{Pos} + \forall\mathbf{G}^{\mathrm{bool}}$.

▶ **Theorem 37.** *Let Q be a query of arity n in $\exists\mathbf{Pos} + \forall\mathbf{G}^{\mathrm{bool}}$. Then* $\mathsf{certain}(Q, A) = \bigcap\{Q(E) \mid E \in \varpi(A)\} \cap \mathsf{Const}^n$.

Proof. The inclusion from left to right follows from the fact that naïve evaluation works for Q, that Q is preserved under strong surjections, and that each $E \in \varpi(A)$ is an image of A. For the inclusion from right to left, it is sufficient to show that if $\mathbf{a}$ is a tuple of constants in $\bigcap\{Q(E) \mid E \in \varpi(A)\}$ then $\mathbf{a} \in Q(I)$ for all $I \in \overline{\cup_{E\in\varpi(A)}[\![E]\!]_{\mathsf{CWA}}}^{\cup}$. But this is a straightforward modification of the proof that formulas in $\exists\mathbf{Pos} + \forall\mathbf{G}^{\mathrm{bool}}$ are preserved under unions of strong surjections (Lemma 10.12) in [12]. ◀

7 Minimality in OCWA*

We return now to OCWA* in general and apply our results from the special case of the previous section. Recall from Section 4 that in order to determine whether $A[\mathbf{X}] \leq_{\mathsf{OCWA*}} B[\mathbf{Y}]$ we have to look for an RCN-cover from $B[\mathbf{Y}]$ to $A^{\mathrm{length}(\mathbf{Y})+1}[\mathbf{X}]$. The reason is that RCN-covers do not compose; it is insufficient just to know that we have an RCN-cover from $B[\mathbf{Y}]$ to $A[\mathbf{X}]$. This fact complicates the study of minimality for OCWA*. However, note that if there exists an RCN-cover from $B[\mathbf{Y}]$ to $A[\mathbf{X}]$ which sends the closed nulls $\mathbf{Y}$ to closed terms in $A[\mathbf{X}]$ – i.e. either to $\mathbf{X}$ or to constants – then, because such covers do compose, we have $A[\mathbf{X}] \leq_{\mathsf{OCWA*}} B[\mathbf{Y}]$. But, on the face of it, we cannot restrict to such “closedness-preserving” covers. Consider the following example.

▶ **Example 38.** The following annotated instances are all semantically equivalent.

$$A[V, W] = \{R(x, y), R(V, W)\} \quad A[V, w] = \{R(x, y), R(V, w)\} \quad A[v, w] = \{R(x, y), R(v, w)\}$$
$$A[v, W] = \{R(x, y), R(v, W)\} \quad B[] = \{R(x, y)\}$$

Although $A[V, W]$ and $B[]$ are equivalent, there is no RCN-cover from $A[V, W]$ to $B[]$ that satisfies the restriction that closed nulls should be sent to closed terms, since everything in $B[]$ is open. It is the (ordinary) RCN-cover to $B^3[]$ that witnesses that $B[] \leq_{\mathsf{OCWA*}} A[V, W]$. Nevertheless, Example 38 hints at a solution to this; the problem with $A[V, W]$, it can be said, is that it has closed nulls that could equivalently have been annotated as open. Once we re-annotate to $A[v, w]$, the equivalence with $B[]$ *is* witnessed by a cover to $B[]$. We show in this section that if we restrict to annotated instances where no closed null can be equivalently replaced by an open null, then any semantic equivalence is witnessed by RCN-covers which preserve closed nulls. These closed nulls can then, essentially, be treated as constants, so that the results of Section 6 can be applied.

We note, first, when an annotated instance is semantically equivalent to an instance without closed nulls. Our fixed semantics in this section is OCWA*.

▶ **Proposition 39.** *Let $A[\mathbf{X}]$ be an annotated instance. The following are equivalent:*

1. *$A[\mathbf{X}]$ is semantically equivalent to an instance $B[]$ in which all nulls are open.*
2. *$[\![A[\mathbf{X}]]\!]$ is closed under unions.*
3. *$A[\mathbf{X}]$ is semantically equivalent to the instance $A[\mathbf{x}]$ obtained by changing the annotation of $A[\mathbf{X}]$ so that all nulls are open.*

Proof. For 2. ⇒ 3., note that $[\![A[\mathbf{X}]]\!] \subseteq [\![A[\mathbf{x}]]\!]$ is clear; and it is also clear that $[\![A[\mathbf{x}]]\!]_{\mathsf{CWA}} \subseteq [\![A[\mathbf{X}]]\!]$. But then, since $[\![A[\mathbf{X}]]\!]$ is closed under unions, $[\![A[\mathbf{x}]]\!] \subseteq [\![A[\mathbf{X}]]\!]$. ◀

▶ **Definition 40.** *Let $A[\mathbf{X}, \mathbf{Y}]$ be an annotated instance. We say that $\mathbf{X}$ is* annotation redundant *(relative to $\mathbf{Y}$) if $A[\mathbf{X}, \mathbf{Y}] \equiv A[\mathbf{x}, \mathbf{Y}]$ i.e. if changing the annotation of $\mathbf{X}$ to "open" yields an equivalent instance. We say that an annotated incomplete instance is* annotation minimal *is no subset of its closed nulls are annotation redundant (with respect to the rest).*

▶ **Lemma 41.** *Let $A[\mathbf{X}, \mathbf{Y}]$ be an annotated incomplete instance and $\mathbf{c}$ a list of the same length as $\mathbf{Y}$ of distinct constants not occurring in A . Then $\mathbf{X}$ is redundant with respect to $\mathbf{Y}$if and only if for all finite lists of instances $I_1, \ldots, I_k \in [\![A[\mathbf{X}, \mathbf{c}]]\!]$ it is the case that $I_1 \cup \ldots \cup I_k \in [\![A[\mathbf{X}, \mathbf{Y}]]\!]$.*

Proof. *If*: We must show that $A[\mathbf{X}, \mathbf{Y}] \equiv A[\mathbf{x}, \mathbf{Y}]$. Let $m \geq 1$ be given, and consider $A^m[\mathbf{x}, \mathbf{Y}]$. Let J be a freeze of $A^m[\mathbf{x}, \mathbf{Y}]$ where $\mathbf{Y}$ is replaced by $\mathbf{c}$ and the other nulls by fresh constants. Then $J = I_1 \cup \ldots \cup I_m$ where $I_1, \ldots, I_m \in [\![A[\mathbf{X}, \mathbf{c}]]\!]$. So $J \in A[\mathbf{X}, \mathbf{Y}]$, by assumption, and then $A[\mathbf{X}, \mathbf{Y}] \equiv A[\mathbf{x}, \mathbf{Y}]$ by Theorem 12.

Only if: If $I_1, \ldots, I_k \in [\![A[\mathbf{X}, \mathbf{c}]]\!]$ then $I_1, \ldots, I_k \in [\![A[\mathbf{x}, \mathbf{c}]]\!]$, and then, since the latter is closed under unions, $I_1 \cup \ldots \cup I_k \in [\![A[\mathbf{x}, \mathbf{c}]\!] \subseteq [\![A[\mathbf{x}, \mathbf{Y}]]\!] = [\![A[\mathbf{X}, \mathbf{Y}]]\!]$. ◀

The following theorem displays the main property of annotation-minimal instances. The proof is rather long and is omitted for reasons of space.

▶ **Theorem 42.** *Let $A[\mathbf{X}]$ and $B[\mathbf{Y}]$ be two annotation-minimal instances such that $A[\mathbf{X}] \equiv B[\mathbf{Y}]$. Then for all strong surjections $f : A^\infty[\mathbf{X}] \twoheadrightarrow B^\infty[\mathbf{Y}]$ it is the case that f restricts to a bijection $f \restriction_{\mathbf{X}} : \mathbf{X} \to \mathbf{Y}$ on the sets of closed nulls.*

▶ **Corollary 43.** *Let $A[\mathbf{X}]$ and $B[\mathbf{Y}]$ be two annotation-minimal instances. Then $A[\mathbf{X}] \equiv B[\mathbf{Y}]$ if and only if there exists RCN-covers $\{f_i : A[\mathbf{X}] \to B[\mathbf{Y}] | 1 \leq i \leq n\}$ and $\{g_j : B[\mathbf{Y}] \to A[\mathbf{X}] | 1 \leq j \leq m\}$ such that f_i restricts to a bijection $f_i \restriction_{\mathbf{X}} : \mathbf{X} \to \mathbf{Y}$ and g_j to a bijection $g_j \restriction_{\mathbf{Y}} : \mathbf{Y} \to \mathbf{X}$.*

▶ **Corollary 44.** *Let $A[\mathbf{X}]$ and $B[\mathbf{Y}]$ be two annotation-minimal instances. Then $A[\mathbf{X}] \equiv B[\mathbf{Y}]$ if and only if $\mathbf{X}$ is of the same length as $\mathbf{Y}$ and there exists injective functions $f : \mathbf{X} \to \mathsf{Const}$ and $g : \mathbf{Y} \to \mathsf{Const}$ such that $A[f(\mathbf{X})/\mathbf{X}] \equiv_{\mathsf{PCWA}} B[g(\mathbf{Y})/\mathbf{Y}]$, where $f(\mathbf{X})$ and $g(\mathbf{Y})$ are disjoint from the constants in $A[\mathbf{X}]$ and $B[\mathbf{Y}]$.*

That is to say, $A[\mathbf{X}]$ and $B[\mathbf{Y}]$ are equivalent if there is a way to "freeze" the closed nulls so that they become PCWA-equivalent.

Now, let $A[\mathbf{X}]$ be an annotation-minimal instance. Let $\mathbf{c}$ be a list of fresh constants, of the same length as $\mathbf{X}$. Then we can compute $\varpi(A[\mathbf{c}]) = \{E_1, \ldots, E_n\}$ and $\chi_p(A[\mathbf{c}]) = E_1 \cup \ldots \cup E_n$, as in Section 6, and then substitute $\mathbf{X}$ back in for $\mathbf{c}$. This yields a set $\varpi(A[\mathbf{X}]) := \{E_1[\mathbf{X}], \ldots, E_n[\mathbf{X}]\}$ of annotated instances and an annotated instance $\chi_p(A)[\mathbf{X}] := E_1[\mathbf{X}] \cup \ldots \cup E_n[\mathbf{X}]$. Since $\chi_p(A)[\mathbf{X}]$ is semantically equivalent to $A[\mathbf{X}]$ and has the same (number of) closed nulls, $\chi_p(A)[\mathbf{X}]$ is annotation-minimal. Thus we have a function $\varpi(-)$ from annotation minimal annotated instances to finite sets of annotated instances, and $\chi_p(-)$ from annotation minimal instances to annotation minimal instances. As in Section 6, we identify $\varpi(A[\mathbf{X}])$ and $\varpi(B[\mathbf{Y}])$ if they "are the same up to renaming of nulls", but in the presence of closed nulls we have to add a condition to what this means: we say that $\varpi(A[\mathbf{X}]) = \varpi(B[\mathbf{Y}])$ if there is a bijection of sets F between them; an isomorphism $f_E : E \to F(E)$ for each $E \in \varpi(A[\mathbf{X}])$; and for all $E, E' \in \varpi(A[\mathbf{X}])$, the homomorphisms f_E and $f_{E'}$ restrict to one and the same bijection of sets $\mathbf{X} \to \mathbf{Y}$. We now have:

▶ **Theorem 45.** *Let $A[\mathbf{X}]$, $B[\mathbf{Y}]$ be an annotation-minimal instances. Then:*

1. *$A[\mathbf{X}] \equiv B[\mathbf{Y}]$ if and only if $\varpi(A[\mathbf{X}]) = \varpi(B[\mathbf{Y}])$;*
2. *$A[\mathbf{X}] \equiv \chi_p(A[\mathbf{X}])$, and $\chi_p(A[\mathbf{X}])$ is annotation-minimal;*
3. *if $E[\mathbf{X}] \in \varpi(A[\mathbf{X}])$ and $A[\mathbf{X}] \equiv B[\mathbf{Y}]$ then $E[\mathbf{X}]$ is a reflective subinstance of $B[\mathbf{Y}]$ (up to annotation-preserving isomorphism); and*
4. *if $A[\mathbf{X}] \equiv B[\mathbf{Y}]$ then $B[\mathbf{Y}]$ is sub-minimal only if there is a strongly surjective homomorphism*
   $\chi_p(A)[\mathbf{X}] \twoheadrightarrow B[\mathbf{Y}]$ restricting to a bijection $\mathbf{X} \to \mathbf{Y}$.

Accordingly, $\varpi(-)$ is representative in the sense of (1) and (2) and minimal in the sense of (3). $\chi_p(-)$ bounds the number of sub-minimal equivalent instances by (4). $\varpi(-)$ (and $\chi_p(-)$) can be extended to all annotated instances by first choosing an equivalent annotation minimal instance and then applying $\varpi(-)$, and (1) ensures that the result does not depend on the choice.

8 Discussion and Conclusion

In this work we study the problems of implication, equivalence, and minimality (and consequently cores) in mixed open and closed worlds. These problems have particular importance in the context of date exchange and remain open for several variants of mixed worlds. In particular, we adress these problems for the Closed Powerset semantics and the OCWA semantics. To this end, we define a novel semantics for mixed worlds that we called OCWA* and subsumes both Closed Powerset and OCWA. Our semantics is introduced with the help of homomorphic covers and it is characterised in terms of such covers. For the minimization problem we presented negative results for several common notions of minimality. Then, we showed that one can find cores using a different notion of minimality.

Observe that homomorphic covers have been already used in several related contexts. In [15], Grahne et al. uses homomorphic covers in the context of source instance recovery in data exchange. In [6], Chaudhuri and Vardi give the existence of a cover as a sufficient condition for conjunctive query containment under bag semantics. In [22], Kostylev et al. use various notions of cover to study annotated query containment. On the other hand, Knauer and Ueckerdt [20] apply this notion to coverage relations between graphs.

In our opinion several more data management scenarios can benefit from the concept of homomorphic cover and the machinery that we have developed for it. For instance, two conjunctive queries whose relational structures cover each other retrieve the same tuples from every relation of any database instance, a fact of potential relevance in e.g. data privacy settings. In the field of constraint programming, this property is closely connected to the notion of a minimal constraint network [14], and may have applications there. For another example, treating one conjunctive query as a view, it can be used to completely rewrite another if there exists a cover from the view (cf. [23]). Thus in this setting, cover-equivalence corresponds to mutual complete rewritability.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

3 Marcelo Arenas, Elena Botoeva, Diego Calvanese, and Vladislav Ryzhikov. Knowledge base exchange: The case of OWL 2 QL. *Artif. Intell.*, 238:11–62, 2016.

4 Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2):7:1–7:72, 2008.

5 Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, pages 1–12, 2007.

6 Surajit Chaudhuri and Moshe Y. Vardi. Optimization of Real Conjunctive Queries. In *PODS*, 1993.

7 Ronald Fagin, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236, 2009.

8 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

9 Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

10 Jan Foniok. *Homomorphisms and Structural Properties of Relational Systems*. PhD thesis, Charles University in Prague, 2007.

11 Henrik Forssell, Evgeny Kharlamov, and Evgenij Thorstensen. Towards Characterising Data Exchange Solutions in Open and Closed Words (Extended Abstract). In *AMW*, 2017.

12 Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. NaïVe Evaluation of Queries over Incomplete Databases. *ACM Trans. Database Syst.*, 39(4):31:1–31:42, December 2014.

13 Ladan Golshanara and Jan Chomicki. Temporal data exchange. *Inf. Syst.*, 87, 2020.

14 Georg Gottlob. On minimal constraint networks. *Artif. Intell.*, 191-192:42–60, 2012.

15 Gösta Grahne, Ali Moallemi, and Adrian Onet. Recovering Exchanged Data. In *PODS*, pages 105–116, 2015.

16 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1):117–126, 1992.

17 André Hernich. Answering Non-monotonic Queries in Relational Data Exchange. In *ICDT*, pages 143–154, 2010.

18 André Hernich, Leonid Libkin, and Nicole Schweikardt. Closed world data exchange. *ACM Trans. Database Syst.*, 36(2):14:1–14:40, 2011.

19 Angelika Kimmig, Alex Memory, Renée J. Miller, and Lise Getoor. A Collective, Probabilistic Approach to Schema Mapping Using Diverse Noisy Evidence. *IEEE Trans. Knowl. Data Eng.*, 31(8):1426–1439, 2019.

20 Kolja Knauer and Torsten Ueckerdt. Three ways to cover a graph. *Discrete Mathematics*, 339(2):745–758, 2016.

21 Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, 2005.

22 Egor V. Kostylev, Juan L. Reutter, and András Z. Salamon. Classification of annotation semirings over containment of conjunctive queries. *ACM Trans. Database Syst.*, 39(1):1, 2014.

23 Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering Queries Using Views. In *PODS*, pages 95–104, 1995.

24 Leonid Libkin and Cristina Sirangelo. Data exchange and schema mappings in open and closed worlds. *J. Comput. Syst. Sci.*, 77(3):542–571, 2011.

25 Thomas Lukasiewicz, Maria Vanina Martinez, Livia Predoiu, and Gerardo I. Simari. Basic Probabilistic Ontological Data Exchange with Existential Rules. In *AAAI*, pages 1023–1029, 2016.

26 Lucian Popa, Yannis Velegrakis, Renée J. Miller, Mauricio A. Hernández, and Ronald Fagin. Translating Web Data. In *VLDB*, 2002.

27 Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. Approximation Algorithms for Schema-Mapping Discovery from Data Examples. *ACM Trans. Database Syst.*, 42(2):12:1–12:41, 2017.

28 Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. Active Learning of GAV Schema Mappings. In *PODS*, pages 355–368, 2018.

Dynamic Complexity of Document Spanners

Dominik D. Freydenberger
Loughborough University, Loughborough, United Kingdom

Sam M. Thompson
Loughborough University, Loughborough, United Kingdom

Abstract

The present paper investigates the dynamic complexity of document spanners, a formal framework for information extraction introduced by Fagin, Kimelfeld, Reiss, and Vansummeren (JACM 2015). We first look at the class of regular spanners and prove that any regular spanner can be maintained in the dynamic complexity class DynPROP. This result follows from work done previously on the dynamic complexity of formal languages by Gelade, Marquardt, and Schwentick (TOCL 2012).

To investigate core spanners we use SpLog, a concatenation logic that exactly captures core spanners. We show that the dynamic complexity class DynCQ is more expressive than SpLog and therefore can maintain any core spanner. This result is then extended to show that DynFO can maintain any generalized core spanner and that DynFO is more powerful than SpLog with negation.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Information systems → Information extraction

Keywords and phrases Document spanners, information extraction, dynamic complexity, descriptive complexity, word equations

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.11

Related Version A full version of this paper is avaliable at [9], `https://arxiv.org/abs/1909.10869`.

Acknowledgements The authors would like to thank the anonymous reviewers as well as Thomas Schwentick for their helpful comments and suggestions. The authors would also like to thank Thomas Zeume for clarifying a result from his thesis.

1 Introduction

Document spanners where introduced by Fagin, Kimelfeld, Reiss, and Vansummeren [4] as a formalization of IBM's information retrieval language AQL. Essentially, they can be explained as a formalism of querying text like one would query a relational database.

The universe of document spanners are *spans*, intervals of positions in a text. For example, if one searches for a word inside a larger text, every match can be understood as being one span inside the text. Spanners generalize this by mapping an input text to a table of spans.

More specifically, the process can be described as follows. First, *primitive spanners*, so-called *extractors*, are used to convert the input text into tables of spans. These extractors can be assumed to be *regex formulas*, which are regular expressions with variables. The tables can then be combined using relational algebra. As one might expect, different types of spanners allow different choices of operators. In this paper, we deal with three types of spanners that were introduced by Fagin et al. [4]. *Regular spanners*, currently the most widely studied in literature, allow the operators $\cup$ (union), π (projection), and $\bowtie$ (join). *Core spanners* extend regular spanners by allowing the string equality selection operator $\zeta^{=}$, which allows one to check whether spans describe the same string (but potentially at different places). *Generalized core spanners* then extend these with the set difference $\setminus$.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 11; pp. 11:1–11:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the last few years, various aspects of spanners have received considerable attention (see our related work section). The main focus was on evaluation and enumeration of results. But very few papers have considered aspects of maintaining the results of spanners under updates on the input text, and these have only focused on regular spanners.

In this paper, we examine the complexity of this problem from a *dynamic complexity* point of view. The classic dynamic complexity setting was independently introduced by Dong, Su, and Topor [3] and Patnaik and Immerman [16]. The "default setting" of dynamic complexity assumes a big relational database that is constantly changing (where the updates consist of adding or removing tuples from relations). The goal is then to maintain a set of auxiliary relations that can be updated with "little effort". As this is a descriptive complexity point of view, little effort is defined as using only first-order formulas. The class of all problems that can be maintained in this way is called DynFO.

A more restricted setting is DynPROP, where only quantifier-free formulas can be used. As one might expect, restricting the update formulas leads to various classes between DynPROP and DynFO. Of particular interest to this paper are DynCQ and DynUCQ, where the update formulas are conjuctive queries or unions of conjunctive queries. As shown by Zeume and Schwentick [21], DynCQ = DynUCQ holds; but it is open whether these are proper subclasses of DynFO (see Zeume [20] for detailed background information).

As document spanners are defined on words, we adapt the dynamic complexity setting for formal languages by Gelade, Marquardt, and Schwentick [10]. This interprets a word structure as a linear order (of positions in the word) with unary predicates for every terminal symbol. To account for the dynamic complexity setting, positions can be undefined, and the update operations are setting a position to a symbol (an insertion or a symbol change) and resetting a position to undefined (deleting a symbol).

We show that in this setting, regular spanners can be maintained in DynPROP, core spanners in DynUCQ (and, hence, by [21] in DynCQ), and generalized core spanners in DynFO. Here, the second of these results is the main result of the present paper (the third follows directly from it, and the first almost immediately from [10]). To achieve it, we do not convert core spanners directly, but use the concatenation logic SpLog as an intermediate model.

SpLog (short for *spanner logic*) was introduced by Freydenberger [6] and has the same expressive power as core spanners (under some caveats that we discuss in Section 2.2). An additional benefit of the main result is that SpLog can be used to simplify proofs that languages or word relations can be maintained in DynCQ.

Related work. Recently, algorithmic and complexity theoretic aspects of evaluation and enumeration of spanners have received a considerable amount of attention, see [1, 5, 7, 8, 6, 12, 13, 14, 17, 18]. But these almost exclusively consider spanners in a static setting. To the authors' knowledge, the only articles to also examine updates are Losemann [12] and Amarilli, Bourhis, Mengel, and Niewerth [1]. Both do not take a DynFO point of view; moreover, both only deal with regular spanners and there is no obvious way to also include the string equalities that are required for core spanners and generalized core spanners.

Doleschal, Kimelfeld, Martens, Nahshon, and Neven [2] introduce the notion of split-correctness. Without going into details, this examines spanners for which it is possible to split the input word into subwords on which the spanner is then evaluated. This can be viewed as a special case of update, but again was restricted to regular spanners.

Gelade, Marquardt, and Schwentick [10] examined the dynamic complexity of formal languages. Their result that DynPROP captures the regular languages is the basis for Proposition 3.1 in the current paper. While they also established that every context free language is in DynFO and that every Dyck-language is in DynQF (DynPROP with auxiliary functions), they did not examine DynUCQ and DynCQ, which the present paper does.

Muñoz, Vortmeier, and Zeume [15] studied the dynamic complexity in a graph database setting, namely for *conjunctive regular path queries (CRQPs)* and *extended conjunctive regular path queries (ECRPQs)*. In particular, Theorem 14 in [15] states that on acyclic graphs, even a generalization of ECRPQs can be maintained in DynFO. Fagin et al. [4] established that on marked paths (a certain type of graph) core spanners have the same expressive powers as a CRPQs with string equalities (a fragment of ECRPQs). While marked paths are not acyclic in a strict sense, Section 7 of [6] proposes a variant of this model that could be directly combined with the construction from [15]. Thus, one could combine these results and observe that core spanners can be maintained in DynFO. In contrast to this, the present paper allows us to lower the upper bound to DynCQ. Moreover, if one is satisfied with DynUCQ, the constructions in the present paper also guarantee that all auxiliary relations only contain active nodes (nodes which carry a letter) of the word-structure, the only exception being the special case where the word-structure represents the empty string.

Structure of the paper. Section 2 contains the central definitions. Section 3 establishes dynamic upper bounds for the three central classes of document spanners (regular, core, and generalized core spanners), in particular the main result (Theorem 3.13). Section 4 further examines the relative expressive powers of core spanners and DynCQ. Section 5 concludes the paper. Some of the longer proofs have been omitted, see the full version for those proofs [9].

2 Preliminaries

Let $\mathbb{N} := \{0, 1, 2 \ldots\}$ and let $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$, where $\setminus$ denotes set difference. We write $|S|$ to represent the *cardinality* of a set S. We use $\subseteq$ for subset and $\subset$ for proper subset. We denote the powerset of S by $\mathcal{P}(S)$. Let $\emptyset$ be the empty set. If R is a relation of arity 0, then R is the empty set, or R is the set containing the empty tuple. We define $[n] := \{1, 2 \ldots n\}$.

Let A be an alphabet[1]. We write $|w|$ to denote the length of a word $w \in A^*$. The number of occurrences of some $a \in A$ in a word $w \in A^*$ is represented by $|w|_a$. We use ε to denote the empty word. Given two words $u \in A^*$ and $v \in A^*$, we write $u \cdot v$, or simply uv for concatenation. If $w = v_1 u v_2$ where $v_1 \in A^*$ and $v_2 \in A^*$, then u is a *subword* of w. We use $\sqsubseteq$ for subword and $\sqsubset$ for the proper subword relation. If u is not a subword of w, we write $u \not\sqsubseteq w$. Let Σ be a finite alphabet of so-called *terminal symbols*. Let Ξ be an infinite set of so-called *variables*, which is disjoint from Σ. Let $\mathcal{L}(A)$ (or $\mathcal{L}(\alpha)$) denote the language of a nondeterministic finite automaton (NFA) A (or of a regular expression α).

The rest of this section is structured as follows: First, we define various types of document spanners in Section 2.1 and equivalent logics (Section 2.2). After that, we define dynamic complexity, with a particular focus on its application to document spanners (Section 2.3).

[1] We use A here as a generic alphabet since we look at both the alphabet of terminal symbols and the alphabet of variables, and the concepts defined here apply to both.

2.1 Document Spanners and Spanner Algebra

In this section, we introduce document spanners and their representations. We begin with *primitive spanners* (Section 2.1.1) and then combine these to *spanner algebras* (Section 2.1.2).

2.1.1 Primitive Spanner Representations

Let $w := a_1 \cdot a_2 \cdots a_n$ be a word, where $n \geq 0$ and $a_1, \ldots, a_n \in \Sigma$. A *span* of w is an interval $[i, j\rangle$ with $1 \leq i \leq j \leq n + 1$ and $i, j \geq 0$. Given a span $[i, j\rangle$ of a word w, we define the subword $w_{[i,j\rangle}$ as $a_i \cdot a_{i+1} \cdots a_{j-1}$.

► **Example 2.1.** Consider the word $w := \mathtt{banana}$. As $|w| = 6$, the spans of w are the $[i, j\rangle$ with $1 \leq i \leq j \leq 7$. For example, we have $w_{[1,2\rangle} = \mathtt{b}$ and $w_{[2,4\rangle} = w_{[4,6\rangle} = \mathtt{an}$. Although both spans describe the same subword $\mathtt{an}$, the two occurrences are at different locations (and, thus, at different spans). Analogously, we have $w_{[1,1\rangle} = w_{[2,2\rangle} = \cdots = w_{[7,7\rangle} = \varepsilon$, but $[i, i\rangle \neq [i', i'\rangle$ for all distinct $1 \leq i, i' \leq 7$.

Let $V \subseteq \Xi$ and $w \in \Sigma^*$. A (V, w)*-tuple* is a function μ that maps each $x \in V$ to a span $\mu(x)$ of w. A set of (V, w)-tuples is called a (V, w)*-relation*. A *spanner* P is a function that maps every $w \in \Sigma^*$ to a (V, w)-relation $P(w)$. We write $\mathsf{SVars}\,(P)$ to denote the set of variables V of a spanner P. Two spanners P_1 and P_2 are *equivalent* if $\mathsf{SVars}\,(P_1) = \mathsf{SVars}\,(P_2)$ and $P_1(w) = P_2(w)$ holds for all $w \in \Sigma^*$.

In the usual applications of spans and spanners, the word w is some type of text. Hence, we can view a spanner P as mapping an input text w to a (V, w)-relation $P(w)$, which can be understood as a table of spans of w.

To define spanners, we use two types of *primitive spanner representations*, the so-called *regex formulas* and *variable-set automata*. Both extend classical mechanisms for regular languages (regular expressions and NFAs, respectively) with variables.

Regex formulas: The syntax of regex formulas is defined by the following $\alpha := \emptyset \mid \varepsilon \mid a \mid (\alpha \vee \alpha) \mid (\alpha \cdot \alpha) \mid (\alpha)^* \mid x\{\alpha\}$, where $a \in \Sigma$ and $x \in \Xi$. We use α^+ to denote $\alpha \cdot \alpha^*$.

Like [6], we define the semantics of regex formulas using two step-semantics with *ref-words* (originally introduced by Schmid [19] in a different context). A ref-word is a word over the extended alphabet $(\Sigma \cup \Gamma)$ where $\Gamma := \{\vdash_x, \dashv_x \mid x \in \Xi\}$. The symbols $\vdash_x$ and $\dashv_x$ represent the beginning and end of the span for the variable x. The first step in the definition of semantics is treating each regex formula α as generators of languages of ref-words $\mathcal{R}(\alpha) \subseteq (\Sigma \cup \Gamma)^*$, which is defined by $\mathcal{R}(\emptyset) := \emptyset$, $\mathcal{R}(a) := \{a\}$ where $a \in \Sigma \cup \{\varepsilon\}$, $\mathcal{R}(\alpha_1 \vee \alpha_2) := \mathcal{R}(\alpha_1) \cup \mathcal{R}(\alpha_2)$, $\mathcal{R}(\alpha_1 \cdot \alpha_2) := \mathcal{R}(\alpha_1) \cdot \mathcal{R}(\alpha_2)$, $\mathcal{R}(\alpha^*) := \mathcal{R}(\alpha)^*$, and $\mathcal{R}(x\{\alpha\}) := \vdash_x \mathcal{R}(\alpha) \dashv_x$.

Let $\mathsf{SVars}\,(\alpha)$ be the set of all $x \in \Xi$ such that $x\{\}$ occurs somewhere in α. A ref-word $r \in \mathcal{R}(\alpha)$ is *valid* if for all $x \in \mathsf{SVars}\,(\alpha)$, we have that $|r|_{\vdash_x} = 1$. We denote the set of valid ref-words in $\mathcal{R}(\alpha)$ as $\mathsf{Ref}(\alpha)$ and say that a regex formula is *functional* if $\mathcal{R}(\alpha) = \mathsf{Ref}(\alpha)$. We write RGX for the set of all functional regex formulas. By definition, for every $\alpha \in \mathsf{RGX}$, every $r \in \mathsf{Ref}(\alpha)$, and every $x \in \mathsf{SVars}\,(\alpha)$, there is a unique factorization $r = r_1 \vdash_x r_2 \dashv_x r_3$.

This allows us to define the second step of the semantics, which turns such a factorization for some variable x into a span $\mu(x)$. To this end, we define a morphism $\mathsf{clr}\colon (\Sigma \cup \Gamma)^* \to \Sigma^*$ by $\mathsf{clr}(a) := a$ for $a \in \Sigma$ and $\mathsf{clr}(g) = \varepsilon$ for all $g \in \Gamma$. For a factorization $r = r_1 \vdash_x r_2 \dashv_x r_3$, $\mathsf{clr}(r_1)$ is the substring of w that appears before $\mu(x)$ and $\mathsf{clr}(r_2)$ is the substring $w_{\mu(x)}$.

We use this for the definition of the semantics as follows: For $\alpha \in \mathsf{RGX}$ and $w \in \Sigma^*$, let $V := \mathsf{SVars}\,(\alpha)$ and (more importantly) $\mathsf{Ref}(\alpha, w) := \{r \in \mathsf{Ref}(\alpha) \mid \mathsf{clr}(r) = w\}$.

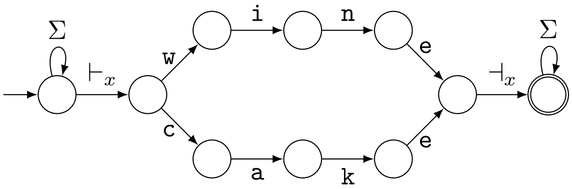
Every $r \in \mathsf{Ref}(\alpha, w)$ defines a (V, w)-tuple μ^r in the following way: For every $x \in \mathsf{SVars}(\alpha)$, we use the unique factorization $r = r_1 \vdash_x r_2 \dashv_x r_3$ to define $\mu^r(x) := [|\mathsf{clr}(r_1)|+1, |\mathsf{clr}(r_1 r_2)|+1\rangle$. The spanner $[\![\alpha]\!]$ is then defined by $[\![\alpha]\!](w) := \{\mu^r \mid r \in \mathsf{Ref}(\alpha, w)\}$ for all $w \in \Sigma^*$.

Variable-set automata: Variable-set automata (short: *vset-automata*) are NFAs that may use variable operations $\vdash_x$ and $\dashv_x$ as transitions. More formally, let $V \subset \Xi$ be a finite set of variables. A variable-set automaton over Σ with variables V is a tuple $A = (Q, q_0, q_f, \delta)$, where Q is the set of states, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the accepting state, and $\delta \colon Q \times (\Sigma \cup \{\varepsilon\} \cup \Gamma_V) \to \mathcal{P}(Q)$ is the transition function with $\Gamma_V := \{\vdash_x, \dashv_x \mid x \in V\}$.

We define the semantics using a two-step approach analogous to the semantic definition of regex formulas. Firstly, we treat A as an NFA that defines the ref-language defined by $\mathcal{R}(A) := \{r \in (\Sigma \cup \Gamma_V)^* \mid q_f \in \delta^*(q_0, r)\}$, where the function $\delta^* \colon Q \times (\Sigma \cup \Gamma_V) \to \mathcal{P}(Q)$ is defined such that for all $p, q \in Q$ and $r \in (\Sigma \cup \Gamma_V)^*$, $q \in \delta^*(p, r)$ if and only if there exists a path in A from p to q with the label r.

Secondly, let $\mathsf{SVars}(A)$ be the set of $x \in V$ such that $\vdash_x$ or $\dashv_x$ appears in A. A ref-word $r \in \mathcal{R}(A)$ is *valid* if for every $x \in \mathsf{SVars}(A)$, $|r|_{\vdash_x} = |r|_{\dashv_x} = 1$, and $\vdash_x$ always occurs to the left of $\dashv_x$. Then $\mathsf{Ref}(A)$, $\mathsf{Ref}(A, w)$ and $[\![A]\!]$ are defined analogously to regex formulas. We denote the set of all vset-automata using $\mathsf{VA_{set}}$. As for regex formulas, a vset-automaton $A \in \mathsf{VA_{set}}$ is called *functional* if $\mathcal{R}(A) = \mathsf{Ref}(A)$.

▶ **Example 2.2.** We define the functional regex formula $\alpha := \Sigma^* \cdot x\{(\texttt{wine}) \vee (\texttt{cake})\} \cdot \Sigma^*$. We also define the functional vset-automaton A as follows:



For all $w \in \Sigma^*$, we have that $[\![\alpha]\!](w) = [\![A]\!](w)$ contains exactly those $(\{x\}, w)$-tuples μ that have $w_{\mu(x)} = \texttt{wine}$ or $w_{\mu(x)} = \texttt{cake}$.

2.1.2 Spanner Algebra

We now introduce an algebra on spanners in order to construct more complex spanners.

▶ **Definition 2.3.** *Two spanners P_1 and P_2 are* compatible *if $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$. We define the following algebraic operators for all spanners P, P_1, P_2:*

- *If P_1 and P_2 are compatible, their* union *$(P_1 \cup P_2)$ and their* difference *$(P_1 \setminus P_2)$ are defined by $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$ and $(P_1 \setminus P_2)(w) := P_1(w) \setminus P_2(w)$.*
- *The* projection *$\pi_Y P$ for $Y \subseteq \mathsf{SVars}(P)$ is defined by $\pi_Y P(w) := P|_Y(w)$, where $P|_Y(w)$ is the restriction of all $\mu \in P(w)$ to Y.*
- *The* natural join *$P_1 \bowtie P_2$ is obtained by defining each $(P_1 \bowtie P_2)(w)$ as the set of all $(V_1 \cup V_2, w)$-tuples μ for which there exists $\mu_1 \in P_1(w)$ and $\mu_2 \in P_2(w)$ with $\mu|_{V_1}(w) = \mu_1(w)$ and $\mu|_{V_2}(w) = \mu_2(w)$, where $V_i := \mathsf{SVars}(P_i)$ for $i \in \{1, 2\}$.*
- *For every k-ary relation $R \subseteq (\Sigma^*)^k$ and variables $x_1, \ldots, x_k \in \mathsf{SVars}(P)$, the* selection *$\zeta^R_{x_1 \ldots x_k} P$ is defined by $\zeta^R_{x_1 \ldots x_k} P(w) := \{\mu \in P(w) \mid (w_{\mu(x_1)}, \ldots, w_{\mu(x_k)}) \in R\}$ for $w \in \Sigma^*$.*

Let $\mathsf{SVars}(P_1 \cup P_2) := \mathsf{SVars}(P_1 \setminus P_2) := \mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$, $\mathsf{SVars}(\pi_Y P) := Y$, $\mathsf{SVars}(P_1 \bowtie P_2) := \mathsf{SVars}(P_1) \cup \mathsf{SVars}(P_2)$, and $\mathsf{SVars}(\zeta^R_{x_1 \ldots x_k}) := \mathsf{SVars}(P)$.

Note that the relations R in the selection are usually infinite; and they are never considered part of the input.

Let O be a spanner algebra and let C be a class of primitive spanner representations, then we use C^O to denote the set of all spanner representations that can be constructed by repeated combinations of the symbols for the operators from O with the spanner representation from C. We denote the closure of $[\![C]\!]$ under the spanner operators O as $[\![C^O]\!]$.

▶ **Example 2.4.** Let $\alpha_1 := \Sigma^* x\{\Sigma^*\}\Sigma^* y\{\Sigma^*\}\Sigma^*$ and $\alpha_2 := \Sigma^* \cdot x\{(\texttt{wine}) \vee (\texttt{cake})\} \cdot \Sigma^*$ (recall Example 2.2). We combine the two regex formulas into a core spanner $P := \pi_x \zeta^{=}_{x,y}(\alpha_1 \bowtie \alpha_2)$. Then $[\![P]\!](w)$ contains all $(\{x\}, w)$-tuples μ such that $w_{\mu(x)}$ is an occurrence of `wine` or `cake` in w that is followed by another occurrence of the same word.

Like Fagin et al. [4], we are mostly concerned with string equality selections $\zeta^{=}$. Following [4, 18], we focus on the class of *regular spanners* $[\![\mathsf{RGX}^{\mathsf{reg}}]\!]$, the class of *core spanners*[2] $[\![\mathsf{RGX}^{\mathsf{core}}]\!]$ and the class of *generalized core spanners* $[\![\mathsf{RGX}^{\mathsf{core}\cup\{\setminus\}}]\!]$, where $\mathsf{reg} := \{\pi, \cup, \bowtie\}$ and $\mathsf{core} := \{\pi, \zeta^{=}, \cup, \bowtie\}$. As shown in [4], we have

$$[\![\mathsf{RGX}^{\mathsf{reg}}]\!] = [\![\mathsf{VA}^{\mathsf{reg}}_{\mathsf{set}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!] \subset [\![\mathsf{RGX}^{\mathsf{core}}]\!] = [\![\mathsf{VA}^{\mathsf{core}}_{\mathsf{set}}]\!] \subset [\![\mathsf{RGX}^{\mathsf{core}\cup\{\setminus\}}]\!] = [\![\mathsf{VA}^{\mathsf{core}\cup\{\setminus\}}_{\mathsf{set}}]\!].$$

In other words, there is a proper hierarchy of regular, core, and generalized core spanners; and for each of the classes, we can choose regex formulas or vset-automata as primitive spanner representations. As shown in [6], functional vset-automata have the same expressive power as vset-automata in general. The size difference can be exponential, but this does not matter for the purpose of the present paper.

2.2 Spanner Logic

In this section, we define SpLog (spanner logic) and relate it to spanners. SpLog is a fragment of $\mathsf{EC}^{\mathsf{reg}}$, the existential theory of concatenation with regular constraints (a logic that is built around the concatenation operator). It was introduced by Freydenberger [6] and has the same expressive power as core spanners; and conversions between both models are possible in polynomial time. To define SpLog, we first introduce *word equations.*

A *pattern* α is a word from $(\Sigma \cup \Xi)^*$. In other words, patterns may contain variables and terminal symbol. A *word equation* is a pair of patterns (η_L, η_R), which are called the *left* and *right* side of the equation, respectively. We usually write a word equation as $\eta_L \doteq \eta_R$. The set of all variables in a pattern α is denoted by $\mathsf{var}(\alpha)$. This is extended to word equations $\eta = (\eta_L, \eta_R)$ by $\mathsf{var}(\eta) := \mathsf{var}(\eta_L) \cup \mathsf{var}(\eta_R)$.

A *pattern substitution* is a morphism $\sigma : (\Sigma \cup \Xi)^* \to \Sigma^*$ such that $\sigma(a) = a$ holds for all $a \in \Sigma$. As every substitution σ is a morphism, we have $\sigma(\alpha_1 \cdot \alpha_2) = \sigma(\alpha_1) \cdot \sigma(\alpha_2)$ for all patterns α_1 and α_2. Hence, to define σ, it suffices to define $\sigma(x) \in \Sigma^*$ for all $x \in \Xi$.

The main idea of SpLog is choosing a special main variable W that shall correspond to the input string of a spanner. SpLog is then an existential-positive logic over words, where the atoms are regular predicates or word equations of the form $\mathsf{W} \doteq \eta_R$. Formally, we define syntax and semantics as follows:

▶ **Definition 2.5.** *Let* $\mathsf{W} \in \Xi$. *Then* $\mathsf{SpLog}(\mathsf{W})$*, the set of all SpLog-formulas with main variable* W*, is defined recursively as containing the following formulas:*
B1. $(\mathsf{W} \doteq \eta_R)$ *for every* $\eta_R \in (\Xi \cup \Sigma)^*$*.*

[2] As this class captures the core functionality of SystemT.

R1. *$(\varphi_1 \wedge \varphi_2)$ for all $\varphi_1, \varphi_2 \in \mathsf{SpLog}(\mathsf{W})$.*
R2. *$(\varphi_1 \vee \varphi_2)$ for all $\varphi_1, \varphi_2 \in \mathsf{SpLog}(\mathsf{W})$ with $\mathsf{free}(\varphi_1) = \mathsf{free}(\varphi_2)$.*
R3. *$\exists x \colon \varphi$ for all $\varphi \in \mathsf{SpLog}(\mathsf{W})$ and $x \in \mathsf{free}(\varphi) \setminus \{\mathsf{W}\}$.*
R4. *$(\varphi \wedge \mathsf{C}_A(x))$ for every $\varphi \in \mathsf{SpLog}(\mathsf{W})$, every $x \in \mathsf{free}(\varphi)$, and every NFA A.*

Let $\mathsf{free}(\varphi)$ be $\mathsf{free}(\eta) := \mathsf{var}(\eta)$, $\mathsf{free}(\varphi_1 \wedge \varphi_2) := \mathsf{free}(\varphi_1 \vee \varphi_2) := \mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2)$, $\mathsf{free}(\exists x \colon \varphi) := \mathsf{free}(\varphi) \setminus \{x\}$, and $\mathsf{free}(\varphi \wedge \mathsf{C}_A(x)) := \mathsf{free}(\varphi)$.

For every pattern substitution σ and every $\varphi \in \mathsf{SpLog}(\mathsf{W})$, we define $\sigma \models \varphi$ as follows:

- *$\sigma \models (\mathsf{W} \doteq \eta_R)$ if $\sigma(\mathsf{W}) = \sigma(\eta_R)$,*
- *$\sigma \models (\varphi_1 \wedge \varphi_2)$ if $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$; and $\sigma \models (\varphi_1 \vee \varphi_2)$ is defined analogously,*
- *$\sigma \models \exists x \colon \varphi$ if $\sigma_{\frac{x}{w}} \models \varphi$ for some $w \in \Sigma^*$, where $\sigma_{\frac{x}{w}}(x) := w$ and $\sigma_{\frac{x}{w}}(y) = \sigma(y)$ if $y \neq x$,*
- *$\sigma \models (\varphi \wedge \mathsf{C}_A(x))$ if $\sigma \models \varphi$ and $\sigma(x) \in \mathcal{L}(A)$.*

Let SpLog be the union of all $\mathsf{SpLog}(\mathsf{W})$ with $\mathsf{W} \in \Xi$. We add and omit parentheses, as long as the meaning remains unambiguous. We also allow constraints of the form $\mathsf{C}_\alpha(x)$, where α is a regular expression. For readability, we use $\varphi(\mathsf{W}; x_1, x_2 \ldots x_k)$ to express that the SpLog-formula φ has the main variable W and free variables $\{x_1, x_2 \ldots x_k\}$. As a convention, assume that no word equation $(\mathsf{W} \doteq \eta_R)$ has the main variable W occur in the right side; that is, that $|\eta_R|_{\mathsf{W}} = 0$ holds.

▶ **Example 2.6.** For the SpLog-formula $\varphi(\mathsf{W}) := \exists x \colon \big((\mathsf{W} \doteq xxx) \wedge \mathsf{C}_{\mathtt{ab}^*}(x)\big)$, we have $\sigma \models \varphi$ if and only if $\sigma(\mathsf{W}) = www$ for some $w \in \mathtt{ab}^*$.

We also extend the definition of SpLog to $\mathsf{SpLog}^\neg$, which we call SpLog *with negation.*

▶ **Definition 2.7.** *Let $\mathsf{W} \in \Xi$. Then $\mathsf{SpLog}^\neg(\mathsf{W})$, the set of $\mathsf{SpLog}^\neg$-formulas with the main variable W, is defined by extending Definition 2.5 with the additional rule that if $\varphi \in \mathsf{SpLog}^\neg(\mathsf{W})$, then $(\neg\varphi) \in \mathsf{SpLog}^\neg(\mathsf{W})$, with $\mathsf{free}(\varphi) = \mathsf{free}(\neg\varphi)$. We define $\sigma \models \neg\varphi$ as:*

- *$\sigma(x) \sqsubseteq \sigma(\mathsf{W})$ for all $x \in \mathsf{free}(\varphi)$, and*
- *$\sigma \models \varphi$ does not hold.*

To compare the expressive power of SpLog and document spanners, we need to overcome the difficulty that the former reasons about words, while the latter reason over positions in an input word. To this end, we use the following notion that was introduced by Freydenberger and Holldack [7] in the context of $\mathsf{EC}^{\mathsf{reg}}$.

▶ **Definition 2.8.** *Let $\varphi \in \mathsf{SpLog}$ with $\mathsf{free}(\varphi) := \{W\} \cup \{x_p, x_c \mid x \in \mathsf{SVars}(P)\}$. Let P be a spanner. Let $[\![\varphi]\!](w)$ denote the set of all σ such that $\sigma \models \varphi$ and $\sigma(\mathsf{W}) = w$. We then say that φ* realizes *P if for all $w \in \Sigma^*$, we have $\sigma \in [\![\varphi]\!](w)$ if and only if $\mu \in P(w)$ where for each $x \in \mathsf{SVars}(P)$ and $[i,j\rangle := \mu(x)$, both $\sigma(x_p) = w_{[1,i\rangle}$ and $\sigma(x_c) = w_{[i,j\rangle}$.*

Intuitively, this definition uses two main ideas: Firstly, the spanner's input word w is represented by the main variable W. Secondly, every spanner variable x is represented by two SpLog-variables x_p and x_c, such that in each (V, w)-tuple μ, we have that x_c contains the actual content $w_{\mu(x)}$ and x_p contains the prefix of w before the start of $\mu(x)$.

As shown in Section 4.1 of [6], under this lens, SpLog has exactly the same expressive power as $[\![\mathsf{RGX}^{\mathsf{core}}]\!]$ (the core spanners), and $\mathsf{SpLog}^\neg$ exactly the same as $[\![\mathsf{RGX}^{\mathsf{core}\cup\{\setminus\}}]\!]$ (the generalized core spanners).

One of the central questions in [4, 6] is which relations R can be added to spanners or SpLog without increasing the expressive power (using ζ^R or a new constraint symbol for R, respectively). This is reflected in the notion of *selectable relations.* A relation $R \subseteq (\Sigma^*)^k$ is called *SpLog-selectable* if for every $\varphi \in \mathsf{SpLog}(\mathsf{W})$ and every sequence $\vec{x} = (x_1, \ldots, x_k)$ of variables with $x_1, \dots, x_k \in \mathsf{free}(\varphi) \setminus \{\mathsf{W}\}$, there is a SpLog-formula $\varphi^R_{\vec{x}}$ with $\mathsf{free}(\varphi) = \mathsf{free}(\varphi^R_{\vec{x}})$, and $\sigma \models \varphi^R_{\vec{x}}$ if and only if $\sigma \models \varphi$ and $(\sigma(x_1), \dots, \sigma(x_k)) \in R$. This is equivalent to the analogously defined notion of core spanner selectable relations, see Section 5.1 of [6] for details. We shall use selectability both in the way to our main result (namely, in Lemma 3.12) and for further observations in Section 4.

2.3 Dynamic Complexity

Our definitions of dynamic complexity are based on the setting of dynamic formal languages as described by Gelade, Marquardt, and Schwentick [10]. In this setting, strings are modeled by a relational structure. Insertions and deletions of symbols can be performed on this structure and (auxiliary) relations are *maintained* by logic formulas, called *update formulas*. We extend this with a predetermined relation which is maintained to hold the result of some spanner performed on the current word. The idea of dynamic complexity, which was introduced by Patnaik and Immerman [16], is to have dynamic descriptive complexity classes based upon the logic needed to maintain a relation, or in our case a spanner. We now formally define these concepts.

Let Σ be a fixed and finite alphabet of terminal symbols. We represent words using a *word-structure*. A word-structure has a fixed and finite set known as the domain $D := [n+1]$ as well as a 2-ary order relation $<$ on D. We use the shorthands $x \leq y$ for $(x < y) \vee (x \doteq y)$. We have in our word-structure the constant \$ which is interpreted by the element $n+1$, the $<$-maximal element of D. This $<$-maximal element marks the end of the word structure and is required for dynamic spanners, which are defined later. For each symbol $\zeta \in \Sigma$ the word-structure has a unary relation $R_\zeta(i)$ and there is at *most* one $\zeta \in \Sigma$ such that $R_\zeta(i)$ for $i \in [n]$. If we have $R_\zeta(i)$ then we write $w(i) = \zeta$, otherwise we write $w(i) = \varepsilon$. If $w(i) \neq \varepsilon$ for some $i \in D$, then we call i a *symbol-element*.

Given a word-structure $\mathcal{W}$, the word that $\mathcal{W}$ represents is denoted by $\mathsf{word}(\mathcal{W})$ and this is defined as $\mathsf{word}(\mathcal{W}) := w(1) \cdot w(2) \cdots w(n)$. Since for some $j \in D$ it could be that $w(j) = \varepsilon$, it follows that the length of the word $\mathsf{word}(\mathcal{W})$ is likely to be less than n. Let $w := \mathsf{word}(\mathcal{W})$, we write $w[i,j]$ to represent the subword $w[i,j] := w(i) \cdot w(i+1) \cdots w(j)$ where $i, j \in D$ such that $i < j$.

We now define the set of *abstract updates* $\Delta := \{\mathsf{ins}_\zeta \mid \zeta \in \Sigma\} \cup \{\mathsf{reset}\}$. A *concrete update* is $\mathsf{ins}_\zeta(i)$ or $\mathsf{reset}(i)$, for some $i \in D \setminus \{\$\}$ and $\zeta \in \Sigma$. The difference between abstract updates and concrete updates is that concrete updates can be *performed* on a word-structure. Given a word-structure with a domain of size n, we use Δ_n to represent the set of possible concrete updates. For some $\partial \in \Delta_n$, we denote the word-structure $\mathcal{W}$ after an update is performed by $\partial(\mathcal{W})$ and this is defined as:

- If $\partial = \mathsf{ins}_\zeta(i)$, then $R_\zeta(i)$ is true and $R_{\zeta'}(i)$ is false for all $\zeta' \in \Sigma$ where $\zeta \neq \zeta'$.
- If $\partial = \mathsf{reset}(i)$ then $R_\zeta(i)$ is false for all $\zeta \in \Sigma$.

All other elements keep the symbol they had before the update. For $k \geq 1$, let $\partial^* := \partial_1, \partial_2, \dots \partial_k$ be a sequence of updates. We use $\partial^*(\mathcal{W})$ as a short hand to represent $\partial_k(\dots(\partial_2(\partial_1(\mathcal{W})))\dots)$. We place the restriction that updates must *change* the string. We do not allow $\mathsf{reset}(i)$ if $w(i) = \varepsilon$ and we do not allow $\mathsf{ins}_\zeta(i)$ if $w(i) = \zeta$.

▶ **Example 2.9.** Given a word-structure $\mathcal{W}$ over the alphabet $\Sigma := \{a, b\}$ with domain $D = [6]$, where $6 = \$$. If we have that $R_a = \{2, 4\}$ and $R_b := \{5\}$, it follows that $\mathsf{word}(\mathcal{W}) = aab$. Performing the operation $\mathsf{ins}_b(1)$ would give us an updated word of $baab$. Say if we then perform $\mathsf{reset}(4)$ on our new word structure, we would have the word bab.

We define the *auxiliary structure* $\mathcal{W}_{aux}$ as a set of relations over the domain of $\mathcal{W}$. A *program state* $\mathcal{S} := (\mathcal{W}, \mathcal{W}_{aux})$ is a word-structure and an auxiliary structure. An *update program* $\vec{P}$ is a finite set of update formulas, which are of the form $\phi^R_{\mathsf{op}}(y; x_1, \ldots, x_k)$. We have an update formula for each $R \in \mathcal{W}_{aux}$ and $\mathsf{op} \in \Delta$. An update, $\mathsf{op}(i)$, performed on $\mathcal{S}$ yields $\mathcal{S}' = (\partial(\mathcal{W}), \mathcal{W}'_{aux})$ where all relations $R' \in \mathcal{W}'_{aux}$ are defined by $R' := \{\vec{j} \mid \bar{\mathcal{S}} \models \phi^R_{op}(i; \vec{j})\}$, where $\vec{j}$ is a k-tuple (where k is the arity of R) and where $\bar{\mathcal{S}} := (\partial(\mathcal{W}), \mathcal{W}_{aux})$.

We use w to denote $\mathsf{word}(\mathcal{W})$ for some word structure $\mathcal{W}$ and we use w' for $\mathsf{word}(\partial(\mathcal{W}))$ where $\partial \in \Delta_n$ is some update performed on $\mathcal{W}$.

Given some $x \in D$ where $w(x) \neq \varepsilon$, we write that $\mathsf{pos}_w(x) = 1$ if for all $x' \in D$ where $x' < x$ we have that $w(x') = \varepsilon$. Let z, y be elements from the domain such that $z < y$ and $w(z) \neq \varepsilon$ and $w(y) \neq \varepsilon$. If for all $x \in D$ where $z < x < y$ we have that $w(x) = \varepsilon$ then $\mathsf{pos}_w(y) = \mathsf{pos}_w(z) + 1$. We write $x \rightsquigarrow_w y$ if and only if $\mathsf{pos}_w(y) = \mathsf{pos}_w(x) + 1$. If it is not the case that $x \rightsquigarrow_w y$ then we write $x \not\rightsquigarrow_w y$.

For every spanner P with $\mathsf{SVars}(P) := \{x_1, x_2 \ldots x_k\}$ and every word-structure $\mathcal{W}$, the spanner relation R^P is a $2k$-ary relation over D where each spanner variable x_i is represented by two components x_i^o and x_i^c. We obtain R^P on $\mathcal{W}$ by converting each $\mu \in P(w)$ into a $2k$-tuple $(x_1^o, x_1^c, x_2^o, x_2^c \ldots x_k^o, x_k^c)$, where for each $i \in [k]$, we have $\mu(x_i) = [\mathsf{pos}_w(x_i^o), \mathsf{pos}_w(x_i^c)\rangle$. The only exception is if $\mu(x_i) = [j, k\rangle$ and $k > |w|$ then $x_i^c = \$$ for such a tuple $(x_1^o, x_1^c, x_2^o, x_2^c \ldots x_k^o, x_k^c)$. In Example 2.11 we give a spanner represented by a regex formula and show the corresponding spanner-relation on a word-structure.

▶ **Definition 2.10.** *A dynamic program is a triple, containing:*

- $\vec{P}$ *- an update program over* $(\mathcal{W}, \mathcal{W}_{aux})$*.*
- INIT *- a first-order initialization program.*
- $R^P \in \mathcal{W}_{aux}$ *- a designated spanner-relation.*

For each $R \in \mathcal{W}_{aux}$, we have some $\psi_R(\vec{j}) \in \mathsf{INIT}$ which defines the initial tuples of R (before any updates to the input structure occur). Note that $\vec{j}$ is a k-tuple where the arity of R is k. For our work ψ_R is a first-order logic formula.

A dynamic program *maintains* a spanner P if we have that $R^P \in \mathcal{W}_{aux}$ always corresponds to $P(\partial^*(\mathcal{W}))$. We can then extend this to saying that we maintain a *relation* if there is a designated $R \in \mathcal{W}_{aux}$ which is always equivalent to some relation where the relation is defined in terms of the input word.

▶ **Example 2.11.** Consider the regex formula $\alpha := \Sigma^* \cdot x\{a \cdot b\} \cdot \Sigma^*$ where $a, b \in \Sigma$ and $x \in \Xi$. Now consider the following word-structure:

1	2	3	4	5	6	\$
a	ε	b	ε	a	ε	ε

Note that the top row is the elements of the domain in order, and the bottom row is the corresponding symbols. If we maintain the *spanner relation* of α, given the word-structure above, we have the relation $R^P \in \mathcal{W}_{aux}$ such that $R^P := \{(1, 5)\}$. Now assume we perform the update $\mathsf{ins}_b(6)$. The word-structure is now in the following state:

1	2	3	4	5	6	\$
a	ε	b	ε	a	b	ε

It must be that $\phi^{R^P}_{\mathsf{ins_b}}(6; x, y)$ updates the relation R^P to $\{(1, 5), (5, \$)\}$ for us to correctly maintain the spanner.

▶ **Definition 2.12.** *DynFO is the class of all relations which can be maintained by update formulas which are defined using first-order logic. DynPROP is a subclass of DynFO where all the update formulas are quantifier-free.*

A first-order formula is a *conjunctive query*, or CQ for short, if it is built up from atomic formulae, conjunction and existential quantification. We also have unions of conjunctive queries, or UCQ for short, which allows for the finite disjunction of conjunctive queries. We therefore have the classes DynCQ and DynUCQ which use conjunctive queries and unions of conjunctive queries as update formulas respectively.

For this work, we assume that the input structure is initially empty and that every auxiliary relation is initialized by some first-order initialization. This is to allow us to use the result from Zeume and Schwentick [21] that DynUCQ = DynCQ. However, in our work we only require a very weak form of initialization and hence if DynUCQ is sufficient, one could define the precise class needed for the precomputation. We do not do this as the dynamic complexity class needed to *maintain* a spanner is the main focus of this work[3].

For the proofs in the present paper, one could change the setting by allowing the insertion of unmarked nodes at any point of the word-structure (with an update to the $<$-relation), given that the word is non-empty. The auxiliary relations in our proofs do not operate on unmarked nodes and do not need to be updated after this. In the same way, we can remove unmarked nodes. However, the present paper does not look at this setting.

3 Core Spanners are in DynCQ

In this section, we first look at the dynamic complexity of regular spanners. We show that any regular spanner can be maintained by a DynPROP program. We then turn our attention to the main result of this paper, that any core spanner can be maintained by a DynCQ program. In doing so, we also show that DynCQ is at least as expressive as SpLog. We then extend this result to show that DynFO is at least as powerful as SpLog with negation, and therefore any generalized core spanner can be maintained in DynFO.

▶ **Proposition 3.1.** *Regular spanners can be maintained in DynPROP.*

Proof. Due to the work done by Fagin et al. [4] we can assume that our vset-automaton is a so called *vset-path union.* We define a vset-path as an ordered sequence of regular deterministic finite automata A_1, A_2, $\dots A_n$ for some $n \in \mathbb{N}$. Each automaton A_i is of the form (Q, q_o, F, δ) where Q is the set of states, $q_0 \in Q$ is the initial state, F is the set of accepting states, and δ is the transition function of the form $\delta\colon Q \times \Sigma \to Q$. We have the extra assumption that each $f \in F$ only has incoming transitions. All automata, $A_1, A_2, \dots A_n$ share the same set of input symbols Σ.

Let A be a vset-path. In A, each automata A_i where $1 < i \leq n$, the initial state for A_i has incoming transitions from each accepting state from the automaton A_{i-1}. These extra transitions between the sequence of automata are labeled, $\vdash_x$ or $\dashv_x$ where $x \in \mathsf{SVars}\,(A)$. We treat the vset-path as a regular vset-automaton and all semantics follow from the definitions in Section 2.1.1. We can assume that A is functional [6].

Any vset-automaton can be represented as a union of vset-paths [4]. Therefore to prove that any regular spanner can be maintained in DynPROP, it is sufficient to prove that we can maintain a spanner represented by a vset-path, since union can be simulated via disjunction.

[3] As helpfully pointed out by one of the anonymous reviewers of this paper.

Let A be a vset-path. From Gelade et al. [10], we know that the following relations can be maintained in DynPROP:

- For any pair of states $p, q \in Q$, $R_{p,q} := \{(i,j) \mid i < j \text{ and } \delta^*(p, w[i+1, j-1]) = q\}$.
- For each state q, $R^I_q := \{i \mid \delta^*(q_0, w[1, j-1]) = q\}$.
- For each state p, $R^F_p := \{j \mid \delta^*(p, [i+1, n]) \in F\}$.

We maintain these relations for the vset-path. Some work is needed to deal with the transitions labeled $\vdash_x$ and $\dashv_x$. Let A_i and A_{i+1} be two sub-automata such that $1 \leq i < n$, where n is the number of sub-automata. Let s_i and s_{i+1} be the starting states for automata A_i and A_{i+1} respectively. Likewise, let F_i and F_{i+1} be the sets of accepting states of A_i and A_{i+1} respectively. The intuition is that if $R_{p,f_i}(x,y)$ where $f_i \in F_i$ holds, then so should $R_{p,s_{i+1}}(x,y)$ since the transition from an accepting state of A_i to the starting state of A_{i+1} is $\vdash_x$ or $\dashv_x$. To achieve this, we have the following update formula for $R_{p,s_{i+1}}$

$$\phi_\partial^{R_{p,s_{i+1}}}(u; x, y) := \bigvee_{f \in F_i} \phi_\partial^{R_{p,f}}(u; x, y).$$

We do the analogous for R^I_q and R^F_p. If $R^I_{f_i}(x)$ holds for any $f_i \in F_i$, then so should $R^I_{s_{i+1}}(x)$. Similarly, if $R^F_{s_{i+1}}(x)$ holds, then so should $R^F_{f_i}(x)$ for all $f_i \in F_i$. To achieve this, we proceed analogously to what was done for $\phi_\partial^{R_{p,s_{i+1}}}(u; x, y)$. We also maintain the 0-ary relation ACC to say whether the word-structure is a member of the language of the vset-path.

We will now give two useful subformulas

$$\psi^{k'} := \bigwedge_{1 \leq i \leq k'} \Big(\bigvee_{\zeta \in \Sigma} (R_\zeta(x^o_i) \wedge {R'}^I_{s_i}(x^o_i) \wedge \bigvee_{\substack{p \in Q, \\ \delta(s_i,\zeta)=p}} ({R'}_{p,s_{i+1}}(x^o_i, x^c_i) \wedge \bigvee_{\zeta_2 \in \Sigma} R_{\zeta_2}(x^c_i))) \Big)$$

and

$$\psi^{\$} := \bigvee_{\zeta \in \Sigma} \big(R_\zeta(x^o_k) \wedge {R'}^I_{s_k}(x^o_k) \wedge {R'}^I_{F_k}(x^c_k) \wedge (x^c_k \doteq \$)\big).$$

We now give the update formula to maintain a vset-path spanner A with variables $\mathsf{SVars}(A) := \{x_1, x_2, \ldots, x_k\}$

$$\phi_\partial^{R^A}(u; x^o_1, x^c_1, \ldots, x^o_k, x^c_k) := \phi_\partial^{\mathsf{ACC}}(u) \wedge \big(\psi^k \vee (\psi^{k-1} \wedge \psi^{\$})\big).$$

Note that, without loss of generality, $R'_{p,q}(x,y)$ is used as a shorthand for $\phi_\partial^{R_{p,q}}(u; x, y)$. ◀

Since Gelade et al. [10] proved that DynPROP maintains exactly the regular languages, it is somewhat unsurprising that we can extend that result to regular spanners. Some work is needed in order to maintain the relation of the spanner, which is why a formal proof of Proposition 3.1 is given.

▶ **Definition 3.2.** *The next symbol relation is defined as* $R_{\mathsf{Next}} := \{(x,y) \in D^2 \mid x \rightsquigarrow_w y\}$.

As stated in Section 2.3, it is known that DynCQ = DynUCQ and therefore to show that a relation can be maintained in DynCQ, it is sufficient to show that the relation can be maintained with UCQ update formulas. We use this to prove many of our results.

▶ **Lemma 3.3.** *The next symbol relation can be maintained in DynCQ.*

To prove Lemma 3.3, we maintain the relations $R_{\mathsf{first}} := \{x \in D \mid \mathsf{pos}_w(x) = 1\}$ and $R_{\mathsf{last}} := \{x \in D \mid \mathsf{pos}_w(y) = |w|\}$. Note that these relations would be undefined for an empty input structure (because $\mathsf{pos}_w(x)$ is undefined). Hence we have that if $|w| = 0$ then $x \in R_{\mathsf{first}}$ if and only if $x = \$$, and $y \in R_{\mathsf{last}}$ if and only if y is the $<$-minimal element. This requires the initialization of $R_{\mathsf{first}} := \{\$\}$ and $R_{\mathsf{last}} := \{1\}$. This is the only initialization required in our work, however the stated *first-order initialization* of auxiliary relations is needed to ensure $\mathsf{DynUCQ} = \mathsf{DynCQ}$.

▶ **Example 3.4.** Consider the following word-structure:

1	2	3	4	5	6	\$
ε	a	b	ε	b	ε	ε

We have that $R_{\mathsf{first}} = \{2\}$ and $R_{\mathsf{last}} = \{5\}$ and $R_{\mathsf{Next}} = \{(2,3),(3,5)\}$.

We will now give an idea for the proof of Lemma 3.3. Let u be the node which is being updated. For insertion, if $x \rightsquigarrow_w y$ and $x < u < y$ then $x \rightsquigarrow_{w'} u \rightsquigarrow_{w'} y$. If $R_{\mathsf{first}}(x)$ and $u < x$, then $R'_{\mathsf{first}}(u)$ and $u \rightsquigarrow_{w'} x$. The analogous is done if $R_{\mathsf{last}}(x)$ and $u > x$. For deletion, if $x \rightsquigarrow_w u \rightsquigarrow_w y$ then $x \rightsquigarrow_{w'} y$. The full proof also looks at when $x \rightsquigarrow_w y$ and $x \rightsquigarrow_{w'} y$ (for example when $u < x$ or when $u > y$). See the full version of the paper for the proof [9].

▶ **Definition 3.5.** *The equal substring relation, R_{eq}, is the set of 4-tuples (x_o, x_c, y_o, y_c) such that $w[x_o, x_c] = w[y_o, y_c]$, $x_c < y_o$, and $w[z] \neq \varepsilon$ for all $z \in \{x_o, x_c, y_o, y_c\}$.*

Less formally, we have that if $(x_o, x_c, y_o, y_c) \in R_{\mathsf{eq}}$ then the word $w[x_o, x_c]$ is equal to the word $w[y_o, y_c]$. For our uses, we do not want these subwords to overlap, hence the constraint $x_c < y_o$. The reason for this will become clear later on when we look at maintaining pattern languages. We also wish that each tuple represents a unique pair of subwords, therefore we have that x_o, x_c, y_o, and y_c each have symbols associated to them.

▶ **Example 3.6.** Consider the following word-structure:

1	2	3	4	5	6	7	8	9	10	\$
a	ε	ε	b	a	ε	b	ε	a	b	ε

The equal substring relation for this structure is $R_{\mathsf{eq}} = \{(1,1,5,5),(1,1,9,9),(4,4,7,7),(4,4,10,10),(5,5,9,9),(7,7,10,10),(1,4,5,7),(1,4,9,10),(4,5,7,9),(5,7,9,10)\}$.

Although $w[3,5] = w[7,9]$, this does not imply $(3,5,7,9) \in R_{\mathsf{eq}}$ because $w[3] = \varepsilon$. We also do not have $(9,10,5,7) \in R_{\mathsf{eq}}$ because $10 > 5$.

▶ **Lemma 3.7.** *The equal substring relation can be maintained in DynCQ.*

We now give a proof idea for Lemma 3.7. There are four main cases for the tuple (x_1, y_1, x_2, y_2) we examine in the full proof.

- Case 1: $w[x_1, y_1] = w[x_2, y_2]$ and $w'[x_1, y_1] \neq w'[x_2, y_2]$.
- Case 2: $w[x_1, y_1] = w[x_2, y_2]$ and $w'[x_1, y_1] = w'[x_2, y_2]$.
- Case 3: $w[x_1, y_1] \neq w[x_2, y_2]$ and $w'[x_1, y_1] = w'[x_2, y_2]$.
- Case 4: $w[x_1, y_1] \neq w[x_2, y_2]$ and $w'[x_1, y_1] \neq w'[x_2, y_2]$.

Where we assume that $y_1 < x_2$. One can see that the main case out of these four is Case 3. One of the interesting sub-cases of Case 3 is illustrated in Figure 1. Here, one can think of the new symbol at node u as a "bridge" between the two equal substrings $w[x_1, v_1]$ and $w[x_2, v_3]$ (which are the word w_1) and the equal substrings $w[v_2, y_1]$ and $w[v_4, y_2]$ (which are the word w_2). Hence, after the update we have that $w'[x_1, y_1] = w'[x_2, y_2]$ even though

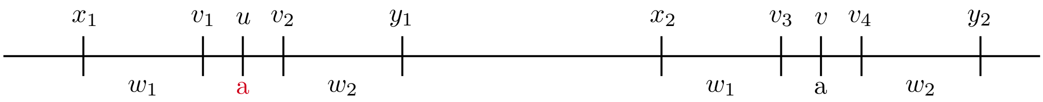


Figure 1 Word after the insertion of the symbol a at node u..

$w[x_1, y_1] \neq w[x_2, y_2]$ (under the assumptions that $w(v) = a$, $v_1 \rightsquigarrow_{w'} u \rightsquigarrow_{w'} v_2$ and that $v_3 \rightsquigarrow_{w'} v \rightsquigarrow_{w'} v_4$). After examining a case like this, one would need to write an update formula to realize it.

The proof of Lemma 3.7 looks through all the cases and produces a UCQ update formula for each. These subformulae are joined together by disjunction to give us an update formula $\phi_\partial^{R_{\mathsf{eq}}}(u)$ which is in DynUCQ, and hence we have proven that we can maintain the equal substring relation in DynCQ. See the full version for the proof [9].

Lemma 3.7 is a central part of the proof of our main result, and some may consider maintaining this relation also to be the most technical aspect of the present paper. This relation will be the main feature of a construction to maintain so-called *pattern languages*, which we then extend with regular constraints to maintain any relations selectable by SpLog.

Given a pattern $\alpha \in (\Sigma \cup \Xi)^+$, we define the *non-erasing* language it generates as $\mathcal{L}_{\mathsf{NE},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma \colon (\Sigma \cup \Xi)^+ \to \Sigma^+ \text{ where } \sigma \text{ is a substitution}\}$. Given the same pattern α, we have $\mathcal{L}_{\mathsf{E},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma \colon (\Sigma \cup \Xi)^+ \to \Sigma^* \text{ where } \sigma \text{ is a substitution}\}$ which is the *erasing* language α generates. Pattern languages are not only used as a part of word equations but also as language generators (see [7] for more details, in particular regarding their relation to document spanners).

▶ **Example 3.8.** Consider $\alpha := axxb$ where $a, b \in \Sigma$ and $x \in \Xi$. Then $ab \in \mathcal{L}_{\mathsf{E},\Sigma}(\alpha)$ with $\sigma(x) = \varepsilon$, but $ab \notin \mathcal{L}_{\mathsf{NE},\Sigma}(\alpha)$. We can also see that $ababab \in \mathcal{L}_{\mathsf{NE},\Sigma}(\alpha)$ and $ababab \in \mathcal{L}_{\mathsf{E},\Sigma}(\alpha)$ using $\sigma(x) = ba$.

We take the definition of maintaining a language from [10]. We can *maintain a language* L if a dynamic program maintains a 0-ary relation which is true if and only if $\mathsf{word}(\mathcal{W}) \in L$.

▶ **Lemma 3.9.** *Every non-erasing pattern language can be maintained in DynCQ.*

Proof. To prove this lemma, we give a way to symbolically construct an update formula to maintain a 0-ary relation $\mathcal{P}$ which updates to true if and only if $w' \in \mathcal{L}_{\mathsf{NE},\Sigma}(\alpha)$ for any specified $\alpha \in (\Sigma \cup \Xi)^+$. Let $|\alpha|$ be the length of the pattern α. Let α_i denote the i^{th} symbol (from Ξ or Σ) of the pattern α where $1 \leq i \leq |\alpha|$. We give the construction in Algorithm 1.

Note that occurrences of R'_{Next} and R'_{eq} in Algorithm 1 are the relations correct *after* the update. To achieve this, we can replace occurrences of $R'_{\mathsf{Next}}(\dots)$ with $\phi_\partial^{R_{\mathsf{Next}}}(\dots)$, where ∂ is the update for which the update formula of $\mathcal{P}$ is being constructed. The equivalent is done for R_{eq}. ◀

▶ **Example 3.10.** Let $\alpha := axbx$ be a pattern such that $a, b \in \Sigma$ and $x \in \Xi$. As stated, we wish to maintain a 0-ary relation $\mathcal{P}$ such that $\mathcal{P}$ is true if and only if $w' \in \mathcal{L}_{\mathsf{NE},\Sigma}(\alpha)$ where w' is our word after some update.

- $\alpha_1 = a$: therefore $\alpha_1 \in \Sigma$ and hence we have $\omega_1 := R_a(t_1) \land R'_{\mathsf{first}}(t_1)$.
- $\alpha_2 = x$: therefore $\alpha_2 \in \Xi$ therefore we have $\omega_2 := R'_{\mathsf{Next}}(t_1, x_2) \land (x_2 \leq t_2) \land \omega_1$.
- $\alpha_3 = b$: therefore $\alpha_3 \in \Sigma$ and hence we have $\omega_3 := R_b(t_3) \land R'_{\mathsf{Next}}(t_2, t_3) \land \omega_2$.
- $\alpha_4 = x$ and $\alpha_4 = \alpha_2$: therefore $\omega_4 := R'_{\mathsf{Next}}(t_3, x_4) \land (x_4 \leq t_4) \land R'_{\mathsf{eq}}(x_2, t_2, x_4, t_4) \land \omega_3$.

Algorithm 1 Pattern Language Update Formula Construction.

```
Input: A pattern $\alpha \in (\Sigma \cup X)^+$.
Output: Update formulas $\phi^{\mathcal{P}}_{\mathsf{ins}_\varsigma}(u)$ and $\phi^{\mathcal{P}}_{\mathsf{reset}}(u)$.
If $\alpha_1 \in \Sigma$ then $\omega_1 := R_{\alpha_1}(t_1) \wedge R'_{\mathsf{first}}(t_1)$;
If $\alpha_1 \in \Xi$ then $\omega_1 := (x_1 \leq t_1) \wedge R'_{\mathsf{first}}(x_1)$;
for $i := 2$ to $|\alpha|$ do
    if $\alpha_i \in \Sigma$ then
        $\omega_i := R_{\alpha_i}(t_i) \wedge R'_{\mathsf{Next}}(t_{i-1}, t_i) \wedge \omega_{i-1}$;
    if $\alpha_i \in \Xi$ then
        if there exists $j \in \mathbb{N}$ where $j < i$ such that $\alpha_i = \alpha_j$ then
            $j_{max}$ := Largest $j$ value such that $j < i$ and $\alpha_i = \alpha_j$;
            $\omega_i := R'_{\mathsf{Next}}(t_{i-1}, x_i) \wedge (x_i \leq t_i) \wedge R'_{\mathsf{eq}}(x_{j_{max}}, t_{j_{max}}, x_i, t_i) \wedge \omega_{i-1}$;
        else
            $\omega_i := R'_{\mathsf{Next}}(t_{i-1}, x_i) \wedge (x_i \leq t_i) \wedge \omega_{i-1}$;
$\omega := \big(\omega_{|\alpha|} \wedge R'_{\mathsf{last}}(t_{|\alpha|})\big)$;
For every occurrence of some $t_i$ in $\omega$, where $i \leq |\alpha|$, add $\exists t_i$ to the front of $\omega$;
For every occurrence of some $x_i$ in $\omega$ add $\exists x_i$ to the front of $\omega$;
$\phi^{\mathcal{P}}_{\mathsf{ins}_\varsigma}(u) := \omega$;   $\phi^{\mathcal{P}}_{\mathsf{reset}}(u) := \omega$;
```

We rearrange the atoms in ω to help with readability, giving us:

$$\omega := R'_{\mathsf{first}}(t_1) \wedge R_a(t_1) \wedge R'_{\mathsf{Next}}(t_1, x_2) \wedge (x_2 \leq t_2) \wedge R'_{\mathsf{Next}}(t_2, t_3) \wedge R_b(t_3) \\ \wedge R'_{\mathsf{Next}}(t_3, x_4) \wedge (x_4 \leq t_4) \wedge R'_{\mathsf{eq}}(x_2, t_2, x_4, t_4) \wedge R'_{\mathsf{last}}(t_4).$$

Hence $\phi^{\mathcal{P}}_{\partial}(u) := \exists t_1, t_2, t_3, t_4, x_2, x_4 \colon (\omega)$ which holds for a word-structure of the form:

$$\begin{array}{ccccccccc} \dots & t_1 & \mathbf{x_2} & \dots & \mathbf{t_2} & t_3 & \mathbf{x_4} & \dots & \mathbf{t_4} & \dots \\ \varepsilon & a & & \dots & & b & & \dots & & \varepsilon \end{array}$$

We have that x_2, t_2, x_4, t_4 are in bold to demonstrate the fact that it must be that $w'[x_2, t_2] = w'[x_4, t_4]$ for $\phi^{\mathcal{P}}_{\partial}(u)$ to hold. Note that t_1 may not be $< -minimal$ and t_4 may not be $< -maximal$, but because $R'_{\mathsf{first}}(t_1)$ and $R'_{\mathsf{last}}(t_4)$ must hold, t_1 and t_4 are the first and last symbol-elements respectively.

One side effect of Lemma 3.9 is that we get the dynamic complexity upper bounds of a class of languages, the pattern languages. Pattern languages were not looked at in [10] and hence this result extends what is known about the dynamic complexity of formal languages.

▶ **Corollary 3.11.** *Every erasing pattern language can be maintained in* DynCQ*.*

Proof. From Jiang et al. [11] it is known that every erasing pattern language is the finite union of non-erasing pattern languages. Therefore, we can create 0-ary relations for each non-erasing pattern language and join them with a disjunction. There is the case where $\varepsilon \in \mathcal{L}_{\mathsf{E},\Sigma}(\alpha)$ which we can deal with using the following: $\exists x \colon (R_{\mathsf{first}}(x) \wedge (x \doteq \$))$. We can do this because $R_{\mathsf{first}} = \{\$\}$ whenever $w = \varepsilon$. ◀

Since we are able to maintain any erasing pattern language in DynCQ, we can extend this result to word-equations in SpLog-formulas. Using this along with the fact that regular languages can be maintained in DynPROP, we can conclude the following:

▶ **Lemma 3.12.** *Any relation selectable in SpLog can be maintained in DynCQ.*

Proof. We prove this lemma using structural induction with the recursive definition of a SpLog formula, given in Definition 2.5.

B1. $(\mathsf{W} \dot{=} \eta_R)$ for every $\eta_R \in (\Xi \cup \Sigma)^*$: Since we are assuming that $\sigma(\mathsf{W}) \in \Sigma^*$ and that η_R does not contain W, we have that $\mathsf{W} \dot{=} \eta_R$ is equivalent to $\sigma(\mathsf{W}) \in \mathcal{L}_{\mathsf{E},\Sigma}(\eta_R)$. We have proven in Corollary 3.11, that we can maintain a 0-ary relation which is true if and only if, given some pattern $\alpha \in (\Xi \cup \Sigma)^*$, the word structure is currently a member of $\mathcal{L}_{\mathsf{E},\Sigma}(\alpha)$. According to the construction which we gave in Lemma 3.9, given a variable $x \in \Xi$, where $x = \alpha_i$, we have two variables $x_i, t_i \in D$ such that the word $w[x_i, t_i]$ represents $\sigma(x)$ for some substitution σ. Removing the existential quantifiers for x_i and t_i allows us to maintain the relation defined by α.

R1. $(\psi_1 \wedge \psi_2)$ for all $\psi_1, \psi_2 \in \mathsf{SpLog}(\mathsf{W})$: Under the assumption that we have update formulas $\phi_\partial^{\psi_1}(u; \vec{v_1})$ and $\phi_\partial^{\psi_2}(u; \vec{v_2})$ for SpLog formulas ψ_1 and ψ_2 respectively, the update formula for $\phi_\partial^{\psi_1 \wedge \psi_2}(u; \vec{v_1} \cup \vec{v_2})$ is $\phi_\partial^{\psi_1}(u; \vec{v_1}) \wedge \phi_\partial^{\psi_2}(u; \vec{v_2})$.

R2. $(\psi_1 \vee \psi_2)$ for all $\psi_1, \psi_2 \in \mathsf{SpLog}(\mathsf{W})$ with $\mathsf{free}(\psi_1) = \mathsf{free}(\psi_2)$: Assuming we have update formulas $\phi_\partial^{\psi_1}(u; \vec{v})$ and $\phi_\partial^{\psi_2}(u; \vec{v})$ for SpLog formulas ψ_1 and ψ_2 respectively, the update formula for $\phi_\partial^{(\psi_1 \vee \psi_2)}(u; \vec{v})$ is $\phi_\partial^{\psi_1}(u; \vec{v}) \vee \phi_\partial^{\psi_2}(u; \vec{v})$.

R3. $\exists x \colon \psi$ for all $\psi \in \mathsf{SpLog}(\mathsf{W})$ and $x \in \mathsf{free}(\psi) \setminus \{\mathsf{W}\}$: If a variable $x \in \Xi$ is existentially quantified within the SpLog formula, then we existentially quantify the variables $x_i, t_i \in D$ where $w[x_i, t_i]$ represents $\sigma(x)$ for some substitution σ.

R4. $(\psi \wedge \mathsf{C}_A(x))$ for every $\psi \in \mathsf{SpLog}(\mathsf{W})$, every $x \in \mathsf{free}(\psi)$, and every NFA A: let $A := (Q, \delta, s, F)$ be an NFA. We have that Q is a finite set of states, $\delta \colon Q \times \Sigma \to Q$ is the transition function, s is the initial state and $F \subseteq Q$ is the set of accepting states. We denote the reflexive and transitive closure of δ as $\delta^* \colon Q \times \Sigma^* \to Q$. For regular constraints, we maintain the relation $R_A := \{(i,j) \in D^2 \mid w[i,j] \in \mathcal{L}(A)\}$

From Proposition 3.3 in Gelade, Marquardt, and Schwentick [10], we know that the following relations can be maintained in DynPROP, and from [20] (Theorem 3.1.5, part b) we know that DynPROP is a strict subclass of DynCQ. Hence we can maintain the following in DynCQ:

$$\begin{aligned} R_{p,q} :=& \{(i,j) \in D^2 \mid i < j \text{ and } \delta^*(p, w[i+1, j-1]) = q\}, \\ I_q :=& \{j \in D \mid \delta^*(s, w[1, j-1]) = q\}, \\ F_p :=& \{i \in D \mid \delta^*(p, w[i+1, n] \in F\}. \end{aligned}$$

Where $p, q \in Q$. We also know, from [10], that we can maintain the 0-ary relation ACC, which is true if and only if $w' \in \mathcal{L}(A)$.

We maintain R_A with $\phi_\partial^{R_A}(u; x, y) := \psi_1^{R_A} \vee \psi_2^{R_A} \vee \psi_3^{R_A} \vee \psi_4^{R_A}$ where each $\psi_i^{R_A}$ is a subformula which we now define for separate cases. Note that $R'(\vec{x})$ is shorthand for $\phi_\partial^R(u; \vec{x})$. We define $\psi_1^{R_A}$ as

$$\psi_1^{R_A} := \exists x_2, y_2 \colon \big(R'_{\mathsf{Next}}(x_2, x) \wedge R'_{\mathsf{Next}}(y, y_1) \wedge \bigvee_{f \in F} (R'_{s,f}(x_2, y_2)\big).$$

Since $R_{p,q}(x,y)$ refers to the substring from position $x+1$ to $y-1$, and we wish to examine the string from position x to y, we look at $R'_{s,f}(x_2, y_2)$ where $x_2 \rightsquigarrow_{w'} x$ and $y \rightsquigarrow_{w'} y_2$. If it is indeed the case that $x_2 \rightsquigarrow_{w'} x$ and $y \rightsquigarrow_{w'} y_2$ then $w'[x_2 + 1, y_2 - 1] = w[x, y]$. Therefore $R'_{s,f}(x_2, y_2)$, for $f \in F$, is true for such x_2 and y_2 if and only if $\delta^*(s, w[x,y]) \in F$ which is the desired behavior for this case. Note that $\psi_1^{R_A}$ fails if there doesn't exist x_2 such that

$x_2 \rightsquigarrow_{w'} x$ or there doesn't exists y_2 such that $y \rightsquigarrow_{w'} y_2$. This is dealt with using $\psi_2^{R_A}, \psi_3^{R_A}$ and $\psi_4^{R_A}$, which we explore next.

If $R'_{\mathsf{last}}(y)$ then $w'[x,y] = w'[x,n]$ where $n = |D|$. Therefore, we can use $F'_s(x_2)$ for some $x_2 \in D$ where $x_2 \rightsquigarrow_{w'} x$ and s is the initial state of the NFA, to see whether $\delta^*(s, w'[x,n]) \in F$ and hence whether $\delta^*(s, w'[x,y]) \in F$. To realize this behavior, we define $\psi_2^{R_A}$ as

$$\psi_2^{R_A} := \exists x_2 \colon \big(R'_{\mathsf{Next}}(x_2, x) \land R'_{\mathsf{last}}(y) \land F'_s(x_2)\big).$$

If $R'_{\mathsf{first}}(x)$ then $w'[1,y] = w'[x,y]$. Therefore, we can use $I'_f(y_2)$ for some $y_2 \in D$ where $y \rightsquigarrow_{w'} y_2$ and $f \in F$, to see whether $\delta^*(s, w'[1,y]) \in F$ and hence whether $\delta^*(s, w'[x,y]) \in F$. To realize this behavior, we define $\psi_3^{R_A}$ as

$$\psi_3^{R_A} := \exists y_2 \colon \big(R'_{\mathsf{Next}}(y, y_2) \land R'_{\mathsf{first}}(x) \land \bigvee_{f \in F} (I'_f(y_2))\big).$$

If $R'_{\mathsf{first}}(x)$ and $R'_{\mathsf{last}}(y)$ then $w'[x,y] = w'$ and therefore it follows that $w'[x,y] \in \mathcal{L}(A)$ if and only if $w' \in \mathcal{L}(A)$. We only need to see if ACC' is true for this case. We realize this behavior by defining $\psi_4^{R_A}$ as

$$\psi_4^{R_A} := R'_{\mathsf{first}}(x) \land R'_{\mathsf{last}}(y) \land \mathsf{ACC}'.$$

To simulate $(\psi \land \mathsf{C}_A(x))$ for every $\psi \in \mathsf{SpLog(W)}$, every $x \in \mathsf{free}(\psi)$, and every NFA A within DynCQ, we do the following; let $\phi_\partial^{\psi}(u; \vec{v})$ be an update formula for $\psi \in \mathsf{SpLog}$ and since for some $\sigma(x)$, where $x \in \mathsf{free}(\psi)$, has $x_i, t_i \in D$ associated with it, we can use $\phi_\partial^{\psi}(u; \vec{v}) \land \phi_\partial^{R_A}(u; x_i, t_i)$ which is true if and only if $w'[x_i, t_i] \in \mathcal{L}(A)$. ◀

Most of the work for this proof follows from Lemma 3.9 and Corollary 3.11. Extra work is done in order to simulate regular constraints, although this follows on from the fact that DynPROP maintains the regular languages [10].

▶ **Theorem 3.13.** *Core spanners can be maintained in DynCQ.*

Proof. Although maintaining the SpLog relation that realizes a spanner is not the same as maintaining the spanner relation as defined in Section 2.3, the changes we need to make are trivial. Let P be a spanner and let ψ_P be a SpLog formula that realizes P. We know that $\mathsf{free}(\psi_P) = \{x_p, x_c \mid x \in \mathsf{SVars}\,(P)\}$, and for every $x \in \mathsf{SVars}\,(P)$ where $[i,j\rangle := \mu(x)$, we have both $\sigma(x_p) = w_{[1,i\rangle}$ and $\sigma(x_c) = w_{[i,j\rangle}$. Let R^P be a relation that maintains the spanner P. The only difference between update formulas that maintain P and update formulas that maintain the relation SpLog selects which realizes P is that the two elements $x_p^o, x_p^c \in D$ that are used to represent the SpLog variable $x_p \in \Xi$ are existentially quantified whereas the two variables $x_c^o, x_c^c \in D$ which represent $x_c \in \Xi$ are not. ◀

Theorem 3.13 shows us that DynCQ is at least as expressive as SpLog. We will use this along with Proposition 4.1 to show that DynCQ is more expressive than core spanners. Given that we can maintain any relation selectable in SpLog using DynCQ, it is no big surprise that adding negation allows us to maintain $\mathsf{SpLog}^\neg$ in DynFO.

▶ **Lemma 3.14.** *Any relation selectable in SpLog¬ can be maintained in DynFO.*

Proof. Let $\psi \in \mathsf{SpLog(W)}$ and let R^ψ be the relation maintaining ψ where the update formulas for R^ψ are in CQ. The extra recursive rule allowing for $(\neg\psi) \in \mathsf{SpLog}^\neg(\mathsf{W})$ can be maintained by $\phi_\partial^{R^{\neg\psi}}(u; \vec{x}) = \neg\phi_\partial^{R^\psi}(u; \vec{x})$. ◀

As with Theorem 3.13, we can use the result from Lemma 3.14 along with Corollary 4.2 to show that DynFO is more expressive than SpLog$^\neg$.

▶ **Theorem 3.15.** *Generalized core spanners can be maintained in DynFO.*

Since SpLog$^\neg$ captures the generalized core spanners, it follows from Lemma 3.14 that any generalized core spanner can be maintained in DynFO. In Section 4 we show that DynFO is more expressive than SpLog$^\neg$, it therefore follows that DynFO is more expressive than generalized core spanners.

4 Relations in SpLog and DynCQ

In this section, we examine the comparative expressive power of SpLog and DynCQ. Recall that we defined the notion of SpLog-selectable relations at the end of Section 2.2. We now define an analogous concept for DynCQ. For a relation $R \subseteq (\Sigma^*)^k$, we define the corresponding relation in the dynamic setting $\bar{R}$ as the $2k$-ary relation of all $(x_1, y_1, \dots, x_k, y_k) \in D^{2k}$ such that $(w[x_1, y_1], \dots, w[x_k, y_k]) \in R$. We say that R is selectable in DynCQ if $\bar{R}$ can be maintained in DynCQ.

For example, the equal length relation is defined as $R_{\mathsf{len}} := \{(w_1, w_2) \mid |w_1| = |w_2|\}$. From Fagin et al. [4] it is known that this relation is not selectable with core spanners. This relation in the dynamic setting is $\bar{R}_{\mathsf{len}} = \{(u_1, u_2, v_1, v_1) \in D^4 \mid |w[u_1, u_2]| = |w[v_1, v_2]|\}$.

▶ **Proposition 4.1.** *The equal length relation is selectable in DynCQ.*

Proof. To maintain the equal length relation, we take the update formulas from Lemma 3.7 and omit any atoms relating to the symbol of an element of the domain D. We also remove the constraint that the first subword must appear before the second. We also use $\bar{R}_{\mathsf{len}}$ in any update formula, rather than R_{eq}. The only exception to omitting all atoms relating to the symbol of an element, is to ensure that $w[u_1] \neq \varepsilon$, $w[u_2] \neq \varepsilon$, $w[v_1] \neq \varepsilon$, and $w[v_2] \neq \varepsilon$. ◀

While this allows us to separate the languages that are definable in SpLog from the ones that can be maintained in DynCQ, we consider the following more wide-ranging example:

▶ **Lemma 4.2.** *The language $\{w \in \Sigma^* \mid |w| = 2^n, n \geq 0\}$ is maintainable in DynCQ.*

Proof. Let P be a 2-ary relation such that $P(x, y)$ holds if and only if $|w[x, y]| = 2^n$ for some $n \in \mathbb{N}$. This can be maintained by having that $P(x, y)$ holds if $|w[x, y]| = 1$ or if there exists $z_1, z_2 \in D$ such that $P(x, z_1)$, $P(z_2, y)$, $R'_{\mathsf{Next}}(z_1, z_2)$ and that $\bar{R}_{\mathsf{len}}(x, z_1, z_2, y)$. If we assume that $|w[x, z_1]| = 2^n$ for some $n \in \mathbb{N}$, which we do because we have the base case of $w[x, y] = a$, and that $|w[x, z_1]| = |w[z_2, y]|$, then it follows that if $R'_{\mathsf{Next}}(z_1, z_2)$ then $w[x, y] = w[x, z_1] \cdot w[z_2, y]$ and therefore $|w[x, y]| = 2|w[x, z_1]|$ and hence $|w[x, y]| = 2^{n+1}$. We then have that $|w| = 2^n$ if $\exists x, y \colon \big(R'_{\mathsf{first}}(x) \wedge R'_{\mathsf{last}}(y) \wedge P'(x, y)\big)$. ◀

For every choice of Σ, this language is not expressible in SpLog$^\neg$ (and, hence, not in SpLog). This is easily seen by considering the case that Σ is unary[4]. As shown in [7] for core spanners and then in [18] for generalized core spanners, both classes collapse to exactly the class of regular languages if $|\Sigma| = 1$. As the language of all words a^{2^n} is not regular, this shows that even DynCQ can define languages that are not expressible in SpLog$^\neg$.

[4] Larger alphabets then follow by observing that the class of SpLog$^\neg$-languages is trivially closed under intersection with regular languages.

Combining this with Theorem 3.13 and Theorem 3.15, we respectively conclude that DynCQ is strictly more expressive than core spanners and that DynFO is strictly more expressive than generalized core spanners.

As explained in Section 6 of [6], there are few inexpressibility results for SpLog that generalize to non-unary alphabets (and basically none for SpLog$^\neg$), apart from straightforward complexity observations that are not particularly illuminating. Nonetheless, Proposition 6.7 in [6] establishes that none of the following relations is SpLog-selectable:

▶ **Proposition 4.3.** *The following relations are DynCQ-selectable but not SpLog-selectable:*

$$\begin{aligned} R_{\mathsf{num}(a)} &:= \{(w_1, w_2) \mid |w_1|_a = |w_2|_a\} \text{ for } a \in \Sigma, \\ R_{\mathsf{perm}} &:= \{(w_1, w_2) \mid |w_1|_a = |w_2|_a \text{ for all } a \in \Sigma\}, \\ R_{\mathsf{rev}} &:= \{(w_1, w_2) \mid w_2 = w_1^R\}, \text{ where } w_1^R \text{ is the reversal of } w_1, \\ R_< &:= \{(w_1, w_2) \mid |w_1| < |w_2|\}, \\ R_{\mathsf{scatt}} &:= \{(w_1, w_2) \mid w_1 \text{ is a scattered subword of } w_2\}, \end{aligned}$$

where w_1 is a scattered subword of w_2 if, for some $n \geq 1$, there exist $s_1, \ldots, s_n, \bar{s}_0, \ldots, \bar{s}_n \in \Sigma^*$ such that $w_1 = s_1 \cdots s_n$ and $w_2 = \bar{s}_0 s_1 \bar{s}_1 \cdots s_n \bar{s}_n$.

Proof. The relations R_{scatt}, $R_{\mathsf{num}(a)}$, and R_{rev} have case distinctions equivalent to the proof of Lemma 3.7, therefore we give the overarching idea of the proof but without exploring every case. See [9] for a full proof of Lemma 3.7.

Maintaining R_{scatt}: For insertion, we give three steps for this proof; inheritance, base case, and an inductive step.

We have that if $w[u_1, u_2]$ is a scattered subword of $w[v_1, v_2]$ and u is outside of the interval $[u_1, u_2]$, then $w'[u_1, u_2]$ remains a scattered subword of $w'[v_1, v_2]$ and therefore $R'_{\mathsf{scatt}}(u_1, u_2, v_1, v_2)$ should hold. We call this step inheritance.

The base case is that given the update $\mathsf{ins}_\zeta(u)$ for some $u \in D$, if there exists $v \in D$ such that $v_1 \leq v \leq v_2$ and $w(v) = w(u) = \zeta$, then it follows that $w(u)$ is a scattered subword of $w[v_1, v_2]$ and therefore $R'_{\mathsf{scatt}}(u, u, v_1, v_2)$ should hold.

For the inductive step, given that we have some update $\mathsf{ins}_\zeta(u)$, if $w[u_1, x_1]$ is a scattered subword of $w[v_1, x_2]$ and $w[x_3, u_2]$ is a scattered subword of $w[x_4, v_2]$, it follows that $w[u_1, u_2]$ is a scattered subword of $w[v_1, v_2]$ if $x_1 \rightsquigarrow_{w'} u \rightsquigarrow_{w'} x_3$ and $w(u)$ is a scattered subword of $w[x_2, x_4]$. Deletion is dealt with analogously, although without the base case.

Maintaining $R_{\mathsf{num}(a)}$: We again give three steps; inheritance, the base case(s), and an inductive step.

We have that if $|w[u_1, u_2]|_a = |w[v_1, v_2]|_a$ and u is outside of the interval $[u_1, u_2]$, then $|w'[u_1, u_2]|_a = |w'[v_1, v_2]|_a$ and therefore $R'_{\mathsf{num}(a)}(u_1, u_2, v_1, v_2)$ should hold. We call this step inheritance. We have that (u_1, u_2, v_1, v_2) is not inherited if $u \in [u_1, u_2]$ or $u \in [v_1, v_2]$, but this should be dealt with by the inductive step.

To maintain $R_{\mathsf{num}(a)}$, we have two base cases. Given the update $\mathsf{ins}_a(u)$, we have that $|w'(u)|_a = |w'(v)|_a$ if $w'(v) = a$.

For the inductive step, we have that if $|w[u_1, x_1]|_a = |w[v_1, x_2]|_a$ and $|w(u)|_a = |w(v)|_a$ and $|w[x_3, u_2]|_a = |w[x_4, v_2]|_a$ where $x_1 \rightsquigarrow_{w'} u \rightsquigarrow_{w'} x_3$ and $x_2 \rightsquigarrow_{w'} v \rightsquigarrow_{w'} x_4$, then $|w'[u_1, u_2]|_a = |w'[v_1, v_2]|_a$. Dealing with deletion is analogous to insertion but without the base case.

Maintaining R_{rev}: We can maintain this with a simple variation of the update formula which maintains R_{eq}. Firstly, we remove the constraint that the first subword must appear before the second. Then, whenever $R_{\mathsf{eq}}(\cdot)$ is used as a subformula, one would need to use $R_{\mathsf{rev}}(\cdot)$ instead. The more involved aspect of altering the update formulas would be to reverse the ordering of certain indices. Informally, check $y \rightsquigarrow_w x$ instead of $x \rightsquigarrow_w y$ where necessary.

Maintaining R_{perm}: $\phi_{\partial}^{R_{\mathsf{perm}}}(u; u_1, u_2, v_1, v_2) := \bigwedge_{\zeta \in \Sigma} \big(\phi_{\partial}^{R_{\mathsf{num}(\zeta)}}(u; u_1, u_2, v_1, v_2)\big).$

Maintaining $R_<$:

$$\phi_{\partial}^{R_<}(u; u_1, u_2, v_1, v_2) := \exists x_1 \exists x_2 \colon \big(R_{\mathsf{len}}(u_1, u_2, x_1, x_2) \\ \wedge (x_1 < v_1) \wedge (v_1 \leq v_2) \wedge (v_2 < x_2)\big).$$

◀

By Lemma 5.1 in [6], a k-ary relation R is SpLog-selectable if and only there is some SpLog-formula $\varphi(\mathsf{W}; x_1, \ldots, x_k)$ such that for all σ that satisfy $\sigma(x_i) \sqsubseteq \sigma(\mathsf{W})$ for all $i \in [k]$, we have $\sigma \models \varphi$ if and only if $(\sigma(x_1), \ldots, \sigma(x_k)) \in R$. One can show with little effort that relations like string inequality, the substring relation, or equality modulo a bounded Levenshtein-distance are all SpLog-selectable (see Section 5.1 of [6]). By Lemma 3.12, we can directly use these relations in constructions for DynCQ-definable languages and DynCQ-selectable relations.

▶ **Example 4.4.** For $k \geq 1$ and $w_1, w_2 \in \Sigma^*$, we say that w_1 is a k-scattered subword of w_2 if there exist $s_1, \ldots, s_k, \bar{s}_0, \ldots, \bar{s}_k \in \Sigma^*$ such that $w_1 = s_1 \cdots s_k$ and $w_2 = \bar{s}_0 s_1 \bar{s}_1 \cdots s_k \bar{s}_k$. This relation is SpLog-selectable[5], as demonstrated by the following SpLog-formula which uses syntactic sugar from Section 5.1 of [6]:

$$\varphi(\mathsf{W}; w_1, w_2) := \exists s_1, \ldots, s_k, \bar{s}_0, \ldots, \bar{s}_k : \Big((w_1 \doteq s_1 \cdots s_k) \wedge (w_2 \doteq \bar{s}_0 s_1 \bar{s}_1 \cdots s_k \bar{s}_k)\Big).$$

Although one could show directly that the k-scattered subword relation is DynCQ-selectable, using SpLog and Lemma 3.12 can avoid hand-waving.

We can even generalize this approach beyond SpLog. In the proof of Lemma 3.12, we use the fact the every regular language is in DynCQ to maintain regular constraints for SpLog. Analogously, we can extend SpLog with relation symbols for any DynCQ-sectable relation and use the resulting logic for DynCQ. Of course, all this applies to SpLog$^{\neg}$ and DynFO.

5 Conclusions

From a document spanner point of view, the present paper establishes upper bounds for maintaining the three most commonly examined classes of document spanners, namely DynPROP for regular spanners, DynCQ for core spanners, and DynFO for generalized core spanners. While the bounds for regular spanners and generalized core spanners are what one might expect from related work, the DynCQ-bound for core spanners might be considered surprising low (keeping in mind, of course, that it is still open whether DynCQ is less expressive than DynFO).

By analyzing the proof of Lemma 3.12, the central construction of this main result, it seems that the most important part of maintaining core spanners is updating the string equality relation and the regular constraints. One big question for future work is whether this might have any practical use for the evaluation of core spanners. Although some may consider this unlikely, there is at least some possibility that some techniques might be useful.

[5] Unlike a relation for unbounded scattered subword.

In the present paper, we only examine updates that affect single letters. At least as far as the main result is concerned, it should be possible to generalize this to cut and paste operations, as they are commonly found in text editors. These other operations beyond single letters are promising directions for further work.

From a dynamic complexity point of view, Section 4 describes how SpLog can be used as a convenient tool that allows shorter proofs that languages can be maintained in DynCQ. One consequence of this is that a large class of regular expressions with backreference operators (see Section 5.3 of [6]) are in fact DynCQ-languages.

References

1 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. In *Proceedings of ICDT 2019*, pages 22:1–22:19, 2019.

2 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-Correctness in Information Extraction. In *Proceedings of PODS 2019*, pages 149–163, 2019.

3 Guozhu Dong, Jianwen Su, and Rodney Topor. Nonrecursive incremental evaluation of datalog queries. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):187–223, 1995.

4 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. *Journal of the ACM*, 62(2):12, 2015.

5 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant Delay Algorithms for Regular Document Spanners. In *Proceedings of PODS 2018*, pages 165–177, 2018.

6 Dominik D. Freydenberger. A Logic for Document Spanners. *Theory of Computing Systems*, 63(7):1679–1754, 2019.

7 Dominik D. Freydenberger and Mario Holldack. Document Spanners: From Expressive Power to Decision Problems. *Theory of Computing Systems*, 62(4):854–898, 2018.

8 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining Extractions of Regular Expressions. In *Proceedings of PODS 2018*, pages 137–149, 2018.

9 Dominik D. Freydenberger and Sam M. Thompson. Dynamic Complexity of Document Spanners, 2019. arXiv:1909.10869.

10 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Transactions on Computational Logic*, 13(3):19:1–19:36, 2012.

11 Tao Jiang, Efim Kinber, Arto Salomaa, Kai Salomaa, and Sheng Yu. Pattern languages with and without erasing. *International Journal of Computer Mathematics*, 50(3-4):147–163, 1994.

12 Katja Losemann. *Foundations of Regular Languages for Processing RDF and XML*. PhD thesis, University of Bayreuth, 2015. URL: https://epub.uni-bayreuth.de/2536/.

13 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document Spanners for Extracting Incomplete Information: Expressiveness and Complexity. In *Proceedings of PODS 2018*, pages 125–136, 2018.

14 Andrea Morciano, Martin Ugarte, and Stijn Vansummeren. Automata-Based Evaluation of AQL queries. Technical report, Université Libre de Bruxelles, 2016.

15 Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. Dynamic Graph Queries. In *Proceedings of ICDT 2016*, pages 14:1–14:18, 2016.

16 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *Journal of Computer and System Sciences*, 55(2):199–209, 1997.

17 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity Bounds for Relational Algebra over Document Spanners. In *Proceedings of PODS 2019*, pages 320–334, 2019.

18 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive Programs for Document Spanners. In *Proceedings of ICDT 2019*, pages 13:1–13:18, 2019.

19 Markus L. Schmid. Characterising REGEX Languages by Regular Languages Equipped with Factor-Referencing. *Information and Computation*, 249:1–17, 2016.

20 Thomas Zeume. *Small dynamic complexity classes*. Springer, 2017.

21 Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. *Journal of Computer and System Sciences*, 88:3–26, 2017.

When Can Matrix Query Languages Discern Matrices?

Floris Geerts
University of Antwerp, Belgium
http://adrem.uantwerpen.be/floris.geerts
floris.geerts@uantwerpen.be

Abstract

We investigate when two graphs, represented by their adjacency matrices, can be distinguished by means of sentences formed in MATLANG, a matrix query language which supports a number of elementary linear algebra operators. When undirected graphs are concerned, and hence the adjacency matrices are real and symmetric, precise characterisations are in place when two graphs (i.e., their adjacency matrices) can be distinguished. Turning to directed graphs, one has to deal with asymmetric adjacency matrices. This complicates matters. Indeed, it requires to understand the more general problem of when two arbitrary matrices can be distinguished in MATLANG. We provide characterisations of the distinguishing power of MATLANG on real and complex matrices, and on adjacency matrices of directed graphs in particular. The proof techniques are a combination of insights from the symmetric matrix case and results from linear algebra and linear control theory.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Mathematics of computing → Graph theory

Keywords and phrases matrix query languages, linear algebra, expressive power

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.12

Acknowledgements I want to thank Lieven Le Bruyn for inspiring discussions and for pointing me to the connections to linear control theory.

1 Introduction

The integration of linear algebra functionalities inside relational database systems is currently high on the agenda [4, 5, 6, 12, 19, 31, 33, 34, 35, 36, 39, 40]. The need for such an integration is due to the increased importance of linear algebra for scalable machine learning and data analytics. From a query language perspective, it is challenging to combine classical relational data operators with linear algebra operators. The Lara language is one such proposal [29] and its connections to classical database query languages has been recently explored [3]. Logics extended with linear algebra operators have been considered as well in an attempt to find logics capturing PTIME and to study the descriptive complexity of linear algebra [15, 16, 17, 18, 22, 26, 27]. An even more basic question is to design a query language for matrices and linear algebra alone. In recent work, a query language for matrices, MATLANG, was introduced in which some basic linear operators are supported [8, 7]. The design of MATLANG is motivated by operations commonly supported by linear algebra packages. It can be seen as a linear algebraic counterpart of the relational algebra on K-relations [9], where K is a semiring representing the domain of matrix entries. We here continue the study of the expressive power of MATLANG. What is known so far is that when MATLANG is regarded as a query language on graphs, i.e., queries in MATLANG take the adjacency matrix of a (directed/undirected) graph as input and return an adjacency matrix, then its expressive power is bounded by aggregate logic with only three non-numerical variables. Furthermore, when asked whether two undirected graphs G and H are *indistinguishable by means of sentences* in MATLANG, denoted by $G \equiv_{\mathsf{MATLANG}} H$, then this precisely corresponds to these graphs being C^3-equivalent [20]. Here, C^3 is the three-variable fragment of first-order logic with counting. With a sentence in MATLANG one means a query which returns scalars on input matrices, in analogy with sentences in logic. A more fine-grained analysis of the impact of each of the operators in MATLANG as a graph query language was provided in [20]. In particular, if $\mathcal{L}$ is a subset of the operators in MATLANG, then $\mathsf{ML}(\mathcal{L})$ refers to the fragment of MATLANG supporting only those operators in $\mathcal{L}$. Precise characterisations were obtained for when two *undirected* graphs G and H are indistinguishable by sentences in $\mathsf{ML}(\mathcal{L})$, denoted by $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ [20]. In particular, a fragment $\mathsf{ML}(\mathcal{L})$ was identified such that $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ if and only if $G \equiv_{\mathsf{C}^2} H$, where C^2 is the two-variable fragment of first-order logic with counting. We remark that $G \equiv_{\mathsf{C}^2} H$ is known to correspond to G and H being *fractional isomorphic* [10, 30, 37, 38], i.e., there exists a doubly stochastic matrix (non-negative and rows and columns sum up to one) S such that $A_G \cdot S = S \cdot A_H$, where A_G and A_H denote adjacency matrices of G and H, respectively. In [20], for each fragment $\mathsf{ML}(\mathcal{L})$ a proper class of matrices was identified such that for undirected graphs G and H, $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ if and only if $A_G \cdot X = X \cdot A_H$ for some matrix X in that class. For example, for the fragment corresponding to C^2-equivalence this class consists of all doubly stochastic matrices. Such characterisations enable to assess the expressive power of the fragment $\mathsf{ML}(\mathcal{L})$. Indeed, graph properties expressible by sentences in $\mathsf{ML}(\mathcal{L})$ should be invariant under such transformations X. All of the above relates to undirected graphs only.

It seems natural to ask what changes when *directed* graphs (or *digraphs*, for short) are considered, and this is the focus of this paper. More precisely, we investigate how the operators in MATLANG interact with adjacency matrices of directed graphs. Compared to the undirected graph case, the adjacency matrices are *not necessarily symmetric* anymore (i.e. the entry A_{ij} in a matrix A may be different from entry A_{ji}). As a consequence, one cannot rely on properties of symmetric matrices, the most important being that symmetric matrices are diagonalisable. Some of the results in [20] relied on this. Furthermore, whereas algebraic graph theory provides a comprehensive insight in the properties of undirected graphs, which underly some of the results in [20], this is less so for directed graphs.

To obtain characterisations for digraphs, we are faced with the more general question of when two *general matrices* (complex or real) can be discerned by sentences in our fragments $\mathsf{ML}(\mathcal{L})$. We identify two techniques that allow us to answer this question:

- A technique from linear algebra for testing when two matrices A and B are related by means of an invertible matrix X, i.e., such that $A \cdot X = X \cdot B$ holds. This technique only works for fragments containing the *trace* operator ($\mathsf{tr}(\cdot)$), which takes the sum of the diagonal entries of a matrix, and *complex conjugate transposition* (*), which switches rows and columns, followed by complex conjugation. We describe this technique in detail and apply it for fragments $\mathsf{ML}(\mathcal{L})$ supporting $\mathsf{tr}(\cdot)$ and * in Section 5.
- A technique originating from the study of linear systems in control theory which allows to *reduce* matrices A and B to their so-called *minimal realisations* $\hat{A}$ and $\hat{B}$, for which the existence of an invertible matrix X such that $\hat{A} \cdot X = X \cdot \hat{B}$ holds is guaranteed. We show that this allows to link the original matrices A and B as well, and connect it to indistinguishability by our fragments. Neither trace nor complex conjugate transposition is needed here. We detail this technique and consider fragments without trace or complex conjugate transposition in Section 6.

The main observation is that many of the results from [20] graciously generalise to general matrices and to adjacency matrices of digraphs in particular. This is especially true for fragments including the trace operator. The differences here are subtle and mostly relate to the absence of complex conjugate transposition. This is not surprising. After all, this is the only operator in MATLANG that has access to the possible asymmetry of matrices. For trace-less fragments, the generalisations are less straightforward, due to the minimal realisation approach. We show, however, that the general results reported in this paper collapse to precisely the same results as for the undirected graph case, when *normal* matrices are used (normal matrices A satisfy $A^* \cdot A = A \cdot A^*$ and they are known to inherit most of the properties of symmetric matrices). As a pleasant side-effect, we thus recover results reported in [20] as special cases of the more general approach used in this paper.

2 Background

We denote by $\mathbb{R}$ ($\mathbb{C}$) the set of real (complex) numbers. The set of $m \times n$-matrices over the real (complex) numbers is denoted by $\mathsf{Mat}_{m\times n}(\mathbb{R})$ ($\mathsf{Mat}_{m\times n}(\mathbb{C})$). Vectors are elements of $\mathsf{Mat}_{m\times 1}(\mathbb{R})$ (or $\mathsf{Mat}_{m\times 1}(\mathbb{C})$), or $\mathsf{Mat}_{1\times m}(\mathbb{R})$ (or $\mathsf{Mat}_{1\times m}(\mathbb{C})$). The entries of an $m \times n$-matrix A are denoted by A_{ij}, for $i \in [1,m]$ and $j \in [1,n]$. The entries of a vector v are denoted by v_i, for $i \in [1,m]$. We often identify $\mathsf{Mat}_{1\times 1}(R)$ with $\mathbb{R}$, and $\mathsf{Mat}_{1\times 1}(\mathbb{C})$ with $\mathbb{C}$ and refer to these as scalars. The following classes of matrices are of interest in this paper: *square* matrices (elements in $\mathsf{Mat}_{n\times n}(\mathbb{R})$ or $\mathsf{Mat}_{n\times n}(\mathbb{C})$), *symmetric* matrices (such that $A_{ij} = A_{ji}$ for all i and j), *doubly stochastic* matrices ($A_{ij} \in \mathbb{R}$, $A_{ij} \geq 0$, $\sum_{j=1}^{n} A_{ij} = 1$ and $\sum_{i=1}^{m} A_{ij} = 1$ for all i and j), *doubly quasi-stochastic* matrices ($\sum_{j=1}^{n} A_{ij} = 1$ and $\sum_{i=1}^{m} A_{ij} = 1$ for all i and j), *orthogonal* matrices ($A \in \mathsf{Mat}_{n\times n}(\mathbb{R})$, $A^{\mathsf{t}} \cdot A = I_{n\times n} = A \cdot A^{\mathsf{t}}$, where A^{t} denotes the *transpose* of A obtained by switching rows and columns of A and $I_{n\times n}$ is the *identity matrix* in $\mathsf{Mat}_{n\times n}(\mathbb{R})$), *unitary* matrices ($A \in \mathsf{Mat}_{n\times n}(\mathbb{C})$, $A^* \cdot A = I_{n\times n} = A \cdot A^*$), and *normal* matrices ($A^* \cdot A = A \cdot A^*$). The matrix $J_{m\times n} \in \mathsf{Mat}_{m\times n}(\mathbb{R})$ denotes the matrix consisting of all ones and $O_{m\times n} \in \mathsf{Mat}_{m\times n}(\mathbb{R})$ denotes the zero matrix. We use I, J and O for $I_{n\times n}$, $J_{m\times n}$ and $O_{m\times n}$, respectively, when the dimensions are clear from the context. We assume familiarity with standard concepts of linear algebra and refer to [2, 28] for more background. A *directed graph* or *digraph* $G = (V,E)$ is defined as usual. The *order* of a digraph is its number of vertices. An *adjacency matrix* of a digraph G of order n, denoted by A_G, is an $n \times n$-matrix whose entries $(A_G)_{ij}$ are set to 1 if and only if $(i,j) \in E$, all other entries are set to 0. We regard *undirected* graphs as digraphs such that $(v,w) \in E$ implies that also $(w,v) \in E$, i.e., their adjacency matrices are symmetric. Strictly speaking, to define an adjacency matrix one requires an ordering on the vertices in G. In this paper, any ordering will do and we thus speak about “the” adjacency matrix of a (di)graph.

3 Matrix Query Languages

As described in Brijder et al. [8], matrix query languages can be formalised as compositions of linear algebra operators. By closing such operators under composition “matrix query languages” are formed. More specifically, for a set $\mathcal{L}$ of linear algebra operators $\mathsf{op}_1, \ldots, \mathsf{op}_k$ the corresponding matrix query language is denoted by $\mathsf{ML}(\mathcal{L})$ and consists of expressions formed by the grammar:

$$e := X \mid \mathsf{op}_1(e_1, \ldots, e_{p_1}) \mid \cdots \mid \mathsf{op}_k(e_1, \ldots, e_{p_k}),$$

where X denotes a *matrix variable* which serves to indicate the input to expressions and p_i denotes the number of inputs required by operator op_i. We allow a single matrix variable X in this paper, although some of the results can be generalised to multiple matrix variables.

The semantics of an expression $e(X)$ in $\mathsf{ML}(\mathcal{L})$ is defined inductively, relative to an *assignment* ν of X to a matrix $\nu(X) \in \mathsf{Mat}_{n\times n}(\mathbb{C})$, for some dimension n. In general, rectangular

Table 1 Linear algebra operators (supported in $\mathsf{MATLANG}$ [8, 20]) and their semantics. In the last column, $+$, $\times$ and $^{-}$ denote addition, multiplication and complex conjugation of complex numbers, respectively.

matrix multiplication ($\mathsf{op}(e_1,e_2)=e_1\cdot e_2$)		
$e_1(\nu(X)) = A \in \mathsf{Mat}_{m\times n}(\mathbb{C})$ $e_2(\nu(X)) = B \in \mathsf{Mat}_{n\times o}(\mathbb{C})$	$e_1(\nu(X))\cdot e_2(\nu(X)) = C \in \mathsf{Mat}_{m\times o}(\mathbb{C})$	$C_{ij} = \sum_{k=1}^{n} A_{ik}\times B_{kj}$
matrix addition ($\mathsf{op}(e_1,e_2)=e_1+e_2$)		
$e_i(\nu(X)) = A^{(i)} \in \mathsf{Mat}_{m\times n}(\mathbb{C})$	$e_1(\nu(X))+e_2(\nu(X)) = B \in \mathsf{Mat}_{m\times n}(\mathbb{C})$	$B_{ij} = A^{(1)}_{ij} + A^{(2)}_{ij}$
scalar multiplication ($\mathsf{op}(e)=ce$, $c\in\mathbb{C}$)		
$e(\nu(X)) = A \in \mathsf{Mat}_{m\times n}(\mathbb{C})$	$ce(\nu(X)) = B \in \mathsf{Mat}_{m\times n}(\mathbb{C})$	$B_{ij} = c\times A_{ij}$
Schur-Hadamard product ($\mathsf{op}(e_1,e_2)=e_1\odot e_2$)		
$e_1(\nu(X)) = A \in \mathsf{Mat}_{m\times m}(\mathbb{C})$ $e_2(\nu(X)) = B \in \mathsf{Mat}_{m\times m}(\mathbb{C})$	$e_1(\nu(X))\odot e_2(\nu(X)) = C \in \mathsf{Mat}_{m\times m}(\mathbb{C})$	$C_{ij} = A_{ij}\times B_{ij}$
complex conjugate transposition ($\mathsf{op}(e)=e^*$)		
$e(\nu(X)) = A \in \mathsf{Mat}_{m\times n}(\mathbb{C})$	$e(\nu(X))^* = A^* \in \mathsf{Mat}_{n\times m}(\mathbb{C})$	$(A^*)_{ij} = \bar{A}_{ji}$
identity ($\mathsf{op}(e)=\mathsf{Id}(e)$)		
$e(\nu(X)) = A \in \mathsf{Mat}_{m\times m}(\mathbb{C})$	$\mathsf{Id}(e(\nu(X)) = I_{m\times m} \in \mathsf{Mat}_{m\times m}(\mathbb{C})$	$I_{ii}=1$, $I_{ij}=0$, $i\neq j$
one-vector ($\mathsf{op}(e)=\mathbb{1}(e)$)		
$e(\nu(X)) = A \in \mathsf{Mat}_{m\times n}(\mathbb{C})$	$\mathbb{1}(e(\nu(X)) = \mathbb{1} \in \mathsf{Mat}_{m\times 1}(\mathbb{C})$	$\mathbb{1}_i = 1$
transpose one-vector ($\mathsf{op}(e)=\mathbb{1}^{\mathsf{t}}(e)$)		
$e(\nu(X)) = A \in \mathsf{Mat}_{m\times n}(\mathbb{C})$	$\mathbb{1}(e(\nu(X)) = \mathbb{1}^{\mathsf{t}} \in \mathsf{Mat}_{1\times m}(\mathbb{C})$	$\mathbb{1}_i = 1$
trace ($\mathsf{op}(e)=\mathsf{tr}(e)$)		
$e(\nu(X)) = A \in \mathsf{Mat}_{m\times m}(\mathbb{C})$	$\mathsf{tr}(e(\nu(X)) = c \in \mathbb{C}$	$c = \sum_{i=1}^{m} A_{ii}$

matrices are allowed but we only focus on square matrices in this paper. We denote by $e\big(\nu(X)\big)$ the result of evaluating $e(X)$ on $\nu(X)$. As expected, $\mathsf{op}_i(e_1(X),\ldots,e_{p_i}(X))(\nu(X)) := \mathsf{op}_i\big(e_1(\nu(X)),\ldots,e_{p_i}(\nu(X))\big)$ for linear algebra operator op_i. In Table 1 we list operators supported in the matrix query language $\mathsf{MATLANG}$ [8]. In the table we also show the semantics of the operators and indicate restrictions on the dimensions such that the operators are well-defined.

▶ Remark 1. We use a slightly modified list of operators than used in [8, 20]. For example, we leave out general function applications and only focus on the Schur-Hadamard product ($\odot$). The reason is that once two matrices are indistinguishably with regards to fragments including $\odot$, then adding more general function applications does not increase the distinguishing power [20]. We also leave out the diagonalisation operator (diag) which turns a vector into a diagonal matrix with the input vector on the diagonal. Previous results show that the real distinguishing power for fragments including diag comes from its ability to simulate the Schur-Hadamard product on vectors [20]. We therefore omit diag and use the Schur-Hadamard product on *vectors*, denoted by $\odot_v$, instead. We assume that all fragments include matrix multiplication, addition and scalar multiplication and we do not list these explicitly in the set $\mathcal{L}$ of supported operators. ◀

4 Problem statement

As mentioned in the introduction we want to understand when two matrices can be distinguished by a sentence in fragments $\mathsf{ML}(\mathcal{L})$. We define an expression $e(X)$ in $\mathsf{ML}(\mathcal{L})$ to be a *sentence* if $e(\nu(X))$ returns a scalar in $\mathbb{C}$ for any assignment ν of X. We note that the type system of $\mathsf{MATLANG}$ [8] allows to check whether an expression in $\mathsf{ML}(\mathcal{L})$ is a sentence.

▶ **Definition 2.** *Two matrices A and B in $\mathsf{Mat}_{n\times n}(\mathbb{C})$ are said to be* $\mathsf{ML}(\mathcal{L})$-equivalent, *denoted by $A \equiv_{\mathsf{ML}(\mathcal{L})} B$, if and only if $e(A) = e(B)$ for all sentences $e(X)$ in $\mathsf{ML}(\mathcal{L})$.* ◀

For (di)graphs G and H, we write $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ if and only if $A_G \equiv_{\mathsf{ML}(\mathcal{L})} A_H$, where A_G and A_H denote the adjacency matrices of G and H, respectively.

We aim to characterise when $A \equiv_{\mathsf{ML}(\mathcal{L})} B$ holds by determining how A and B relate to each other. A typical characterisation will be *similarity-based*, stating that $A \equiv_{\mathsf{ML}(\mathcal{L})} B$ if and only if $A \cdot X = X \cdot B$ for some matrix X with some specific properties, depending on the fragment $\mathsf{ML}(\mathcal{L})$ under consideration. We will see that for some fragments more complicated relationships than $A \cdot X = X \cdot B$ are needed. When digraphs are concerned, we also provide characterisations in terms of graph properties.

5 Fragments with the trace operation

We start with the equivalence of matrices for the fragments $\mathsf{ML}(\mathsf{tr},{}^*)$, $\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})$ and $\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1},\odot_v)$, and sub-fragments without * but with $\mathbb{1}^{\mathsf{t}}(\cdot)$ instead. We leave out the biggest fragment $\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1},\odot,\mathsf{Id})$ since an inspection of the proof for that fragment (sketched in [20]) shows that it works for arbitrary matrices. In particular, we have that for (di)graphs G and H, $G \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1},\odot,\mathsf{Id})} H$ if and only if $G \equiv_{\mathsf{C}^3} H$. For all fragments considered, pairs of undirected graphs are known that separate them [20]. We provide *new separating pairs* when there is a distinction between graphs and digraphs. We will see that this occurs when * is not supported. We next outline a general proof strategy for characterising equivalence for fragments with the trace operation and complex conjugate transposition (Section 5.1). This strategy will then be applied to the various fragments under consideration (Section 5.2).

5.1 Proof strategy

We use Specht's Theorem (see e.g., [32] or Theorem 2.2.6 in [28]) as the basis for our proof strategy. It can be stated as follows. Let $\mathcal{A} = A_1,\ldots,A_k$ and $\mathcal{B} = B_1,\ldots,B_k$ be two sequences of k matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$ which are closed under *, i.e., each A_i^* is in $\mathcal{A}$, and similarly, each B_i^* is in $\mathcal{B}$. Then, $\mathcal{A}$ and $\mathcal{B}$ are called *(simultaneously) unitary similar* if there exists a unitary matrix U such that for all $i \in [1,k]$, $A_i \cdot U = U \cdot B_i$. Specht's Theorem provides necessary and sufficient conditions for this to hold. More precisely, $\mathcal{A}$ and $\mathcal{B}$ are simultaneously unitary similar if and only if $\mathsf{tr}(w(A_1,\ldots,A_k)) = \mathsf{tr}(w(B_1,\ldots,B_k))$ for all words $w(x_1,\ldots,x_k)$ over variables $x_1,\ldots,x_k$. As an example of what $w(A_1,\ldots,A_k)$ means, if $\mathcal{A} = A, A^*$ and $w(x,y) = xxyx$, then $w(A,A^*) = A\cdot A\cdot A^*\cdot A$. So, variables in words are substituted by matrices and concatenation is interpreted as matrix multiplication. The real analogue of Specht's Theorem is as follows [32]: Let $\mathcal{A}$ and $\mathcal{B}$ be two sequences of k matrices in $\mathsf{Mat}_{n\times n}(\mathbb{R})$ which are closed under transposition. Then, $\mathcal{A}$ and $\mathcal{B}$ are called *(simultaneously) orthogonal similar* if there exists a (real) orthogonal matrix Q such that $A_i \cdot Q = Q \cdot B_i$ for all $i \in [1,k]$. Again, this is equivalent to requiring $\mathsf{tr}(w(A_1,\ldots,A_k)) = \mathsf{tr}(w(B_1,\ldots,B_k))$ for all words $w(x_1,\ldots,x_k)$ over variables $x_1,\ldots,x_k$[1].

We can use Specht's Theorem to characterise $\mathsf{ML}(\mathcal{L})$-equivalence for fragments with trace and *, as follows. Let A and B be two matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Let $\Sigma = e_1(X),\ldots,e_k(X)$ be a finite sequence of expressions such that (i) each $e_i(X)$ is in $\mathsf{ML}(\mathcal{L})$; (ii) each $e_i(X) \in \Sigma$

[1] For both the complex and real version of Specht's Theorem there are bounds on the length of words that one needs to consider, a rough bound being $2n^2$ [32]. Some recent progress and tighter bounds are reported in [41]. These quantitative bounds do not play a role in what follows.

evaluates to a matrix, i.e., $e_i(A) \in \mathsf{Mat}_{n\times n}(\mathbb{C})$ (and hence also $e_i(B) \in \mathsf{Mat}_{n\times n}(\mathbb{C})$); and (iii) when $e_i(X) \in \Sigma$ also $(e_i(X))^* \in \Sigma$, i.e., Σ is closed under *.

Given $\Sigma = e_1(X), \dots, e_k(X)$ and A and B, we then construct two sequences of matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$: $\Sigma(A) := e_1(A), \dots, e_k(A)$ and $\Sigma(B) := e_1(B), \dots, e_k(B)$. Clearly, these sequences are closed under complex conjugation by the definition of Σ. Let $w(x_1, \dots, x_k)$ be a word over $x_1, \dots, x_k$. For each such word we consider the $\mathsf{ML}(\mathcal{L})$-*sentence*:

$$e_w(X) := \mathsf{tr}\big(w(e_1(X), \dots, e_k(X))\big).$$

Then, $A \equiv_{\mathsf{ML}(\mathcal{L})} B$ implies that $e_w(A) = e_w(B)$ for any word $w(x_1, \dots, x_k)$ and hence, by Specht's Theorem, there exists a unitary matrix U such that $e_i(A) \cdot U = U \cdot e_i(B)$ for all $i \in [1, k]$. We will always assume that in Σ, $e_1(X) := X$, such that $A \equiv_{\mathsf{ML}(\mathcal{L})} B$ implies that $A \cdot U = U \cdot B$ and $e_i(A) \cdot U = U \cdot e_i(B)$ for $i \in [2, k]$.

The sequences Σ of $\mathsf{ML}(\mathcal{L})$-expressions which we will use ensure that the unitary/orthogonal matrix is restricted such that *similarity is preserved* by the operators in $\mathsf{ML}(\mathcal{L})$. This allows, by induction on the structure of expressions in $\mathsf{ML}(\mathcal{L})$, to show that when $\Sigma(A)$ and $\Sigma(B)$ are simultaneously unitary equivalent, then $A \equiv_{\mathsf{ML}(\mathcal{L})} B$. We will not detail these inductive proofs as they are similar to those underlying the results in [20].

5.2 Results

$\mathsf{ML}(\mathsf{tr}, {}^*)$-equivalence

For undirected graphs it is known that $G \equiv_{\mathsf{ML}(\mathsf{tr},^*)} H$ if and only if $A_G \cdot Q = Q \cdot A_H$ for an orthogonal matrix Q [20]. The proof relies on the Spectral Theorem of symmetric matrices (see e.g., Theorem 2.5.3 in [28]), which does not apply for general matrices and, in particular, for adjacency matrices of directed graphs. Instead, we here follow our proof strategy. Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. To apply Specht's Theorem, we consider the sequence $\Sigma = e_1(X) := X, e_2(X) := X^*$. Hence, $\Sigma(A) = A, A^*$ and $\Sigma(B) = B, B^*$ and $A \equiv_{\mathsf{ML}(\mathsf{tr},^*)} B$ implies that $A \cdot U = U \cdot B$ and $A^* \cdot U = U \cdot B^*$ for a unitary matrix U. This shows one direction of the following proposition. The other direction is shown by induction on the structure of expressions and uses that $\mathsf{tr}(\cdot)$ is invariant under similarity, i.e., $\mathsf{tr}(A) = \mathsf{tr}(P^{-1} \cdot A \cdot P)$ for any matrix A and invertible matrix P in $\mathsf{Mat}_{n\times n}(\mathbb{C})$.

▶ **Proposition 3.** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then $A \equiv_{\mathsf{ML}(\mathsf{tr},^*)} B$ if and only if A and B are unitary similar. When A and B are real matrices, then orthogonal similarity can be used.* ◀

Proposition 3 holds in particular for adjacency matrices representing digraphs, hereby generalising the characterisation for undirected graphs. We can say a bit more by rephrasing the trace conditions underlying Specht's Theorem in terms of so-called *semi-walks* in digraphs.

Let π be a string in $\{\leftarrow, \rightarrow\}^*$ of length k. A *semi-walk ρ of type π* in a digraph $G = (V, E)$ is a sequence of $k + 1$ vertices $v_1, v_2, \dots, v_{k+1}$ in V such that for each pair of consecutive vertices v_i and v_{i+1}, (v_i, v_{i+1}) is an edge in G if $\pi_i =$"$\rightarrow$" or (v_{i+1}, v_i) is an edge if $\pi_i =$"$\leftarrow$". A *closed* semi-walk of type π is a semi-walk of that type which starts and ends in the same vertex. Let $w(x, y)$ be a word of length k over variables x and y. We define the *type of* $w(x, y)$ as the string $\pi(w) \in \{\leftarrow, \rightarrow\}^k$ such that $\pi(w)_i =$"$\rightarrow$" if the ith symbol in $w(x, y)$ is x, and $\pi(w)_i =$"$\leftarrow$" if the ith symbol of $w(x, y)$ is y. It is now readily verified that $\mathsf{tr}(w(A_G, A_G^{\mathsf{t}}))$ counts the number of closed semi-walks of type $\pi(w)$ in the digraph G represented by adjacency matrix A_G.

▶ **Corollary 4.** *Let G and H be digraphs of the same order. Then, $G \equiv_{\mathsf{ML(tr,*)}} H$ if and only if G and H have the same number of closed semi-walks of any type.* ◀

We remark that when G and H are undirected graphs, semi-walks are simply walks and the type is just the length of the walk. In this case, Corollary 4 implies that $G \equiv_{\mathsf{ML(tr,*)}} H$ if and only if G and H have the same number of closed walks of any length, as reported in [20]. For undirected graphs, it was furthermore shown that $G \equiv_{\mathsf{ML(tr,*)}} H$ if and only if $G \equiv_{\mathsf{ML(tr)}} H$ [20]. After all, complex conjugate transposition on symmetric real matrices does not have any effect. By contrast, we will see that the presence of * has an impact on digraphs. First, however, we look into $\mathsf{ML(tr)}$-equivalence.

▶ **Proposition 5.** *Let A and B be in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML(tr)}} B$ if and only if A and B have the same characteristic polynomial.*

Proof. It is well-known that $\mathsf{tr}(A^k) = \mathsf{tr}(B^k)$, for any k, is equivalent to A and B having the same characteristic polynomial (and thus eigenvalues) (see e.g., Problem 2.4.P10 in [28]), and having $\mathsf{tr}(A^k) = \mathsf{tr}(B^k)$, for any k, is clearly equivalent to $A \equiv_{\mathsf{ML(tr)}} B$. ◀

We can complement this proposition for digraphs G and H by: $G \equiv_{\mathsf{ML(tr)}} H$ if and only if G and H have the same number of closed semi-walks of type "$\rightarrow^k$", for any k.

Proposition 5 immediately implies that a simple similarity-based characterisation of $\mathsf{ML(tr)}$-equivalence does not exist. Indeed, suppose that $A \equiv_{\mathsf{ML(tr)}} B$ would be equivalent to $A \cdot X = X \cdot B$ for some unitary/orthogonal matrix X, then A and B must have the same Jordan normal form (up to reordering of the Jordan blocks). Matrices with the same characteristic polynomial, however, do not necessarily have the same Jordan normal form.

▶ **Example 6.** Consider the digraphs G_1 ($\bullet\!\!\rightarrow\!\!\bullet\quad\bullet$) and H_1 ($\bullet\!\!\rightarrow\!\!\bullet\!\!\rightarrow\!\!\bullet$) with adjacency matrices $A_{G_1} = \left[\begin{smallmatrix}0&1&0\\0&0&0\\0&0&0\end{smallmatrix}\right]$ and $A_{H_1} = \left[\begin{smallmatrix}0&1&0\\0&0&1\\0&0&0\end{smallmatrix}\right]$, respectively. These matrices are in Jordan normal form, but different. So A_{G_1} and A_{H_1} cannot be similar using an invertible matrix. From the diagonals, however, we can see that both have z^3 (eigenvalue 0 with multiplicity 3) as characteristic polynomial. Hence, $A_{G_1} \equiv_{\mathsf{ML(tr)}} A_{H_1}$ by Proposition 5 (alternatively, one simply observes that both digraphs have no closed semi-walks of type "$\rightarrow^k$" for any k.). ◀

We next show that $\mathsf{ML(tr)}$- and $\mathsf{ML}(^*, \mathsf{tr})$-equivalence of *normal matrices* relate just like for undirected graphs. This is not surprising. Normal matrices are known to inherit many properties of symmetric matrices (see e.g., Section 2.5 in [28]).

▶ **Proposition 7.** *Let A and B be normal matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML(tr,*)}} B$ if and only if $A \equiv_{\mathsf{ML(tr)}} B$.*

Proof. (sketch) We note that $A \equiv_{\mathsf{ML(tr,*)}} B$ trivially implies $A \equiv_{\mathsf{ML(tr)}} B$. The reverse implication holds because if A and B are normal matrices, then $A^* = p(A)$ and $B^* = p(B)$ for some polynomial $p(z)$ (see e.g., Problem 2.5.P26 in [28]). Intuitively, this implies that we can eliminate occurrences of A^* in $\mathsf{tr}(w(A, A^*))$, hereby reducing such expressions to linear combinations of $\mathsf{tr}(A^k)$ for some k's, and $\mathsf{ML(tr)}$-equivalence guarantees that $\mathsf{tr}(A^k) = \mathsf{tr}(B^k)$ for all k. Specht's Theorem and Proposition 3 then imply that $A \equiv_{\mathsf{ML(tr,*)}} B$. ◀

The digraphs G_1 ($\bullet\!\!\rightarrow\!\!\bullet\quad\bullet$) and H_1 ($\bullet\!\!\rightarrow\!\!\bullet\!\!\rightarrow\!\!\bullet$), with non-normal adjacency matrices, show that Proposition 7 does not hold in general. Indeed, note that G_1 has one closed semi-walk of type "$\rightarrow\leftarrow$", whereas H_1 has two such walks. Hence, $G_1 \not\equiv_{\mathsf{ML(tr,*)}} H_1$ by Corollary 4.

$\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})$-equivalence

Whereas the trace operator enables counting closed semi-walks in (di)graphs, the inclusion of the $\mathbb{1}(\cdot)$-operator enables to count the number of not necessarily closed (semi-)walks. Indeed, one can use sentences $(\mathbb{1}(X))^* \cdot w(X, X^*) \cdot \mathbb{1}(X)$ in $\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})$ to count the number of semi-walks of the type of the word $w(x, y)$ when evaluated on an adjacency matrix of a digraph. It was shown that for undirected graphs, $G \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})} H$ if and only if $A_G \cdot Q = Q \cdot A_H$ for an orthogonal doubly quasi-stochastic matrix Q if and only if G and H have the same number of closed and not necessarily closed walks of any length [20]. The proof relied on the Spectral Theorem for symmetric matrices. Our proof strategy, however, allows to generalise this result to digraphs and general matrices.

Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. To apply Specht's Theorem we use the sequence $\Sigma = e_1(X) := X, e_2(X) := X^*, e_3(X) := \mathbb{1}(X) \cdot (\mathbb{1}(X))^*$. Hence, $\Sigma(A) = A, A^*, J$ and $\Sigma(B) = B, B^*, J$ and $A \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})} B$ implies $A \cdot U = U \cdot B$ (and thus also $A^* \cdot U = U \cdot B^*$) and $J \cdot U = U \cdot J$ for a unitary matrix U. We note that $J \cdot U = U \cdot J$ implies that U can be chosen such that $A \cdot U = U \cdot B$ and $U \cdot \mathbb{1} = \mathbb{1}$ hold (see Lemma 4 in [42]). In other words, U can be chosen to be unitary and doubly quasi-stochastic. It is easily verified, along the same lines as in [20], by induction on the structure of expressions, that the existence of such a unitary matrix also implies $A \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})} B$. We thus have shown:

► **Proposition 8.** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then $A \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})} B$ if and only if $A \cdot U = U \cdot B$ for a unitary doubly quasi-stochastic matrix U. When A and B are real matrices, we can use an orthogonal doubly quasi-stochastic matrix instead.* ◄

Specialised to adjacency matrices of digraphs G and H, we can further complement this by:

► **Corollary 9.** *Let G and H be digraphs of the same order. Then, $G \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})} H$ if and only if G and H have the same number of semi-walks of any type and the same number of closed semi-walks of any type.*

Proof. (sketch) The only if direction requires some explanation. Suppose that A_G and A_H have the same number of semi-walks of any type and the same number of closed semi-walks of any type. We argue that $\mathsf{tr}(w(A_G, A_G^*, J)) = \mathsf{tr}(w(A_H, A_H^*, J))$ for any word $w(x, y, z)$. Specht's Theorem together with Proposition 8 then imply that $G \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})} H$. It is easily verified that $\mathsf{tr}(w(A_G, A_G^*, J))$ is either of the form $\mathsf{tr}(w'(A_G, A_G^*))$ (when J does not occur) or can be reduced to an expression of the form $c \prod_{i\in[1,k]} \mathsf{tr}(w_i(A_G, A_G^*) \cdot J)$ for some $c \in \mathbb{N}$. We note that $\mathsf{tr}(w_i(A_G, A_G^*) \cdot J) = \mathbb{1}^{\mathsf{t}} \cdot w_i(A_G, A_G^*) \cdot \mathbb{1}$. Hence, in both cases $\mathsf{tr}(w(A_G, A_G^*, J))$ is fully determined by the number of semi-walks and closed semi-walks in G. Similarly, for $\mathsf{tr}(w(A_H, A_H^*, J))$. ◄

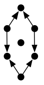
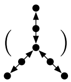
For undirected graphs, $G \equiv_{\mathsf{ML}(\mathsf{tr},{}^*,\mathbb{1})} H$ was also shown to be equivalent to $G \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})} H$ [20] and to A_G and A_H, and their complements $\bar{A}_G$ and $\bar{A}_H$, having the same characteristic polynomial [43]. Here, the *complement* $\bar{A}$ of a matrix A is defined as $J - A - I$, similarly for $\bar{B}$ of B. The latter equivalence extends more generally:

► **Proposition 10.** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$ if and only if A and B, and $\bar{A}$ and $\bar{B}$ have the same characteristic polynomial.*

Proof. This is an immediate consequence of the following identity (see e.g., [21]) linking characteristic polynomials of A and $\bar{A}$ to the walk generating function: $\frac{p_A(z)}{p_{\bar{A}}(z)} = 1 - \sum_{k\geq 0}(-z-1)^k \mathsf{tr}(A^k \cdot J)$, where $p_A(z)$ and $p_{\bar{A}}(z)$ denote the characteristic polynomials of A and $\bar{A}$, respectively. Indeed, when $A \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1})} B$ holds, $p_A(z) = p_B(z)$ (by Proposition 5) and also

$\mathsf{tr}(A^k \cdot J) = \mathsf{tr}(B^k \cdot J)$. Hence, the identity above tells that $p_{\bar{A}}(z) = p_{\bar{B}}(z)$. Conversely, if $p_A(z) = p_B(z)$ and $p_{\bar{A}}(z) = p_{\bar{B}}(z)$ we must have that $\mathsf{tr}(A^k \cdot J) = \mathsf{tr}(B^k \cdot J)$. It is readily verified, as in the proof of Corollary 9, that this implies that $A \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$ holds. ◀

Clearly, for digraphs, $\mathsf{ML}(\mathsf{tr}, \mathbb{1}, \mathbb{1}^{\mathsf{t}})$-equivalence coincides with the digraphs having the same number of semi-walks and closed semi-walks of type "$\rightarrow^k$", for any k. However, no simple similarity-based characterisation of $\mathsf{ML}(\mathsf{tr}, \mathbb{1}, \mathbb{1}^{\mathsf{t}})$-equivalence for digraphs exists.

▶ **Example 11.** We use again the Jordan normal form argument as in Example 6. Indeed, the digraphs G_2 () and H_2 () are easily seen to be $\mathsf{ML}(\mathsf{tr}, \mathbb{1}, \mathbb{1}^{\mathsf{t}})$-equivalent (they have no closed semi-walks of type "$\rightarrow^k$", have both 7 semi-walks of length 0, 6 semi-walks of type "$\rightarrow$" and no semi-walks of type "$\rightarrow^k$", for $k > 1$). Nevertheless, one can verify that their Jordan normal forms are different. So no invertible matrix X can exist such that $A_{G_2} \cdot X = X \cdot A_{H_2}$. ◀

We can say a bit more by allowing more complicated ways of linking A and B. We develop this further in Section 6 when focusing on $\mathsf{ML}(\mathbb{1}, \mathbb{1}^{\mathsf{t}})$-equivalence. So, stay tuned.

A similar proof as for Proposition 7 shows that normal matrices simplify matters:

▶ **Corollary 12.** *Let A and B be normal matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1})} B$ if and only if $A \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$.* ◀

We thus recover and generalise the characterisation for $\mathsf{ML}(\mathsf{tr}, \mathbb{1}, \mathbb{1}^{\mathsf{t}})$-equivalence of undirected graphs [20] to normal matrices. We note that graphs G_2 and H_2 show that Corollary 12 does not extend to non-normal matrices. Indeed, $G_2 \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})} H_2$ but $G_2 \not\equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1})} H_2$. For example, G_2 has 8 semi-walks of type "$\leftarrow\rightarrow\leftarrow\rightarrow$" while H_2 has 48 semi-walks of that type. So, Corollary 9 implies that $G_2 \not\equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1})} H_2$.

$\mathsf{ML}(\mathsf{tr}, ^*, \mathbb{1}, \odot_v)$-equivalence

We next include pointwise vector multiplication ($\odot_v$), i.e., the Schur-Hadamard product on vectors. The proof of the characterisation of $\mathsf{ML}(\mathsf{tr}, ^*, \mathbb{1}, \odot_v)$-equivalence obtained for undirected graphs in [20] generalises easily to digraphs and general matrices. The key insight for undirected graphs was that pointwise multiplication of vectors allows to compute so-called *equitable partitions* of undirected graphs. On graphs, equitable partitions correspond to the partition obtained by vertex colour refinement [23]. For digraphs and general matrices, equitable partitions and colour refinement have been considered as well (see e.g., [1, 25, 42, 24]). We only need to use this general notion of equitable partition, and the proofs in [20] carry over almost verbatim.

Let $A \in \mathsf{Mat}_{n\times n}(\mathbb{C})$. A partition $[n] = \biguplus_{i\in[1,q]} V_i$ is *row-equitable* for A if there are complex numbers r_{ij} such that for all $k \in V_i$, $\sum_{\ell\in V_j} A_{k\ell} = r_{ij}$. That is, the sum of row entries of A for columns in V_j is constant (r_{ij}) and independent of the chosen row in V_i. Similarly, a partition $[n] = \biguplus_{i\in[1,q]} V_i$ is *column-equitable* for A if there are complex numbers c_{ij} such that for all $k \in V_i$, $\sum_{\ell\in V_j} A_{\ell k} = c_{ij}$. A partition $[n] = \biguplus_{i\in[1,q]} V_i$ is *equitable* for A if it is both row- and column-equitable[2] for A. We remark that any matrix has an equitable partition given by the trivial one consisting of singleton elements.

[2] For symmetric matrices, such as adjacency matrices of undirected graphs, the notions of row- and column-equitability coincide.

In the following we represent partitions by *indicator vectors.* More specifically, if $[n] = \biguplus_{i\in[1,q]} V_i$ is a partition, then we represent it by binary vectors $\mathbb{1}_{V_1},\dots,\mathbb{1}_{V_q}$ such that $\mathbb{1}_{V_i}$ holds a 1 at row j if $j \in V_i$ and holds a 0 otherwise.

A matrix may have multiple equitable partitions, but only a unique *coarsest* one. That is, there is unique equitable partition of which any other equitable partition is a refinement. We can relate two matrices based on equitable partitions. More precisely, matrices A and B in $\mathsf{Mat}_{n\times n}(\mathbb{C})$ are said to have a *common equitable partition* if there exists partitions $[n] = \biguplus_{i\in[1,q]} V_i$ and $[n] = \biguplus_{i\in[1,q]} W_j$ which are equitable for A and B, respectively, and if r_{ij} and r'_{ij} denote the complex numbers for row-equitability of the two partitions, and c_{ij} and c'_{ij} the complex numbers for column-equitability, then $r_{ij} = r'_{ij}$ and $c_{ij} = c'_{ij}$ for $i,j \in [1,q]$. We can compute equitable partitions in $\mathsf{ML}(^*,\mathbb{1},\odot_v)$ (the trace operator is not needed) and we can test for the existence of a common equitable partition:

▶ **Proposition 13** **([20]).** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Let $[n] = \biguplus_{i\in[1,q]} V_i$ be the coarsest equitable partition for A. There exists expressions $\mathsf{equit}_j(X) \in \mathsf{ML}(\mathbb{1},^*,\odot_v)$, depending on A, such that $\mathbb{1}_{V_j} = \mathsf{equit}_j(A)$ for $j \in [1,q]$. Furthermore, $A \equiv_{\mathsf{ML}(^*,\mathbb{1},\odot_v)} B$ implies that A and B have a common equitable partition witnessed by the partitions represented by $\mathsf{equit}_i(A)$ and $\mathsf{equit}_i(B)$, for $i \in [1,q]$.*

Proof. Compared to the proof for undirected graphs [20] we now need to use * to ensure both row- and column-equitability. The presence of $\odot_v$ allows one to simulate the colour refinement process on matrices (see e.g., [24, 42]) by: extracting the indicator vectors from matrices, and by intersecting indicator vectors in order to create a refined partition. ◀

A similarity-based characterisation of $\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)$-equivalence is obtained using Specht's Theorem. We consider the sequence $\Sigma = e_1(X) := X, e_2(X) := X^*, \mathsf{equit}_i(X)\cdot\mathsf{equit}_i^*(X), i \in [1,q]$, such that $\mathsf{equit}_i(A)$ computes indicator vectors of an equitable partition $[n] = \biguplus_{i\in[1,q]} V_i$ of A. Hence, $\Sigma(A) = A, A^*, E_1 = \mathbb{1}_{V_1}\cdot\mathbb{1}_{V_1}^{\mathsf{t}},\dots,E_q = \mathbb{1}_{V_q}\cdot\mathbb{1}_{V_q}^{\mathsf{t}}$ and $\Sigma(B) = B, B^*, F_1 = \mathbb{1}_{W_1}\cdot\mathbb{1}_{W_1}^{\mathsf{t}},\dots,F_q = \mathbb{1}_{W_q}\cdot\mathbb{1}_{W_q}^{\mathsf{t}}$, where $[n] = \biguplus_{i\in[1,q]} W_i$ is an equitable partition of B. We have that $A \equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)} B$ implies the existence of a unitary matrix U such that $A\cdot U = U\cdot B$ and $E_i\cdot U = U\cdot F_i$ for $i \in [1,q]$. The latter conditions imply that U can be chosen such that $\mathbb{1}_{V_i} = U\cdot\mathbb{1}_{W_i}$ for $i \in [1,q]$. That the existence of such a similarity between A and B implies that $A \equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)} B$ holds, is not straightforward but the argument given in [20] can be generalised to general matrices. We only need a generalisation of Lemma 2.1 in [11] which states, translated to our setting, that all vectors $e(A)$ which can be computed by means of expressions $e(X)$ in $\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)$ can be written as a linear combination of indicator vectors $\mathbb{1}_{V_i}$, for $i \in [1,q]$. Similarly for $e(B)$. Since $\mathbb{1}_{V_i} = U\cdot\mathbb{1}_{W_i}$, $e(A) = U\cdot e(B)$ and this can be used to show, by induction on the structure of expressions, that $A \equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)} B$.

▶ **Proposition 14.** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)} B$ if and only if $A\cdot U = U\cdot B$ and $\mathbb{1}_{V_i} = U\cdot\mathbb{1}_{W_i}$ for $i \in [1,q]$. For real matrices A and B one can use orthogonal matrices Q such that $\mathbb{1}_{V_i} = Q\cdot\mathbb{1}_{W_i}$, for $i \in [1,q]$* ◀

In the proposition, $\mathbb{1}_{V_i}$ and $\mathbb{1}_{W_i}$, for $i \in [1,q]$, witness that A and B have a common equitable partition. This exactly matches the characterisation given for undirected graphs in [20] but beware that row- and column-equitable partitions are used in this general setting. We observe that $\mathbb{1}_{V_i} = U\cdot\mathbb{1}_{W_i}$ implies that U is doubly quasi-stochastic (simply note that $\mathbb{1} = \sum_{i\in[1,q]}\mathbb{1}_{V_i} = \sum_{i\in[1,q]}\mathbb{1}_{W_i}$). For undirected graphs G and H, we also have that $G \equiv_{\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)} H$ if and only if $G \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}},\odot_v)} H$. This equivalence does not hold in general, however. We remark that $\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}},\odot_v)$- and $\mathsf{ML}(\mathsf{tr},^*,\mathbb{1},\odot_v)$-equivalence do coincide when normal matrices are considered, just as for undirected graphs. It suffices again to observe that * can be eliminated for normal matrices (see e.g., the proof of Proposition 7).

6 Fragments without the trace operator and/or complex conjugate transposition

It should be clear by now that the absence of complex conjugate transposition has an impact when digraphs and general matrices are concerned. Moreover, for fragments that do not support $\mathsf{tr}(\cdot)$, we cannot use the proof strategy based on Specht's Theorem. We follow a different route in this section, inspired by the theory of *linear systems* in control theory (see e.g., [13]) and the notion of *minimal realisation* in particular. The fragments we consider are: $\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})$, $\mathsf{ML}(^*,\mathbb{1})$, and $\mathsf{ML}(^*,\mathbb{1},\odot_v)$.

6.1 Minimal realisations

In control theory, one analyses linear systems described by a triple $\langle A,C,D\rangle$ with $A\in\mathsf{Mat}_{n\times n}(\mathbb{C})$, $C\in\mathsf{Mat}_{m\times n}(\mathbb{C})$, and $D\in\mathsf{Mat}_{n\times o}(\mathbb{C})$, and one wants to understand the "dynamics" $C\cdot A^k\cdot D$, for $k\geq 0$, of the system. To this aim, one typically finds a *minimal realisation* of $\langle A,C,D\rangle$. That is, a system $\langle\hat{A},\hat{C},\hat{D}\rangle$ with $\hat{A}\in\mathsf{Mat}_{q\times q}(\mathbb{C})$, $\hat{C}\in\mathsf{Mat}_{m\times q}(\mathbb{C})$, and $\hat{D}\in\mathsf{Mat}_{q\times o}(\mathbb{C})$ such that (i) $C\cdot A^k\cdot D=\hat{C}\cdot\hat{A}^k\cdot\hat{D}$, for all $k\geq 0$, i.e., it has the same dynamics; and (ii) the dimension q, also called the *order of the system*, is minimal. The importance of minimal realisations is that they are *unique*, up to unitary similarity. That is, for any two minimal realisations $\langle\hat{A},\hat{C},\hat{D}\rangle$ and $\langle\hat{A}',\hat{C}',\hat{D}'\rangle$ of $\langle A,C,D\rangle$, there exists a unitary matrix $Z\in\mathsf{Mat}_{q\times q}(\mathbb{C})$ such that $\hat{A}=Z^*\cdot\hat{A}'\cdot Z$, $\hat{C}\cdot Z=\hat{C}'$ and $\hat{D}=Z^*\cdot\hat{D}'$. When real matrices are concerned, they are unique up to orthogonal similarity. If one has given two systems $\langle A,C,D\rangle$ and $\langle B,C',D'\rangle$ that have the same dynamics, i.e., $C\cdot A^k\cdot D=C'\cdot B^k\cdot D'$, for $k\geq 0$, then this implies that $\langle A,C,D\rangle$ and $\langle B,C',D'\rangle$ have the same minimal realisation (up to similarity). We use this observation to link the matrices A and B. To see how this relates to $\mathsf{ML}(\mathcal{L})$-equivalence, consider the following examples.

▶ **Example 15.** Let A and B in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A\equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})}B$ implies $\mathbb{1}^{\mathsf{t}}\cdot A^k\cdot\mathbb{1}=\mathbb{1}^{\mathsf{t}}\cdot B^k\cdot\mathbb{1}$ for all $k\geq 0$. After all, we can consider sentences $e_k(X):=\mathbb{1}^{\mathsf{t}}(X)\cdot X^k\cdot\mathbb{1}(X)$ in $\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})$ and $A\equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})}B$ implies that $e_k(A)=e_k(B)$. So, when we consider the systems $\langle A,\mathbb{1}^{\mathsf{t}},\mathbb{1}\rangle$ and $\langle B,\mathbb{1}^{\mathsf{t}},\mathbb{1}\rangle$, $A\equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})}B$ implies that these have the same dynamics and hence, similar minimal realisations. ◀

We note that for $\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})$-equivalence we only need to consider systems of the form $\langle A,C,D\rangle$ with $C=D^*$. From now on, we denote such a system by $\langle A,D\rangle$ instead of $\langle A,D^*,D\rangle$.

▶ **Example 16.** Let A and B in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A\equiv_{\mathsf{ML}(^*,\mathbb{1})}B$ implies $\mathbb{1}^{\mathsf{t}}\cdot w(A,A^*)\cdot\mathbb{1}=\mathbb{1}^{\mathsf{t}}\cdot w(B,B^*)\cdot\mathbb{1}$ for all words $w(x,y)$. We are here thus interested in the same systems as before, i.e., $\langle A,\mathbb{1}\rangle$ and $\langle B,\mathbb{1}\rangle$, but have in mind a more general notion of "dynamics" in which arbitrary words $w(x,y)$ are considered. ◀

One can show, following closely the proof for standard minimal realisations (see e.g., Chapter 25 in [14]), that results from control theory extend to this more general setting, provided that we define a *generalised minimal realisation* of $\langle A,D\rangle$ as a system $\langle\hat{A},\hat{D}\rangle$ of minimal order (i.e., the dimension of $\hat{A}$ is minimal) and such that $D^*\cdot w(A,A^*)\cdot D=(\hat{D})^*\cdot w(\hat{A},(\hat{A})^*)\cdot\hat{D}$ for every word $w(x,y)$. One can show that:

▶ **Proposition 17.** *Let $A\in\mathsf{Mat}_{n\times n}(\mathbb{C})$ and $D\in\mathsf{Mat}_{n\times o}(\mathbb{C})$. Then, $\langle A,D\rangle$ has a unique generalised minimal realisation, up to unitary similarity. Furthermore, orthogonal similarity can be used when A and D are real matrices.* ◀

Standard minimal realisations can be computed in several ways. We here use the *Kalman decomposition* method [13]. Let $\langle A, D\rangle$ and $\langle B, D'\rangle$ be two systems such that $D^* \cdot A^k \cdot D = (D')^* \cdot B^k \cdot D'$, for any k. The Kalman decomposition procedure can be used to obtain two special unitary bases U and V in $\mathsf{Mat}_{n\times n}(\mathbb{C})$, leveraging that minimal realisations are similar, such that:

$$U^* \cdot A \cdot U = \begin{bmatrix} \bigstar_{p\times p} & \bigstar_{p\times q} & \bigstar_{p\times r} \\ O_{q\times p} & \hat{A} & \bigstar_{q\times r} \\ O_{r\times p} & O_{r\times q} & \bigstar_{r\times r}\end{bmatrix}, V^* \cdot B \cdot V = \begin{bmatrix} \bigstar_{p'\times p'} & \bigstar_{p'\times q} & \bigstar_{p'\times r'} \\ O_{q\times p'} & \hat{A} & \bigstar_{q\times r'} \\ O_{r'\times p'} & O_{r'\times q} & \bigstar_{r'\times r'}\end{bmatrix},$$

$$U^* \cdot D = \begin{bmatrix} \bigstar_{p\times o} \\ \hat{D} \\ O_{r\times o}\end{bmatrix}, \text{ and } V^* \cdot D' = \begin{bmatrix} \bigstar_{p'\times o} \\ \hat{D} \\ O_{r'\times o}\end{bmatrix},$$

where each occurrence of $\bigstar$ represents a different matrix of dimensions specified in the subscripts. We can see that a minimal realisation $\langle \hat{A}, \hat{D}\rangle$ (of order q) is embedded in these matrices, albeit in different positions. The latter can be phrased by means of matrix transformations. More specifically, one can show that

$$P_A \cdot A \cdot S = S \cdot B \cdot P_B, \tag{1}$$

where P_A and P_B are matrices representing *orthogonal projection operators* on the *controllable and observable spaces* of $\langle A, D\rangle$ and $\langle B, D'\rangle$, respectively, and S is a matrix such that

$$D = S \cdot D' \text{ and } D^* \cdot S = (D')^*. \tag{2}$$

Intuitively, the controllable and observable space of a system $\langle A, D\rangle$, denoted by $\mathsf{CO}_{\langle A,D\rangle}$, consists of all vectors obtained by a linear combination of $A^i \cdot D$, for varying i, and which are *not* in the null space of vectors obtained by a linear combination of $D^* \cdot A^j$, for varying j. When interested in the dynamics $D^* \cdot A^k \cdot D$ only those vectors matter and, in fact, $\hat{A}$ is the matrix representation of A restricted to this space[3]. So, equation (1) is just stating that A and B are related (by matrix S) after appropriate projections are in place. Using properties of projection operators P_A and P_B, and (2), the following can be verified:

▶ **Theorem 18.** *Let A and B be in $\mathit{Mat}_{n\times n}(\mathbb{C})$ and D and D' in $\mathsf{Mat}_{n\times o}(\mathbb{C})$. Then, $D^* \cdot A^k \cdot D = (D')^* \cdot B^k \cdot D'$, for any k, if and only if there exists a matrix S such that $D = S \cdot D'$ and $D^* \cdot S = (D')^*$ and such that $P_A \cdot A \cdot S = S \cdot B \cdot P_B$ for projection operators P_A and P_B on $\mathsf{CO}_{\langle A,D\rangle}$ ad $\mathsf{CO}_{\langle B,D'\rangle}$, respectively.* ◀

We can further show that this result also holds for our general notion of dynamics, using a *generalised notion of controllable and unobservable space* based on words rather than powers of matrices.

▶ **Proposition 19.** *Let A and B be in $\mathit{Mat}_{n\times n}(\mathbb{C})$ and D and D' in $\mathsf{Mat}_{n\times o}(\mathbb{C})$. Then, $D^* \cdot w(A, A^*) \cdot D = (D')^* \cdot w(B, B^*) \cdot D'$ for all words $w(x, y)$ if and only if there exists a matrix S such that $D = S \cdot D'$ and $D^* \cdot S = (D')^*$ and such that $P_A \cdot A \cdot S = S \cdot B \cdot P_B$ and $P_A \cdot A^* \cdot S = S \cdot B^* \cdot P_B$ for projection operators P_A and P_B on $\widetilde{\mathsf{CO}}_{\langle A,D\rangle}$ and $\widetilde{\mathsf{CO}}_{\langle B,D'\rangle}$, respectively.* ◀

Here, $\widetilde{\mathsf{CO}}$ indicates that we work with the generalised CO space. In all this, real matrices can be used when A, B, D and D' are real. These results *directly translate* into characterisations of $\mathsf{ML}(\mathcal{L})$-equivalence for the fragments considered in this section.

[3] We refer to any textbook on linear systems, such as [13], for more background.

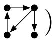
6.2 Results

$\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})$-equivalence

For undirected graphs, $G \equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})} H$ if and only if $A_G \cdot S = S \cdot A_H$ for a doubly quasi-stochastic matrix S if and only if G and H have the same number of walks of any length [20]. For general matrices, we can say the following. Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. From Example 15, we know that $A \equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$ implies that the systems $\langle A, \mathbb{1}\rangle$ and $\langle B, \mathbb{1}\rangle$ have the same dynamics. A direct application of Theorem 18 results in:

▶ **Corollary 20.** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$ if and only if there is a doubly quasi-stochastic matrix S such that $P_A \cdot A \cdot S = S \cdot B \cdot P_B$ for projection operators P_A and P_B.* ◀

So, the only difference with the undirected graph case is the use of the projection operators. This cannot be avoided, as shown in the example below. We also observe that for digraphs, $G \equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})} H$ is clearly equivalent to G and H having the same number of semi-walks of type "$\to^k$" for any k, similar as in the undirected graph case.

▶ **Example 21.** Consider directed graphs G_4 () and H_4 ( ). Both have 4 semi-walks of type "$\to^k$" for any k, i.e., $\mathbb{1}^{\mathsf{t}} \cdot A^k_{G_4} \cdot \mathbb{1} = 4 = \mathbb{1}^{\mathsf{t}} \cdot A^k_{H_4} \cdot \mathbb{1}$, for all $k \geq 0$. It is an easy exercise to show that there is no doubly quasi-stochastic matrix S such that $A_{G_4} \cdot S = S \cdot A_{H_4}$. We thus have to rely on Corollary 20. It can be verified that the minimal realisations of $\langle A_{G_4}, \mathbb{1}\rangle$ and $\langle A_{H_4}, \mathbb{1}\rangle$ consist of $\langle \hat{A} = [1], \hat{\mathbb{1}} = [2]\rangle$. So, indeed, $4 = [2]^{\mathsf{t}} \cdot [1]^k \cdot [2]$ for any k, as desired. Furthermore,

$$\underbrace{\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}}_{P_{A_{G_4}}} \cdot A_{G_4} \cdot \underbrace{\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}}_{S} = \underbrace{\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}}_{S} \cdot A_{H_4} \cdot \underbrace{\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}}_{P_{H_4}}.$$

That the same matrices are being used on both sides is just specific for this example. ◀

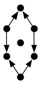
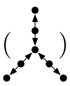
Corollary 20 begs the question why no projection operators are required when undirected graphs are concerned. The reason is that for normal matrices, the transformation matrices involved in the proof of Theorem 18 can be chosen to consist of eigenvectors of A and B (recall that normal matrices have n independent eigenvectors). This allows to simplify the expression $P_A \cdot A \cdot S = S \cdot B \cdot P_B$ into $A \cdot S = S \cdot B$.

▶ **Proposition 22.** *Let A and B be normal matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML}(*,\mathbb{1})} B$ if and only if $A \equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$ if and only if there exists a doubly quasi-stochastic matrix S such that $A \cdot S = S \cdot B$. For real matrices, S can be assumed to be real.* ◀

This is again analogous to the undirected graph case. We anticipated in Section 5 that will say something more about $\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})$-equivalence. One can verify, by an analysis of expressions, that $A \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$ if and only if $A \equiv_{\mathsf{ML}(\mathsf{tr})} B$ and $A \equiv_{\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$. Hence, Proposition 10 and Corollary 20 result in:

▶ **Corollary 23.** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML}(\mathsf{tr},\mathbb{1},\mathbb{1}^{\mathsf{t}})} B$ if and only if A and B have the same characteristic polynomial and $P_A \cdot A \cdot S = S \cdot B \cdot P_B$ for a doubly quasi-stochastic matrix S.* ◀

In other words, compared to Proposition 10, we can replace the condition related to the complements $\bar{A}$ and $\bar{B}$ by the revised similarity condition.

▶ **Example 24.** The digraphs G_2 () and H_2 () from Example 11 are $\mathsf{ML}(\mathsf{tr}, \mathbb{1}, \mathbb{1}^{\mathsf{t}})$-equivalent. They have the same characteristic polynomial ($p_{A_{G_2}}(z) = -z^7 = p_{A_{H_2}}(z)$) and one can verify that the minimal realisation of $\langle A_{G_2}, \mathbb{1}\rangle$ and $\langle A_{H_2}, \mathbb{1}\rangle$ is given by $\hat{A} = \begin{bmatrix} \frac{6}{7} & -\frac{1}{7}(3\sqrt{3}) \\ \frac{4\sqrt{3}}{7} & -\frac{6}{7} \end{bmatrix}$ and $\hat{\mathbb{1}} = \begin{bmatrix} \sqrt{7} \\ 0 \end{bmatrix}$. We see that $\hat{\mathbb{1}}^{\mathsf{t}} \cdot \hat{\mathbb{1}} = 7$, $\hat{\mathbb{1}}^{\mathsf{t}} \cdot \hat{A} \cdot \hat{\mathbb{1}} = 6$ and for $k \geq 2$, $\hat{\mathbb{1}}^{\mathsf{t}} \cdot \hat{A}^k \cdot \hat{\mathbb{1}} = 0$ because $\hat{A}^2 = O$. This is in accordance with Example 11. We have seen that there is no orthogonal similarity between A_{G_2} and A_{H_2}. One can also verify that no doubly quasi-stochastic matrix links these matrices. We thus have to rely on Corollary 20 and one can show that $P_{A_{G_2}} \cdot A_{G_2} \cdot S = S \cdot A_{H_2} \cdot P_{A_{H_2}}$ with

$$P_{A_{G_2}} = P_{A_{H_2}} = \begin{bmatrix} \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \end{bmatrix},$$

and S the identity matrix. Again, that $P_{A_{G_2}} = P_{A_{H_2}}$ is just a coincidence. ◀

$\mathsf{ML}(^*, \mathbb{1})$-equivalence

Consider two matrices A and B in $\mathsf{Mat}_{n\times n}(\mathbb{C})$ and consider the systems $\langle A, \mathbb{1}\rangle$ and $\langle B, \mathbb{1}\rangle$. As we have seen in Example 16, $A \equiv_{\mathsf{ML}(^*,\mathbb{1})} B$ implies that $\mathbb{1}^{\mathsf{t}} \cdot w(A, A^*) \cdot \mathbb{1} = \mathbb{1}^{\mathsf{t}} \cdot w(B, B^*) \cdot \mathbb{1}$ for every word $w(x, y)$ and hence Proposition 19 applies, resulting in:

▶ **Proposition 25.** *Let A and B be in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then, $A \equiv_{\mathsf{ML}(^*,\mathbb{1})} B$ if and only if $P_A \cdot A \cdot S = S \cdot B \cdot P_B$ and $P_A \cdot A^* \cdot S = S \cdot B^* \cdot P_B$ for a doubly quasi-stochastic matrix S and for projection operators P_A and P_B.* ◀

We remark that P_A and P_B are now projection operators on the generalised controllable spaces of $\langle A, \mathbb{1}\rangle$ and $\langle B, \mathbb{1}\rangle$, respectively.

▶ **Example 26.** We do not have a digraph example at hand such that $G \equiv_{\mathsf{ML}(^*,\mathbb{1})} H$ holds and which shows the necessity of the projection operators in Proposition 25. All efforts resulted in $A_G \cdot S = S \cdot A_H$ and $A_G^{\mathsf{t}} \cdot S = S \cdot A_H^{\mathsf{t}}$ for a doubly quasi-stochastic matrix S, i.e., without needing the projection operators. Finding such a digraph example or showing that the projection operators can be eliminated is left as an open problem. ◀

$\mathsf{ML}(^*, \mathbb{1}, \odot_v)$-equivalence

For undirected graphs, $\mathsf{ML}(^*, \mathbb{1}, \odot_v)$-equivalence coincides with the graphs being fractionally isomorphic, with having a common equitable partition, and with being C^2-equivalent [37, 38, 10, 30, 20]. We recall that a fractional isomorphism between graphs G and H is a doubly stochastic matrix S such $A_G \cdot S = S \cdot A_H$.

When considering $\mathsf{ML}(^*, \mathbb{1}, \odot_v)$-equivalence of arbitrary matrices, we simply need to use the corresponding notions of equitable partitions as in Section 5 for $\mathsf{ML}(\mathsf{tr}, ^*, \mathbb{1}, \odot_v)$-equivalence. We can again use Proposition 19 to obtain a characterisation.

▶ **Example 27.** Let A, B in $\mathsf{Mat}_{n\times n}(\mathbb{C})$ and let $\mathbb{1}_{V_i}$ and $\mathbb{1}_{W_i}$, for $i \in [1,q]$, represent common equitable partitions of A and B, respectively. We can obtain these indicator vectors by expressions $\mathsf{equit}_i(X) \in \mathsf{ML}(^*,\mathbb{1},\odot_v)$, for $i\in[1,q]$ (Proposition 13). Hence, $A \equiv_{\mathsf{ML}(^*,\mathbb{1},\odot_v)} B$ implies that $\mathbb{1}_{V_i}^{\mathsf{t}} \cdot w(A,A^*)\cdot \mathbb{1}_{V_j} = \mathbb{1}_{W_i}^{\mathsf{t}}\cdot w(B,B^*)\cdot \mathbb{1}_{W_j}$ for every word $w(x,y)$ and any $i,j\in[1,q]$. By letting $D=[\mathbb{1}_{V_1} \,\dots\, \mathbb{1}_{V_p}]$ and $D'=[\mathbb{1}_{W_1} \,\dots\, \mathbb{1}_{W_p}]$, $A \equiv_{\mathsf{ML}(^*,\mathbb{1},\odot_v)} B$ implies that $D^*\cdot w(A,A^*)\cdot D = (D')^*\cdot w(B,B^*)\cdot D'$ for every word $w(x,y)$. Hence, $\langle A,D\rangle$ and $\langle B,D'\rangle$ have the same generalised dynamics. ◀

Hence, Proposition 19 implies that $A\equiv_{\mathsf{ML}(^*,\mathbb{1},\odot_v)} B$ if and only if $P_A\cdot A\cdot S = S\cdot B\cdot P_B$ and $P_A\cdot A^*\cdot S = S\cdot B^*\cdot P_B$ for a matrix S satisfying $\mathbb{1}_{V_i} = S\cdot \mathbb{1}_{W_i}$ and $\mathbb{1}_{V_i}^{\mathsf{t}}\cdot S = \mathbb{1}_{W_i}^{\mathsf{t}}$, for all $i\in[1,q]$. We remark again that this implies that S is doubly quasi-stochastic. We can do better, however, and eliminate the projection operators and ensure that S is doubly-stochastic and hence, A and B are fractionally isomorphic.

▶ **Corollary 28.** *Let A and B be matrices in $\mathsf{Mat}_{n\times n}(\mathbb{C})$. Then $A\equiv_{\mathsf{ML}(^*,\mathbb{1},\odot_v)} B$ if and only if there is a doubly stochastic matrix S such that $A\cdot S = S\cdot B$ and $A^*\cdot S = S\cdot B^*$, $\mathbb{1}_{V_i} = S\cdot\mathbb{1}_{W_i}$ and $\mathbb{1}_{V_i}^{\mathsf{t}}\cdot S = \mathbb{1}_{W_i}^{\mathsf{t}}$, for all $i\in[1,q]$, for indicator vectors describing a common equitable partition of A and B.*

Proof. (Sketch). It can be verified that the transformation matrices underlying the proof of Proposition 19 for the systems $\langle A,D\rangle$ and $\langle B,D'\rangle$ from Example 27 are of a very particular form. Indeed, using the fact that D and D' represent equitable partitions, the column vectors in these matrices can be shown to span the generalised controllable and observable space of $\langle A,D\rangle$ and $\langle B,D'\rangle$. As a consequence, one obtains that $P_A = \Pi\cdot\Pi^{\mathsf{t}}$, $P_V = \Pi'\cdot(\Pi')^{\mathsf{t}}$ and $S=\Pi\cdot(\Pi')^{\mathsf{t}}$, where $\Pi = D\cdot(D^{\mathsf{t}}\cdot D)^{-1/2}$ and $\Pi' = D'\cdot((D')^{\mathsf{t}}\cdot D')^{-1/2}$. Hence, $\Pi\cdot(\Pi^{\mathsf{t}})\cdot A\cdot\Pi\cdot(\Pi')^{\mathsf{t}} = \Pi\cdot(\Pi')^{\mathsf{t}}\cdot B\cdot \Pi'\cdot(\Pi')^{\mathsf{t}}$. A second crucial observation is that, in a similar way as shown in the proof of Theorem 4.1 in [24], one can verify that $\Pi\cdot\Pi^{\mathsf{t}}$ commutes with A and similarly, $\Pi'\cdot(\Pi')^{\mathsf{t}}$ commutes with B, due to equitability. Further manipulation then shows that P_A and P_B can be omitted, resulting in $A\cdot\Pi\cdot(\Pi')^{\mathsf{t}} = \Pi\cdot(\Pi')^{\mathsf{t}}\cdot B$. It now suffices to observe that for $S=\Pi\cdot(\Pi')^{\mathsf{t}}$, we have that $S_{vw} = \frac{1}{|V_k|}$ for the unique part V_k such that $i\in V_k$ and $j\in W_k$, and $S_{vw}=0$ otherwise. It is now easy to verify that S satisfies the conditions stated in the Corollary. ◀

For digraphs, the existence of a stochastic matrix S such that $A_G\cdot S = S\cdot A_H$ and $A_G^{\mathsf{t}}\cdot S = S\cdot A_H^{\mathsf{t}}$ hold is known to correspond to $G\equiv_{\mathsf{C}^2} H$ [1, 25]. Hence, the previous Corollary implies that $G\equiv_{\mathsf{ML}(^*,\mathbb{1},\odot_v)} H$ if and only if $G\equiv_{\mathsf{C}^2} H$, just as for undirected graphs. We also remark that Corollary 28 can be shown by relying on known correspondences between fractional isomorphisms and equitable partitions of matrices [1, 25]. Nevertheless, proving it by relying on minimal realisations (Proposition 19) further illustrates the usefulness of our approach.

▶ Remark 29. We can obtain similar results for $\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}},\odot_v)$-equivalence. In this case, we have to use standard linear systems described by $\langle A,C_c,D_r\rangle$ where C_c consists of transposed indicator vectors of a column-equitable partition of A and D_r consists of indicator vectors of a row-equitable partition of A. Note that C_c is not necessarily equal to the transpose of D_r. Then, given $\langle A,C_c,D_r\rangle$ and $\langle B,C_c',D_r'\rangle$, a generalisation of Theorem 18 allows to obtain a relationship between A and B. We defer the precise analysis of $\mathsf{ML}(\mathbb{1},\mathbb{1}^{\mathsf{t}},\odot_v)$-equivalence to future work. ◀

7 Concluding remarks

While at first, it seemed daunting to understand the distinguishing power of MATLANG on general matrices, we showed that moving to this more general setting (compared to adjacency matrices of undirected graphs) makes the analysis more elegant. Of course, the right tools are needed, such as the connection with Specht's Theorem and mininal realisations of linear systems. Considering the general setting has as additional advantage that previous results can be seen as special cases. Although we focused on the setting where MATLANG expressions only take a single matrix as input, some of our results can be generalised. This is particularly true for cases relying on Specht's Theorem. It is less clear how to deal with multiple inputs by relying on linear systems theory.

In this work, we consider equivalence of matrices by sentences that allow an arbitrary number of applications of the supported operators. In practice, one would like to understand the impact of allowing, say only k matrix multiplications. Indeed, each operator application has a computational cost attached. Developing the right tools for analysing such a quantified setting is, we believe, an interesting line of research.

References

1 Albert Atserias and Elitza N. Maneva. Sherali-Adams Relaxations and Indistinguishability in Counting Logics. *SIAM J. Comput.*, 42(1):112–137, 2013. doi:10.1137/120867834.

2 Sheldon Axler. *Linear Algebra Done Right.* Springer, third edition, 2015. doi:10.1007/978-3-319-11080-6.

3 Pablo Barceló, Nelson Higuera, Jorge Pérez, and Bernardo Subercaseaux. On the Expressiveness of LARA: A Unified Language for Linear and Relational Algebra. In *23nd International Conference on Database Theory, ICDT 2020*, volume 155 of *LIPIcs*, pages 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICDT.2020.6.

4 Matthias Boehm, Michael W. Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Arvind C. Surve, and Shirsih Tatikonda. SystemML: Declarative machine learning on Spark. *Proceedings of the VLDB Endowment*, 9(13):1425–1436, 2016. doi:10.14778/3007263.3007279.

5 Matthias Boehm, Arun Kumar, and Jun Yang. *Data Management in Machine Learning Systems.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019. doi:10.2200/S00895ED1V01Y201901DTM057.

6 Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Prithviraj Sen, Alexandre V. Evfimievski, and Niketan Pansare. On Optimizing Operator Fusion Plans for Large-scale Machine Learning in SystemML. *Proceedings of the VLDB Endowment*, 11(12):1755–1768, 2018. doi:10.14778/3229863.3229865.

7 Robert Brijder, Floris Geerts, Jan Van Den Bussche, and Timmy Weerwag. On the Expressive Power of Query Languages for Matrices. *ACM Trans. Database Syst.*, 44(4), 2019. doi:10.1145/3331445.

8 Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. On the Expressive Power of Query Languages for Matrices. In *21st International Conference on Database Theory*, ICDT, pages 10:1–10:17, 2018. doi:10.4230/LIPIcs.ICDT.2018.10.

9 Robert Brijder, Marc Gyssens, and Jan Van den Bussche. On Matrices and K-Relations. In Andreas Herzig and Juha Kontinen, editors, *Proceedings of 11th International Symposium on Foundations of Information and Knowledge Systems, FoIKS 2020*, volume 12012 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2020. doi:10.1007/978-3-030-39951-1_3.

10 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. `doi:10.1007/BF01305232`.

11 Ada Chan and Chris D. Godsil. *Symmetry and eigenvectors*, pages 75–106. Springer Netherlands, 1997. `doi:10.1007/978-94-015-8937-6_3`.

12 Lingjiao Chen, Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel. Towards Linear Algebra over Normalized Data. *Proceedings of the VLDB Endowment*, 10(11):1214–1225, 2017. `doi:10.14778/3137628.3137633`.

13 Martin J. Corless and Art Frazho. *Linear Systems and Control: An Operator Perspective*. Series: & Hall/CRC Pure and Applied Mathematics. CRC Press, 2003.

14 Mohammed Dahleh, Munther A. Dahleh, and George Verghese. Lectures Notes on Dynamic Systems and Control (MIT OpenCourseWare), 2004.

15 Anuj Dawar. On the Descriptive Complexity of Linear Algebra. In *Proceedings of the 15th International Workshop on Logic, Language, Information and Computation*, WoLLIC, pages 17–25, 2008. `doi:10.1007/978-3-540-69937-8_2`.

16 Anuj Dawar, Erich Grädel, and Wied Pakusa. Approximations of Isomorphism and Logics with Linear-Algebraic Operators. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 112:1–112:14, 2019. `doi:10.4230/LIPIcs.ICALP.2019.112`.

17 Anuj Dawar, Martin Grohe, Bjarki Holm, and Bastian Laubner. Logics with Rank Operators. In *Proceedings of the 24th Annual IEEE Symposium on Logic In Computer Science, LICS*, pages 113–122, 2009. `doi:10.1109/LICS.2009.24`.

18 Anuj Dawar and Bjarki Holm. Pebble games with algebraic rules. *Fund. Inform.*, 150(3-4):281–316, 2017. `doi:10.3233/FI-2017-1471`.

19 Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. Compressed linear algebra for large-scale machine learning. *The VLDB Journal*, pages 1–26, 2017. `doi:10.1007/s00778-017-0478-1`.

20 Floris Geerts. On the Expressive Power of Linear Algebra on Graphs. In *Proceedings of the 22nd International Conference on Database Theory, ICDT*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:19. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.ICDT.2019.7`.

21 Chris Godsil. Cospectral Graph Complements and Generating Function of Walks. Mathematics Stack Exchange. URL: `https://math.stackexchange.com/q/2600510`.

22 Erich Grädel and Wied Pakusa. Rank Logic is Dead, Long Live Rank Logic! In *Proceedings of the 24th EACSL Annual Conference on Computer Science Logic, CSL*, pages 390–404, 2015. `doi:10.4230/LIPIcs.CSL.2015.390`.

23 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017. `doi:10.1017/9781139028868`.

24 Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension Reduction via Colour Refinement. In *22th Annual European Symposium on Algorithms*, ESA, pages 505–516, 2014. `doi:10.1007/978-3-662-44777-2_42`.

25 Martin Grohe and Martin Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015. URL: `http://doi.org/10.1017/jsl.2015.28`.

26 Martin Grohe and Wied Pakusa. Descriptive complexity of linear equation systems and applications to propositional proof complexity. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–12, 2017. `doi:10.1109/LICS.2017.8005081`.

27 Bjarki Holm. *Descriptive Complexity of Linear Algebra*. PhD thesis, University of Cambridge, 2010.

28 Roger A. Horn and Charles R. Johnson. *Matrix Analysis 2nd ed.* Cambridge University Press, 2013.

29 Dylan Hutchison, Bill Howe, and Dan Suciu. LaraDB: A Minimalist Kernel for Linear and Relational Algebra Computation. In *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, BeyondMR, pages 2:1–2:10, 2017. doi:10.1145/3070607.3070608.

30 Neil Immerman and Eric Lander. Describing Graphs: A First-Order Approach to Graph Canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday*, pages 59–81. Springer, 1990. doi:10.1007/978-1-4612-4478-3_5.

31 Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J. Gao. Declarative Recursive Computation on an RDBMS. *Proceedings of the VLDB Endowment*, 12(7):822–835, 2019. URL: http://www.vldb.org/pvldb/vol12/p822-jankov.pdf, doi:10.14778/3317315.3317323.

32 Naihuan Jing. Unitary and orthogonal equivalence of sets of matrices. *Linear Algebra and its Applications*, 481:235–242, 2015. doi:10.1016/j.laa.2015.04.036.

33 Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-Database Learning with Sparse Tensors. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS*, pages 325–340, 2018. doi:10.1145/3196959.3196960.

34 Andreas Kunft, Alexander Alexandrov, Asterios Katsifodimos, and Volker Markl. Bridging the Gap: Towards Optimization Across Linear and Relational Algebra. In *Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, BeyondMR, pages 1:1–1:4, 2016. doi:10.1145/2926534.2926540.

35 Andreas Kunft, Asterios Katsifodimos, Sebastian Schelter, Tilmann Rabl, and Volker Markl. BlockJoin: Efficient Matrix Partitioning Through Joins. *Proceedings of the VLDB Endowment*, 10(13):2061–2072, 2017. doi:10.14778/3151106.3151110.

36 Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, and Christopher M. Jermaine. Scalable Linear Algebra on a Relational Database System. *IEEE Trans. Knowl. Data Eng.*, 31(7):1224–1238, 2019. doi:10.1109/TKDE.2018.2827988.

37 Motakuri V. Ramana, Edward R. Scheinerman, and Daniel Ullman. Fractional isomorphism of graphs. *Discrete Mathematics*, 132(1-3):247–265, 1994. doi:10.1016/0012-365X(94)90241-0.

38 Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory: a Rational Approach to the Theory of Graphs*. John Wiley & Sons, 1997. URL: https://www.ams.jhu.edu/ers/wp-content/uploads/sites/2/2015/12/fgt.pdf.

39 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning Linear Regression Models over Factorized Joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD, pages 3–18, 2016. doi:10.1145/2882903.2882939.

40 Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q. Ngo, and XuanLong Nguyen. A Layered Aggregate Engine for Analytics Workloads. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019*, pages 1642–1659, 2019. doi:10.1145/3299869.3324961.

41 Yaroslav Shitov. An improved bound for the lengths of matrix algebras. *Algebra Number Theory*, 13(6):1501–1507, 2019. doi:10.2140/ant.2019.13.1501.

42 Mario Thüne. *Eigenvalues of matrices and graphs*. PhD thesis, University of Leipzig, 2012.

43 Erwin R. van Dam, Willem H. Haemers, and Jack H. Koolen. Cospectral graphs and the generalized adjacency matrix. *Linear Algebra and its Applications*, 423(1):33–41, 2007. doi:10.1016/j.laa.2006.07.017.

Distribution Constraints: The Chase for Distributed Data

Gaetano Geck

TU Dortmund University, Germany
gaetano.geck@tu-dortmund.de

Frank Neven

Hasselt University and transnational University of Limburg, Belgium
frank.neven@uhasselt.be

Thomas Schwentick

TU Dortmund University, Germany
thomas.schwentick@tu-dortmund.de

Abstract

This paper introduces a declarative framework to specify and reason about distributions of data over computing nodes in a distributed setting. More specifically, it proposes distribution constraints which are tuple and equality generating dependencies (tgds and egds) extended with node variables ranging over computing nodes. In particular, they can express co-partitioning constraints and constraints about range-based data distributions by using comparison atoms. The main technical contribution is the study of the implication problem of distribution constraints. While implication is undecidable in general, relevant fragments of so-called data-full constraints are exhibited for which the corresponding implication problems are complete for EXPTIME, PSPACE and NP. These results yield bounds on deciding parallel-correctness for conjunctive queries in the presence of distribution constraints.

2012 ACM Subject Classification Theory of computation → Database constraints theory; Theory of computation → Logic and databases; Information systems → Parallel and distributed DBMSs

Keywords and phrases tuple-generating dependencies, chase, conjunctive queries, distributed evaluation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.13

Related Version A full version of this paper is available on arXiv [27], `https://arxiv.org/abs/2003.00965`.

Acknowledgements We thank Michael Benedikt, Bas Ketsman, Andreas Pieris, Phokion Kolaitis, Christopher Spinrath, Brecht Vandevoort, and Thomas Zeume for helpful discussions on various aspects of this work.

1 Introduction

Distributed storage and processing of data has been used and studied since the 1970s and became more and more important in the recent past. One of the most fundamental questions in distributed data management is the following: *how should data be replicated and partitioned over the set of computing nodes?* It is paramount to answer this question well as the placement of data determines the reliability of the system and is furthermore critical for its scalability including the performance of query processing.

On the one hand, despite the importance of this question and decades of research, the placement strategies remained rather simple for a long time: horizontal or vertical fragmentation of relations – or hybrid variants thereof [37]. These placement strategies often require a reshuffling of the data for each binary join in the processed query which are commonly based on a range or hash partitioning of the relevant attributes. Recently, however, more elaborated schemes of data placement like co-partitioning, single hypercubes (for multiway-joins) or multiple hypercubes (for skewed data) gained some attention [3, 12, 30, 39, 41, 45].

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 13; pp. 13:1–13:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the other hand, there is a long tradition in studying tuple- and equality-generating dependencies (tgds/egds) as a simple but versatile tool to describe relationships among relational data. The research on these dependencies focuses mainly on the implication[1] problem. More precisely, since the implication problem in general is undecidable, several fragments have been considered in an attempt to locate the boundaries of decidability and complexity. Commonly, these fragments are defined by syntactical restrictions on the sets of dependencies, like weak acyclicity, weak guardedness, stickiness, wardedness, ... [15–17, 23].

It seems desirable to connect these two strands of research. Being able to reason about the placement of data offers database management systems additional optimisation potential, for instance, when it comes to the placement of new data or when the cost of a query execution plan is estimated. In the latter case, a reshuffling phase, which often dominates the processing time, can sometimes be omitted completely because the query at hand is already parallel-correct[2] under the current distribution.

The goal of this paper is to make a first step towards a connection between existing partitioning schemes and well-known reasoning frameworks. With this intent, we introduce *distribution constraints* – a variant of tgds/egds that is specifically geared towards distributed data – and study its implication problem. In particular, we identify fragments of distribution constraints by the complexity of the associated implication problem. Although the implication problem is certainly not the only – and, admittedly, not the most innovative – problem related to reasoning about distributed data, it is yet a basic problem that is likely to have connections to other algorithmical questions centering around this topic (like *how to derive a new distribution for the next query, making use of the current distribution?*).

Contributions. We start by defining *distribution constraints* as tgds and egds with atoms of the form $R(x,y)@\kappa$, in which κ is understood as a node variable with the intended meaning that fact $R(x,y)$ is at node κ. To achieve decidability, we further require that distribution tgds are *data-full*, i.e., only node variables may be quantified existentially.

We demonstrate that distribution constraints can express several common distribution schemes, incorporating range and hash partitionings [37], co-partitionings [21, 25], hierarchical partitionings (as used in Google's F1 [39, 41]), predicate-based reference partitionings [45], hypercube distributions [3, 12], and multi-round communication.

► **Example 1.** As an example, consider the following set of distribution tgds, describing a "derived horizontal" fragmentation [37] of relation Msg based on the Range-predicate and the message's sender id s:

$$\begin{aligned}&\texttt{Range}(\ell,u) \to \texttt{Range}(\ell,u)@\kappa,\\ &\texttt{Msg}(s,r) \to \texttt{Msg}(s,r)@\kappa,\\ &\texttt{Msg}(s,r)@\kappa, \texttt{Range}(\ell,u)@\lambda, \ell \leq s, s \leq u \to \texttt{Msg}(s,r)@\lambda\end{aligned}$$

The first two rules enforce that, for every Range- and every Msg-fact, there is a responsible node (indicated by the node variable κ). The third rule ensures that every Msg-fact can be found at every node whose Range-bounds match the sender id. We remark that the above set of constraints implies the following distribution tgd:

$$\texttt{Msg}(s_1,r), \texttt{Msg}(s_2,r), \texttt{Range}(\ell,u), \ell \leq s_1, s_1 \leq u, \ell \leq s_2, s_2 \leq u \to \texttt{Msg}(s_1,r)@\kappa, \texttt{Msg}(s_2,r)@\kappa,$$

[1] Does a dependency τ always hold if a set Σ of dependencies is satisfied, $\Sigma \models \tau$?

[2] Parallel correctness is a basic notion of distributed query evaluation [7], also addressed in Section 3.3.

which states that all pairs of messages with the same receiver can be found at a common node if their senders fall in the same range. In other words, if the above set of constraints is satisfied over a distributed instance, then so is the just mentioned dtgd.

On the technical level, we show that the implication problem is EXPTIME-complete for these constraints in general, and we identify classes of distribution constraints where the complexity drops to PSPACE or even NP and classes where this is not the case. These classes are determined by simple syntactic criteria based on the amount of data associated with node variables.

Since distribution constraints incorporate all *full* tgds (without existential quantification), EXPTIME-hardness of their implication problem readily follows from an early result by Chandra, Lewis and Makowsky [19]. However, the latter result relies on the use of relation atoms of arbitrarily high arity, while the EXPTIME-hardness results in this paper already hold for a fixed schema of maximum arity of 3 (or 2, w.r.t. data variables). The corresponding upper bounds are established by an adaptation of the standard chase procedure [23, 36].

The fragments studied here are defined depending on, first, the sizes of the node variables' *contexts* (the data variables occuring together with the node variable in some atom) and, second, on the distinction of data-collecting tgds and node-creating tgds (without/with existentially quantified node variable in the head). For a fixed integer b, a node variable has *bounded context* if its context size is at most b. Thanks to the obvious relationship between distribution constraints and standard constraints, the complexity results in this paper can also be viewed as results on fragments of standard tgds/egds.

Related work. There is a rich literature on restrictions of (sets of) tgds that yield a decidable (general and finite) implication problem [4, 36]. We discuss how our distributed constraints relate to classical constraints in Section 3. Restricting the use of existential variables in tgds is a common approach to define fragments of tgds that yield a decidable implication problem. Interestingly, the rather simple restriction to data-full dtgds studied here, is orthogonal to prominent examples like weak acyclicity, weak guardedness, stickiness and wardedness [15–17, 23].

Dependencies with arithmetic comparisons have been used in the context of Data Exchange [5, 43]. However, these papers mainly study full and weakly acyclic tgds and are thus orthogonal to our framework. There is further work on dependencies with stronger arithmetic constraints, e.g. [9, 11, 22, 33].

Declarative specifications for distributed data have also been studied before. Notable examples are Webdamlog and the already mentioned Data Exchange setting (which can be seen as a restricted form of distribution constraints with a global and a single local database). We refer to the book [8] for a relatively recent overview of Data Exchange.

Our notation $R(x)@\kappa$ for distributed atoms resembles that of Webdamlog, $R@\kappa(x)$, a dialect of datalog that was designed for distributed data management.[3] Besides implementing a system [2, 34] based on this dialect, the theoretical research on this language has mostly focussed on establishing a hierarchy among some of its fragments in terms of their expressiveness [3]. Neglecting the notational similarities, there seems to be no overlap between the research on Webdamlog – with its fixpoint evaluation mechanism (which even allows facts to vanish) – and the results on distribution constraints that we present in this paper.

[3] Annotated atoms have already been used before in Datalog dialects. For instance, in Dedalus [6], where they describe timestamps.

Particularly, Webdamlog seems to prohibit existential quantification of node variables and assumes, accordingly, that the number of nodes is explicitly fixed with the input. Distribution constraints, in contrast, *do allow* existential quantification of node variables, which affects the modeling capabilities and the complexity of the reasoning process.

Organisation of this paper. After providing the necessary preliminaries in Section 2, we formally define distribution constraints in Section 3, compare them with classical constraints, and give examples of their versatility. In Section 4, we define the implication problem and extend the standard chase to distribution constraints. In Section 5, we address the complexity of the implication problem and, finally, conclude in Section 6.

Due to space limitations, most proofs and some examples will be given in a full version, which we plan to publish on arXiv.

2 Preliminaries

In this section, we fix our notation for the basic concepts of this paper. Specific definitions for our framework are given in Section 3.

2.1 Databases and queries

Let dom and var be disjoint infinite sets of *data values* and *data variables*, respectively. For simplicity, we do not distinguish between different data types and assume that dom is linearly ordered. We denote data variables as usual with x, y, z, $\ldots$. A *schema* is a set $\mathcal{S}$ of relation symbols, where each relation symbol $R \in \mathcal{S}$ has some fixed arity $\mathsf{ar}(R)$. We write $\mathsf{ar}(\mathcal{S})$ for the *maximum* arity $\mathsf{ar}(R)$ of any $R \in \mathcal{S}$. A *relation atom over* $\mathcal{S}$ is of the form $R(t_1, \ldots, t_k)$ where R is a relation symbol of arity k and $t_1, \ldots, t_k \in \mathsf{dom} \cup \mathsf{var}$. A relation atom is a *fact* if $t_1, \ldots, t_k \in \mathsf{dom}$. A *comparison atom* is of the form $t < t'$ or $t \leq t'$ with $t, t' \in \mathsf{dom} \cup \mathsf{var}$. The set of data values occuring in a set $\mathcal{A}$ of (relational or comparison) atoms is denoted $\mathsf{adom}(\mathcal{A})$. Similarly, the set of variables in $\mathcal{A}$ is denoted $\mathsf{var}(\mathcal{A})$. Instances are finite sets of facts over a given schema $\mathcal{S}$.

A *valuation* for a set $\mathcal{A}$ of atoms is a mapping $V : \mathsf{var}(\mathcal{A}) \to \mathsf{dom}$. It *satisfies* $\mathcal{A}$ *on instance* I if $V(A) \in I$ holds for each relation atom $A \in \mathcal{A}$ and $V(t)\theta V(t')$ holds for each comparison atom $t\theta t'$ in $\mathcal{A}$. We often denote by V also the extension of V to dom defined by $V(a) = a$ for every $a \in \mathsf{dom}$.

A *conjunctive query* Q is of the form $S(x_1, \ldots, x_m) :- R_1(\boldsymbol{z}_1), \ldots, R_\ell(\boldsymbol{z}_\ell)$, where the *head* of the query, $\mathsf{head}_Q = S(x_1, \ldots, x_m)$, has a relation atom S not in $\mathcal{S}$ and its *body*, $\mathsf{body}_Q = \{R_1(\boldsymbol{z}_1), \ldots, R_\ell(\boldsymbol{z}_\ell)\}$, is a finite set of relation atoms over $\mathcal{S}$. In the following, all queries are assumed to be *safe*, that is, each variable in the head occurs at least once in some body atom. If V is a valuation that satisfies body_Q, we say that V *derives* fact $V(\mathsf{head}_Q)$. The *result* $Q(I)$ of query Q on instance I is the set of all derived facts.

2.2 Dependencies

A *tgd* σ is of the form $\mathcal{A}, \mathcal{C} \to \mathcal{A}'$, for sets $\mathcal{A}, \mathcal{A}'$ of relation atoms and a set $\mathcal{C}$ of comparison atoms with $\mathsf{var}(\mathcal{C}) \subseteq \mathsf{var}(\mathcal{A})$. Here, $\mathcal{A}'$ form its *head*, and $\mathcal{A}, \mathcal{C}$ its *body*, denoted $\mathsf{head}_\sigma = \mathcal{A}'$ and $\mathsf{body}_\sigma = \mathcal{A} \cup \mathcal{C}$, respectively. We refer to $\mathcal{A}$ by rbody_σ. The tgd is called *full* if $\mathsf{var}(\mathcal{A}') \subseteq \mathsf{var}(\mathcal{A})$. An instance I *satisfies* a tgd σ if, for every valuation V of body_σ that satisfies body_σ on I, there is an extension V' onto head_σ that satisfies head_σ on I.

An *egd* σ is of the form $\mathcal{A}, \mathcal{C} \rightarrow x = y$, for a set $\mathcal{A}$ of relation atoms and a set $\mathcal{C}$ of comparison atoms with $\mathsf{var}(\mathcal{C}) \cup \{x, y\} \subseteq \mathsf{var}(\mathcal{A})$. An instance I *satisfies* an egd σ if $V(x) = V(y)$ for every valuation V that satisfies body_σ on I, where body_σ is defined as for tgds.

Sets of dependencies are satisfied by an instance if each dependency in the set is satisfied. Satisfaction of a single dependency σ or a set Σ of dependencies by some instance I is denoted $I \models \sigma$ and $I \models \Sigma$, respectively.

A dependency τ is *implied* by a set Σ of dependencies, denoted $\Sigma \models \tau$, if $I \models \Sigma$ implies $I \models \tau$ for every instance I. For more precise statements, we can mention the actual domain in our notation. For example, we write $I \models_{\mathbb{N}} \tau$ if implication holds for all (finite) instances over $\mathbb{N}$.

We use the terms *dependencies* and *constraints* interchangeably.

2.3 Distributed Databases

We model a *network* of database servers as a finite set $\mathcal{N}$ of *nodes* and we denote its *size* by $|\mathcal{N}|$. We usually denote nodes by k and ℓ. A *distributed instance* $D = (G, \mathbf{I})$ consists of a *global instance* G and a family $\mathbf{I} = (I_k)_{k \in \mathcal{N}}$ of *local instances*, one for each node of $\mathcal{N}$, such that $\bigcup I_k \subseteq G$. We denote G by $\text{global}(D)$ and $(I_k)_{k \in \mathcal{N}}$ by $\text{local}(D)$.

We note that distributions allow redundant placement of facts, which is often desirable. Furthermore, it is not necessary to place all facts of the global instance on some node. A fact f is *skipped*[4] by D if $f \in \text{global}(D)$ but f does not occur in $\text{local}(D)$.

We write $f@_D k$ to denote that a fact f occurs at some node k, that is, $f \in I_k$. We drop D if it is clear from the context. We call $f@_D k$ a *distributed fact*. Sometimes we say that a set of facts *meet* in D when they all occur in the same local instance.

▶ **Example 2.** Consider a network $\mathcal{N} = \{1, 2\}$ of size 2 and a distributed instance $D = (G, \{I_1, I_2\})$ with $G = \{R(a,b), S(b), S(c), S(d)\}$, $I_1 = \{R(a,b), S(b)\}$ and $I_2 = \{S(b), S(c)\}$. Then fact $S(d)$ is skipped by D. The instance D can also be represented by the distributed facts $\{R(a,b), S(b), S(c), S(d), R(a,b)@1, S(b)@1, S(b)@2, S(c)@2\}$.

2.4 Parallel-correctness

Building on the computation model of massively parallel communication (MPC) [12], the naive evaluation of a conjunctive query Q over a distributed instance D evaluates Q separately for each local instance in $\text{local}(D)$. For $\text{local}(D) = (I_k)_{k \in \mathcal{N}}$, we write $Q_{\text{naive}}(D)$ for $\bigcup_{k \in \mathcal{N}} Q(I_k)$. Following [7], we say that a query Q is *parallel-correct* on D, if the naive evaluation produces the correct result, i.e., if $Q_{\text{naive}}(D) = Q(\text{global}(D))$.

3 Distribution constraints

We first introduce our framework for distribution constraints and afterwards give examples for its use.

[4] We note that allowing skipped facts makes the framework more flexible. They can be disallowed by simple distribution constraints, as discussed in Subsection 3.2.3.

3.1 Definition

Let nvar be an infinite set of *node variables* disjoint from dom and var. A *distributed atom* $A@\kappa$ consists of a relation atom A and a *node variable* κ in nvar. Recall that we refer to the variables of A as *data variables*. For a set of (distributed) atoms $\mathcal{A}$, we denote by $\mathsf{nvar}(\mathcal{A})$ the set of node variables occurring in atoms in $\mathcal{A}$. For a set $\mathcal{A}$ of relation atoms and a node variable κ, $\mathcal{A}@\kappa$ denotes the set $\{A@\kappa \mid A \in \mathcal{A}\}$.

Distribution tgds (dtgds) are defined just as tgds but they can additionally have distributed atoms in their body and their head. *Distribution egds (degds)* are defined just as egds but can have distributed atoms in their body. We do not allow node variables in comparison atoms (but we do allow them in the equality atom of a head in the case of degds). A degd $\mathcal{A}, \mathcal{C} \to A'$ is *node-identifying* if the equality atom A' refers to node variables only and *value-identifying* if, instead, A' refers to data variables only. We do *not* consider equality atoms where a node variable is identified with a data variable. We are particularly interested in *data-full* dtgds, for which the data variables in the head all occur in the body.

By $\mathcal{T}_{\text{all}}$ we denote the class of *all* dtgds and by $\mathcal{T}_{\text{df}}$ the class of data-full dtgds. By $\mathcal{E}_{\text{all}}$ we denote the class of *all* degds.

Satisfaction of dtgds and degds is defined in the obvious way with generalised valuations that may additionally map node variables to nodes. For a distributed atom $A' = A@\kappa$, we write $V(A') \in D$, if $V(A)@V(\kappa)$ is a distributed fact of D. For a relation atom A, we write $V(A) \in D$ if $V(A) \in \text{global}(D)$.

▶ **Example 3.** Given a schema $\mathcal{S}$ with binary relation symbols R and S, the following dtgd $\sigma = R(x,y), S(x,y) \to R(x,y)@\kappa, S(x,y)@\kappa$ is satisfied on a distributed instance D if, whenever global(D) contains two facts $R(a,b)$ and $S(a,b)$, for arbitrary data values $a, b \in \mathsf{dom}$, they meet in some local instance.

Below, in Section 3.2, we illustrate how distribution constraints can model global, local and global-to-local constraints.

▶ **Example 4.** The dtgd $E(x,y)@\kappa, E(y,z)@\kappa, E(z,x) \to E(z,x)@\kappa$ stipulates that every computing node has "complete" information w.r.t. open triangles on a binary relation E. That is, whenever a node contains two legs of a triangle, it also contains the closing leg if it exists in the global database.

Clearly, the differentiation between node and data variables in dtgds/degds can be seen as just syntactic sugar for standard relational schemas. The above restrictions (at most one node variable, at a fixed position, data-fullness) can then be seen as restrictions of classical constraints. In this sense, a dtgd like $R(x)@\kappa, S(x)@\mu \to T(x)@\kappa$ could be rewritten into a standard tgd of the form $R(\kappa,x), S(\mu,x) \to T(\kappa,x)$. The restriction to data-full dtgds thus translates to the restriction of existential quantification to these first attributes.

However, as the following example illustrates, our restriction to existential quantification of node variables does not translate into any of the restricted fragments with low complexity, which we are aware of.

▶ **Example 5.** Let Σ consist of a node-creating dtgd $R(x)@\kappa \to T(x)@\mu$ and a data-collecting dtgd $T(x)@\kappa, T(y)@\kappa, T(z)@\mu, T(w)@\mu \to U(x,y,z,w)@\kappa$. The corresponding set of standard tgds

$$\begin{aligned} R(\kappa,x) &\to T(\mu,x), \\ T(\kappa,x), T(\kappa,y), T(\mu,z), T(\mu,w) &\to U(\kappa,x,y,z,w), \end{aligned}$$

is neither sticky nor weakly guarded nor warded.[5] The set Σ has, however, bounded context (and its associated implication problem is shown to be in NP in Section 5).

Furthermore, the set consisting of $R(x,y)@\kappa \rightarrow S(x,y)@\mu$ and $S(x,x)@\kappa \rightarrow R(x,x)@\mu$ is not weakly acyclic but data-full with bounded context.

3.2 Examples of distribution constraints

In the following, we provide examples illustrating the versatility of distribution constraints. We begin with an examination of certain uses of distributed atoms. In principle, distributed atoms can be used in the body and in the head of constraints, referring to multiple node variables. Some more restricted uses seem particularly useful however.

We use the schema $\{\texttt{Emp}(\text{name}, \text{title}),\ \texttt{Sal}(\text{title}, \text{salary}),\ \texttt{Addr}(\text{name}, \text{address})\}$ as a running example for the remainder of this section.

3.2.1 Global dtgds and degds

We call distribution constraints *global* if they do not contain any distributed atom. These constraints refer to the global instance of a distributed database only – irrespective of the local databases. Formally, a dtgd (resp., degd) σ is a *global constraint* if σ is a tgd (resp., egd).

▶ **Example 6.** The following constraints are examples of a global dtgd and a global degd: $\texttt{Emp}(n,t) \rightarrow \texttt{Sal}(t,s)$, and $\texttt{Sal}(t,s), \texttt{Sal}(t,s') \rightarrow s = s'$. Together they specify that every employee has a unique salary.

3.2.2 Local dtgds and degds

Distribution constraints where every relation atom is a distributed atom and where all these atoms refer to the same node variable are called *local*. These constraints specify conditions that hold on *every* local instance, viewed on its own – irrespective of the global instance or other local instances.

▶ **Example 7.** The following is a local dtgd expressing that whenever a fact $\texttt{Emp}(a,b)$ occurs at node k there is a fact $\texttt{Sal}(b,c)$, for some element c in dom, that occurs at node k as well: $\texttt{Emp}(x,y)@\kappa \rightarrow \texttt{Sal}(y,z)@\kappa$. The following local value identifying degd expresses that, relative to each node, each employee (name) has a unique address. $\texttt{Addr}(x,y)@\kappa, \texttt{Addr}(x,y')@\kappa \rightarrow y = y'$.

3.2.3 Global-Local dtgds

Lastly, we call a dtgd *global-local* if none of its body atoms is distributed while all its heads atoms are distributed and refer to the same node variable.

▶ **Example 8.** The global-local constraint $\texttt{Emp}(x,y), \texttt{Sal}(y,z) \rightarrow \texttt{Emp}(x,y)@\kappa, \texttt{Sal}(y,z)@\kappa$ expresses that if there is an Emp-fact and a Sal-fact with the same title-attribute then these facts meet at some node. This means that the join condition between Emp and Sal induced by the schema is maintained in the horizontal decomposition of the global database.

[5] The set Σ' is not sticky because the marked variable μ occurs more than once in τ'. It is not (weakly) guarded because a single atom cannot contain both variables κ and μ that occur in affected positions of τ'. Finally, it is not warded because the dangerous variable κ appears in more than one atom in the body of τ'.

Global-local constraints can also express that the database has no skipped facts, i.e., facts $f \in \text{global}(D)$ with $f \notin \text{local}(D)$. To this end, for each relation symbol R a global-local constraint $R(x_1, \ldots, x_{\mathsf{ar}(R)}) \to R(x_1, \ldots, x_{\mathsf{ar}(R)})@\kappa$ can be added. Indeed, it is this ability of distributed constraints to disallow skipped facts that made us allow them in first place. For a schema $\mathcal{S}$, we denote by $\mathcal{U}(\mathcal{S})$ the set of all global-local constraints that express that there are no skipped facts.

By symmetry also local-global constraints can be defined. An example would be the dtgd $\mathtt{Emp}(x, y)@\kappa, \mathtt{Sal}(y, z)@\kappa \to \mathtt{Addr}(x, z')$ (even though for this particular schema, the constraint is rather contrived). Nevertheless, local-global constraints allow to state explicitly that every local fact is also a global fact, by stating $R(x_1, \ldots, x_m)@\kappa \to R(x_1, \ldots, x_m)$, for every relation R (with $m = \mathsf{ar}(R)$).

3.3 Applications of distribution constraints

We give some applications of distribution constraints like defining range, hash and co-partitionings and testing for parallel-correctness. In the full version, we illustrate how hypercube distributions can be incorporated and discuss query answering and multi-round query evaluation. The paper [35] further explores the use of distribution constraints to model distributed evaluation strategies for Datalog in the context of parallel-correctness and parallel-boundedness in the multi-round MPC model.

3.3.1 Range and hash partitioning

Distribution constraints can easily incorporate the commonly used range and hash partitionings (see for example [31, 37]). Example 1 already illustrates range partitionings.

The following two distribution constraints define a hash partitioning of the relation $\mathtt{Emp}(\text{name}, \text{dept})$ on the attribute department:

$$\mathtt{Emp}(n, d) \to \mathtt{Emp}(n, d)@\kappa$$
$$\mathtt{Emp}(n, d)@\kappa, \mathtt{Emp}(n', d) \to \mathtt{Emp}(n', d)@\kappa$$

The first rule enforces that every Emp-tuple occurs at a node while the second rule ensures that Emp-tuples within the same department are placed together. The above approach where hash functions are implicit should be contrasted with the modeling of Hypercube distributions, discussed in the full version, where hash functions are made explicit.

3.3.2 Co-partitioning

A popular way to avoid expensive remote join operations – already used in early parallel systems – is to co-partition tables on their join key [21, 25]. Generalizations of the latter technique where co-partitioning is determined by more complex join predicates have been shown to be effective in modern systems as well [38, 39, 41, 45].

Consider, for instance, the following (simplified) relations from the TPC-H schema [1]: $\mathtt{Lineitem}(\text{linekey}, \text{orderkey})$, $\mathtt{Orders}(\text{orderkey}, \text{custkey})$, and $\mathtt{Customer}(\text{custkey}, \text{cname})$.

Zamanian, Binnig, and Salama [45] exemplify the following co-partitioning scheme: `Lineitem` is hash-partitioned by linekey, `Orders` tuples are co-partitioned with `Lineitem` tuples with the same orderkey, and `Customer` tuples are co-partitioned with `Orders` tuples with the same custkey. As a consequence, the join $\mathtt{Lineitem} \bowtie \mathtt{Orders} \bowtie \mathtt{Customer}$ can be evaluated without expensive remote joins. We note that the work in [45] is by no means restricted to single-round or communication-free evaluation of queries. Knowledge of co-location of tuples is used to rewrite query plans and to determine those parts that can be evaluated without additional reshuffling. Partitionings are also not considered to be static but should adapt over time to changes in workload and the data (e.g., [32, 40]).

▶ **Example 9.** The following distribution constraints define the co-partitioning scheme mentioned above:

$$\texttt{Lineitem}(\ell, o) \to \texttt{Lineitem}(\ell, o)@\kappa \tag{1}$$

$$\texttt{Orders}(o, c) \to \texttt{Orders}(o, c)@\kappa \tag{2}$$

$$\texttt{Customer}(c, n) \to \texttt{Customer}(c, n)@\kappa \tag{3}$$

$$\texttt{Lineitem}(\ell, o), \texttt{Lineitem}(\ell, o')@\kappa \to \texttt{Lineitem}(\ell, o)@\kappa \tag{4}$$

$$\texttt{Lineitem}(\ell, o)@\kappa, \texttt{Orders}(o, c) \to \texttt{Orders}(o, c)@\kappa \tag{5}$$

$$\texttt{Orders}(o, c)@\kappa, \texttt{Customer}(c, n) \to \texttt{Customer}(c, n)@\kappa \tag{6}$$

Basically, Constraint (1) expresses that every Lineitem fact in the global database occurs at some node; similarly for Constraints (2) and (3). Constraint (4) then expresses that Lineitem facts are hashed on the first attribute: for every item ℓ with order o' stored on some server, every other order of that item is stored there too. Constraint (5) expresses that $\texttt{Orders}(o, c)$ facts are co-located with $\texttt{Lineitem}(\ell, o)$ facts, while Constraint (6) expresses that $\texttt{Customer}(c, n)$ facts are co-located with $\texttt{Orders}(o, c)$ facts. All together the distribution constraints imply that the join condition between the three relations is maintained in the horizontal decomposition of the global database.

3.3.3 Hierarchical partitioning schemes

Recent database systems like Google's F1 [39, 41] use hierarchical partitioning schemes to provide performance while ensuring consistency under updates. Hierarchical partitioning is a variant of the *co-partitioning* approach [42], introduced as *predicate-based reference partitioning* [45]. This approach allows to formulate a hashing condition for a relation S, given that a relation R is already distributed, in the following style: first, every S-fact has to be distributed, and second, if an S-fact joins with an R-fact on a predefined set of attributes, then the S-fact is distributed to every node where such an R-fact exists.

This is easily modeled by the following dtgds:

$$S(\boldsymbol{z}) \to S(\boldsymbol{z})@\kappa,$$
$$R(\boldsymbol{y})@\kappa, S(\boldsymbol{z}) \to S(\boldsymbol{z})@\kappa,$$

where variables $\boldsymbol{y}$ and $\boldsymbol{z}$ share some common variables $x_1, \ldots, x_n$ representing the join predicate. Notice that Example 9 follows this scheme. Another example, illustrating Google's AdWord scenario, is given in the full version.

3.3.4 Parallel-Correctness

We show that parallel-correctness of a conjunctive query can be captured by a dtgd but not always by a data-full one.

▶ **Example 10.** Let $Q = H(n, s) \leftarrow \texttt{Emp}(n, t), \texttt{Sal}(t, s)$ be a conjunctive query. Then, Q is parallel-correct on a distributed instance D if every fact from $Q(\text{global}(D))$ is derived at some node (due to the monotonicity of CQs, we do not need to check the converse statement). This can be expressed by the dtgd $\texttt{Emp}(x, y), \texttt{Sal}(y, z) \to \texttt{Emp}(x, y')@\kappa, \texttt{Sal}(y', z)@\kappa$. We note that in this dtgd κ and y' occur only in the head. So, this dtgd is *not* data-full.

4 Reasoning

We consider the implication problem for distribution constraints in Section 4.1, and adapt the chase to degds and data-full dtgds in Section 4.2, as a means to solve it.

4.1 The implication problem

We stress that the implied dependency τ in the definition below, is not required to belong to the class $\mathcal{C}$. That is, τ can be an arbitrary distribution constraint.

▶ **Definition 11.** *The* implication problem $\textsc{Imp}(\mathcal{C}, d)$, *parameterised by a class $\mathcal{C}$ of dependencies and a domain d asks, for a finite set Σ from $\mathcal{C}$ and a single distribution constraint τ, whether $\Sigma \models_d \tau$. Possible choices for d are $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. If the choice of d does not matter or is clear from the context, we also write $\textsc{Imp}(\mathcal{C})$. For each $\alpha \geq 1$, we denote by $\textsc{Imp}_\alpha(\mathcal{C}, d)$ the restriction of $\textsc{Imp}(\mathcal{C}, d)$ to inputs (Σ, τ) in which the arity of each relation symbol (with respect to data) is at most α.*

Since every tgd is a dtgd and the implication problem for tgds without comparison atoms is undecidable, we instantly get the following.

▶ **Observation 12** ([13, 19])**.** *$\textsc{Imp}(\mathcal{T}_{all})$ is undecidable.*

To facilitate automatic reasoning, it thus makes sense to consider restricted kinds of constraints. An immediate observation is that most of the examples in Section 3 only use data-full distribution constraints. In the remainder of this paper, we therefore restrict our attention to this class.

▶ Remark 13. A full tgd $\sigma = \mathcal{A}, \mathcal{C} \to \{A'_1, \ldots, A'_p\}$ can be transformed into an equivalent set of tgds $\{\mathcal{A}, \mathcal{C} \to A'_1, \ldots, \mathcal{A}, \mathcal{C} \to A'_p\}$ with a singleton head.[6]In particular, this applies to dtgds *without existential quantification*. Similarly, data-full dtgds *with existentially quantified node variables* can be decomposed into data-full dtgds with at most one node variable in their head. We thus assume w.l.o.g. in *upper bound proofs* that all dtgds in Σ are decomposed in this fashion.

Since data-full dtgds have at most one node variable in their head, we can distinguish three kinds of data-full dtgds:

- *node-creating dtgds* like $R(x, y) \to R(x, y)@\kappa$, in which the one node variable in their head is existentially quantified;
- *data-collecting dtgds* like $S(x)@\kappa, T(x)@\lambda \to T(x)@\kappa$, which are dtgds that have one distributed head atom without existential quantification (they collect facts in a local node); and,
- *global dtgds* like $R(x, y) \to U(x)$ or $R(x, y)@\kappa, T(x)@\kappa \to S(y)$ that have one head atom without node variable (they contribute global facts).[7]

For brevity, we sometimes refer to *generating* and *collecting* dtgds.

We call the unique node variable that occurs in the head of a node-creating or data-collecting dtgd σ the *head variable* of σ.

[6] This observation is used also used the context of normalised schema mappings [24, 28].

[7] The term *global dtgd* thus represents a superset of global and local-global dtgds from Section 3.2.

4.2 The chase for distribution constraints

The classical way for deciding implication of tgds and egds builds on the *chase* procedure. In the following, we adapt the chase from [23] for distribution constraints from $\mathcal{T}_{\mathrm{df}}$ and $\mathcal{E}_{\mathrm{all}}$.

▶ **Definition 14** (chase step). *A dtgd σ is* applicable *to a distributed instance D with a valuation W if W satisfies* body_σ *on D and there exists no valuation W' for σ identical to W on* body_σ *such that $W'(\mathsf{head}_\sigma) \subseteq D$. Furthermore, if σ is node-creating then $W(\kappa)$ must be a node k not occurring in D, where κ is the head variable of σ. The* result $\mathsf{chase}(c, D)$ *of applying $c = (\sigma, W)$ on D is the distributed instance $D' = D \cup W(\mathsf{head}_\sigma)$ and we write $D \xrightarrow{c} D'$.*

For instance, if a distributed database D consists of facts $R(a)@1$ and $S(a, b)@2$, then dependency $\sigma = R(x)@\kappa, S(x, y)@\lambda \to R(x)@\mu, S(x, y)@\mu$ is applicable to D, as witnessed by the valuation W where $W(x, y) = (a, b)$ and $W(\kappa, \lambda, \mu) = (1, 2, 3)$. Application leads to $D' = D \cup \{R(a)@3, S(a, b)@3\}$.

If σ is node-creating, we say that the chase step *generates* the new node $W(\kappa)$ with an initial set $W(\mathsf{head}_\sigma)$ of facts. If σ is data-collecting, we say that the chase step *collects* the facts from $W(\mathsf{head}_\sigma)$ in node $W(\kappa)$. If σ is global, we say that the chase step *targets* the global database. The chase step *contributes* the set $W(\mathsf{head}_\sigma)$ of global facts.

▶ **Definition 15** (chase sequence). *Let Σ be a set of distribution constraints and D a distributed instance. A* chase sequence *for D with Σ is a sequence $\mathbf{D} = D_0, D_1, \ldots$ of distributed instances with $D_0 = D$ and $D_i \xrightarrow{(\sigma_i, W_i)} D_{i+1}$, for every $i \geq 0$ and some $\sigma_i \in \Sigma$ and valuation W_i. We write* $\mathsf{chase}(\mathbf{D})$ *for the final instance of $\mathbf{D}$, if $\mathbf{D}$ is finite.*

A chase sequence $\mathbf{D}$ fails*, if there is a degd $\sigma \in \Sigma$ with a head $t = t'$ and a valuation W, such that W satisfies* body_σ *on* $\mathsf{chase}(\mathbf{D})$ *and $W(t) \neq W(t')$. It is* successful*, if it is finite, does not fail and* $\mathsf{chase}(\mathbf{D})$ *has no applicable chase step.*

The following easy observation is crucial for our results.

▶ **Proposition 16.** *For each distributed database D and each set Σ of constraints from $\mathcal{T}_{df}$ and $\mathcal{E}_{all}$, there are no infinite chase sequences for D.*

For classical tgds and egds, to test $\Sigma \models \tau$, the chase is basically applied to a "canonical database" $V(\mathsf{rbody}_\tau)$, for some one-one valuation V. Due to comparison atoms, this does not suffice in our setting. Instead, we consider a set of canonical databases, which allows for all possible linear orders on the variables of body_τ. It depends on the general domain d which we allow to be one of $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$. More precisely, it is defined over a set $\mathsf{dom}(\Sigma, \tau, d)$ of data values that contains all constants of Σ and τ and, between each pair of successive constants all intermediate values from d or as many intermediate values as there are variables in body_τ. The set of canonical databases then consists of all databases of the form $V(\mathsf{rbody}_\tau)$ for valuations whose range is in $\mathsf{dom}(\Sigma, \tau, d)$.

Towards a formal definition, $c_1 < \cdots < c_\ell$ denote the constants in $\Sigma \cup \{\tau\}$ and let m be the number of data variables in body_τ. If $d = \mathbb{Q}$ then $\mathsf{dom}(\Sigma, \tau, d)$ consists of $c_1, \ldots, c_\ell$, all values $c_1 - m, \ldots, c_1 - 1$, all values $c_\ell + 1, \ldots, c_\ell + m$ and all values of the form $c_i + \frac{j}{m+1}(c_{i+1} - c_i)$, for $i \in \{1, \ldots, \ell - 1\}$ and $j \in \{1, \ldots, m\}$. If $d = \mathbb{Z}$, it consists of $c_1, \ldots, c_\ell$, all values $c_1 - m, \ldots, c_1 - 1$, all values $c_\ell + 1, \ldots, c_\ell + m$ and all values of the form $c_i + j$, for $i \in \{1, \ldots, \ell - 1\}$ and $j \in \{1, \ldots, m\}$ with $c_i + j < c_{i+1}$. If $d = \mathbb{N}$, it is defined as for $d = \mathbb{Z}$ with the additional constraint that elements of the form $c_1 - j$ must be non-negative.

By $\mathcal{D}(\Sigma, \tau, d)$ we denote the set of all distributed databases $V(\mathsf{rbody}_\tau)$, for which V maps data variables to values in $\mathsf{dom}(\Sigma, \tau, d)$ and node variables one-one to an initial segment of (a disjoint copy of) the natural numbers.

The following result shows that, to decide implication, it suffices to apply the chase to all databases in $\mathcal{D}(\Sigma, \tau, d)$.

▶ **Proposition 17.** *Let $\Sigma \cup \{\tau\}$ be a set distribution constraints from $\mathcal{T}_{df}$ and $\mathcal{E}_{all}$ and let d be one of $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$. Then the following statements are equivalent.*

(1) $\Sigma \models_d \tau$.

(2) *For every database $D = V(\mathsf{rbody}_\tau)$ in $\mathcal{D}(\Sigma, \tau, d)$ and every successful chase sequence $\mathbf{D}$ for D with Σ, there is an extension V' of V that satisfies head_τ on $\mathsf{chase}(\mathbf{D})$.*

(3) *For every database $D = V(\mathsf{rbody}_\tau)$ in $\mathcal{D}(\Sigma, \tau, d)$ there exists a chase sequence $\mathbf{D}$ for D with Σ, that fails or for which there is an extension V' of V that satisfies head_τ on $\mathsf{chase}(\mathbf{D})$.*

The straightforward proof is given in the full version.

5 Complexity

In this section, we study the complexity of the implication problem for data-full dtgds (and arbitrary degds). In general, this problem turns out to be in EXPTIME, in fact as EXPTIME-complete. We then study restrictions of dtgds and degds that lower the complexity of the implication problem. In fact, we identify fragments whose implication problems are Π_2^p-complete (NP-complete without comparison atoms) or PSPACE-complete. To wrap up the picture, we finally identify fragments that already yield EXPTIME-hardness.

For most practical cases, the relevant complexity is Π_2^p or even NP: the former is the case, e.g., if the database schema (or at least its arity) is fixed and if the number of atoms (or at least the number of variables) is bounded by some a-priori constant. The latter is the case if, additionally, there are no comparison atoms. In particular, the "natural" generalisations of our examples have at most Π_2^p (or NP) complexity.

The first result of this section states that $\textsc{Imp}(\mathcal{T}_{\text{df}})$ is EXPTIME-complete. The upper bound is very simple but shows that the problem is not harder than implication of full (non-distributed) tgds. On the other hand, in the distributed setting, EXPTIME-hardness already holds for fixed schemas with small arity, whereas this problem is easily seen to be in Π_2^p for (non-distributed) full dependencies.

▶ **Theorem 18.** *$\textsc{Imp}(\mathcal{T}_{df} \cup \mathcal{E}_{all})$ is EXPTIME-complete. The lower bound already holds for a fixed schema of arity 2.*

Proof sketch. The lower bound follows from Theorem 26, which is shown in Subsection 5.3 and offers a collection of types of distributed constraints that make the implication problem EXPTIME-hard.

The upper bound uses Proposition 17. Since $\mathsf{dom}(\Sigma, \tau, d)$ has polynomial size in $|\Sigma| + |\tau|$, the set $\mathcal{D}(\Sigma, \tau, d)$ contains only exponentially many databases, from which the chase needs to start. Furthermore, each constraint in Σ can be applied at most an exponential number of times, since there are only exponentially many different valuations, and each of them can fire at most once. Therefore, each chase sequence is of at most exponential length. ◀

Intuitively, EXPTIME-hardness for the class $\mathcal{T}_{\text{df}}$ of data-full dtgds (and even without degds) follows from the need to keep track of an exponential number of nodes, as can be seen from the proof of the lower bound of Theorem 26.

In the following two subsections, we turn to restricted classes with lower complexity[8] for the implication problem. The fragments that we study are not motivated from practical considerations (since there we already have "low" complexity). They were rather obtained by considering syntactic properties under which the chase behaves better than in general.

First of all, these fragments require a fixed bound on the arity of relations. Furthermore, they bound the amount of data that is associated with a single node in a dtgd, in various ways. To state this more precisely, we use the following notions.

▶ **Definition 19** (bounded context). *The* context $\mathsf{cont}_\kappa(\mathcal{A})$ *of a node variable κ in a set $\mathcal{A}$ of atoms is the set of (data) variables occurring in atoms referring to κ. The context* $\mathsf{cont}_\kappa(\sigma)$ *of κ in a dtgd σ is* $\mathsf{cont}_\kappa(\mathsf{rbody}_\sigma \cup \mathsf{head}_\sigma)$. *The context* $\mathsf{cont}_\kappa(\sigma)$ *of κ in a degd σ is* $\mathsf{cont}_\kappa(\mathsf{rbody}_\sigma)$.

A node variable κ has b-bounded context *in σ if* $|\mathsf{cont}_\kappa(\sigma)| \leq b$. *It has* b-bounded body context *if* $|\mathsf{cont}_\kappa(\mathsf{rbody}_\sigma)| \leq b$.

For instance, in the two following constraints,

- dtgd $\sigma_1 = R(x, y, z), S(y)@\kappa \rightarrow T(x)@\kappa$ and
- degd $\sigma_2 = S(x)@\kappa, S(y)@\kappa, R(x, y, z)@\lambda \rightarrow \kappa = \lambda$,

node variable κ has context $\{x, y\}$ and thus 2-bounded context. The body context of κ in σ_1 is even 1-bounded. Note that, in σ_2, node variable λ has 3-bounded body context and that, since there is no other node variable, the body context of this constraint is bounded by $3 = \max\{2, 3\}$ in general.

We sometimes simply speak of *bounded context* if b is clear from the, well, context.

5.1 Classes with Π_2^p-reasoning

In this subsection, we consider two fragments which allow reasoning in Π_2^p in general, and in NP, if there are no comparison atoms.

The first fragment requires only one restriction (besides the usual arity restriction). The *bounded generation* fragment $\mathcal{T}_{\text{bg}}^b$ allows all global and data-collecting dtgds but only node-creating dtgds, in which the head variable has b-bounded context. We refer to the latter as node-creating dtgds of Type (G1), cf. Table 1.

▶ **Theorem 20.** *For fixed $\alpha \geq 1$ and $b \geq 1$, problem* $\text{IMP}_\alpha(\mathcal{T}_{bg}^b \cup \mathcal{E}_{all})$ *is*

1. Π_2^p*-complete in general and*
2. NP*-complete, if restricted to inputs without comparison atoms.*

Proof idea. The lower bounds follow by reductions from the containment problem for conjunctive queries (with or without comparisons) [20, 44]. For two queries $\mathcal{Q}$ and $\mathcal{Q}'$ of the respective classes, query $\mathcal{Q}$ is contained in $\mathcal{Q}'$ if and only if $\{\sigma\} \models \tau$, where $\sigma = \mathsf{body}_{\mathcal{Q}'} \rightarrow \mathsf{head}_{\mathcal{Q}'}$ and $\tau = \mathsf{body}_{\mathcal{Q}} \rightarrow \mathsf{head}_{\mathcal{Q}}$ are considered as global data-collecting dtgds (with or without comparisons).

The proofs of the upper bounds use Condition (3) from Proposition 17 and rely on the fact that, in each chase sequence, thanks to the (G1)-restriction only a polynomial number of nodes is generated and thanks to the arity restriction, each can carry only a polynomial number of facts. The Π_2^p upper bound can be almost directly inferred from the quantifier structure of Condition (3). The NP upper bound follows since, essentially, only one initial database needs to be considered. Below, we provide the details.

[8] This statement holds under the common assumption that Π_2^p and PSPACE are smaller than EXPTIME.

We begin by showing that all possible chase sequences have polynomial length. Let Σ be a set of dependencies in $\mathcal{T}^b_{\text{bg}} \cup \mathcal{E}_{\text{all}}$ and τ be a distribution constraint. We recall that the chase applies a node-creating chase step with a dtgd σ and a valuation W only if no node with the facts from $W(\mathsf{head}_\sigma)$ exists. However, for each $\sigma \in \Sigma$, the number of variables in head_σ occurring in atoms related to κ is at most b and thus the number of different valuations of head_σ (with the initial values derived from D_τ) is at most $|\mathsf{dom}(\Sigma, \tau, d)|^b$, and thus polynomial. Therefore, the number of chase steps using a σ of Type (G1) is polynomially bounded. In particular, the chase generates only a polynomial number of nodes. Since data-collecting dtgds have only one atom in their head and α is a bound on the arity of atoms, there can only be a polynomial number of chase steps using data-collecting dtgds, for each node. Similarly, there can only be a polynomial number of chase steps using global dtgds. Altogether there can be only a polynomial number of chase steps in each chase sequence. As mentioned before, the Π^p_2 upper bound follows from Condition (3) in Proposition 17. Universal quantification is over all databases in $\mathcal{D}(\Sigma, \tau, d)$, the chase sequence $\mathbf{D}$ is existentially quantified and that it fails or there exists an appropriate extension can be verified by further existential quantification.

If there are no comparison atoms in Σ and τ it suffices to start the chase from *one* canonical database of the form $V(\mathsf{body}_\tau)$, for some one-one valuation V that does not map any variables of body_τ to constants of body_τ. However, the chase needs to be defined in a slightly different fashion: if a degd with a head of the form $t = t'$ is applicable via a valuation W then in the result $W(t)$ and $W(t')$ are identified, unless they are different constants from body_τ. If the latter is the case, the chase fails. For this version of the chase, Proposition 17 holds as well.

The NP upper bound then follows, since only one initial database needs to be used and only one chase sequence of polynomial length needs to be guessed. ◀

The other class of dtgds considered in this subsection allows node-creating dtgds with head variables with unbounded context. The simple argument of the proof of Theorem 20 therefore does not work anymore. However, it turns out that there are simple (and still generous) restrictions that guarantee a Π^p_2 (NP) upper bound for the implication problem. To this end, we define the *bounded context fragment* $\mathcal{T}^b_{\text{bc}}$ of dtgds as follows (cf. Table 1).

▶ **Definition 21** (bounded context dtgds). *A node-creating dtgd σ is in $\mathcal{T}^b_{bc}$, if*

(G1) *its head variable has b-bounded context, or*

(G2) *all node variables in its body have b-bounded context.*

A data-collecting dtgd σ is in $\mathcal{T}^b_{bc}$, if

(C1) *its head variable has b-bounded body context, or*

(C2) *all other node variables have b-bounded context.*

For instance, all global dtgds and degds are in $\mathcal{T}^b_{bc}$.

The *bounded context fragment* $\mathcal{E}^b_{\text{bc}}$ of degds is defined similarly.

▶ **Definition 22** (bounded context degds). *A degd σ with only data variables in its head is in $\mathcal{E}^b_{bc}$. A degd $\sigma = \mathcal{A} \to \kappa = \mu$ is in $\mathcal{E}^b_{bc}$, if*

(E1) *κ and μ have b-bounded context, or*

(E2) *μ and all node variables that do not occur in the head have b-bounded context.*

The degds of $\mathcal{E}^b_{\text{bc}}$ are illustrated in Table 1. In degds of Type (E2), we call κ (but not μ) the *head variable.*

We can now state the second result of this subsection.

▶ **Theorem 23.** *For fixed $\alpha \geq 1$ and $b \geq 1$, problem* $\textsc{Imp}_\alpha(\mathcal{T}^b_{\mathrm{bc}} \cup \mathcal{E}^b_{\mathrm{bc}})$ *is*

1. Π^p_2*-complete in general and*
2. *NP-complete, if restricted to inputs without comparison atoms.*

Proof idea. For the upper bounds, we show that any chase sequence for $\mathcal{T}^b_{\mathrm{bc}} \cup \mathcal{E}^b_{\mathrm{bc}}$ can be *normalised* such that only a polynomial number of *witness nodes* are needed to trigger any chase steps. Since every node has only a polynomial number of facts, this implies that it suffices to consider chase sequences of polynomial length. The remaining arguments are then as for Theorem 20. The lower bounds follow by the same reduction as in Theorem 20. ◀

5.2 Classes with PSPACE-reasoning

In this subsection, we consider a fragment of distribution constraints that does not guarantee polynomial-length chase sequences but, intuitively, sequences of polynomial "width". Consequently, their implication problem turns out as PSPACE-complete.

The fragment $\mathcal{T}^b_{\mathrm{wbc}}$ is defined as follows (cf. Table 1).

▶ **Definition 24** (weakly bounded distribution tgds)**.** *Let $b \geq 1$. A dtgd σ is in the class $\mathcal{T}^b_{wbc}$ of* weakly bounded distribution tgds *if it is in $\mathcal{T}^b_{\mathrm{bc}}$ or it obeys the following restriction:*

(G3) σ is node-creating and exactly one of its node variables does not *have b-bounded body context.*

▶ **Theorem 25.**

1. $\textsc{Imp}_\alpha(\mathcal{T}^b_{wbc} \cup \mathcal{E}^b_{\mathrm{bc}})$ *is in PSPACE, for every $\alpha \geq 1$ and $b \geq 1$.*
2. $\textsc{Imp}_\alpha(\mathcal{T}^b_{wbc})$ *is PSPACE-hard for $\alpha \geq 1$ and $b \geq 0$. This lower bound even holds without comparison atoms.*

Proof idea. The lower bound (2) is shown similarly as PSPACE-hardness of the implication problem for inclusion dependencies over schemas of *unbounded* arity [4, 18]. In a nutshell, in this reduction each node carries *one tuple*, encoded with unary relations.

For the upper bound, unlike for $\mathcal{T}_{\mathrm{bc}}$, we do not have a polynomial length bound for chase sequences for $\mathcal{T}_{\mathrm{wbc}}$. In fact, it might be the case that a chase sequence generates an exponential number of nodes. However, we can still use a polynomially bounded set Z of witness nodes for the bounded node variables of (G3) dtgds and for all other constraints. They do not account for the unbounded node variables in (G3) constraints, but we show that those only need to occur in linear succession. The basic idea of the algorithm is to guess Z (and the facts on nodes from Z) and to verify in polynomial space, for each node in Z, that it is produced by a chase sequence. These verifying computations all assume the same set Z. We use a kind of timestamps to avoid cyclic reasoning. The details of this proof are given in the full version. ◀

5.3 Classes with EXPTIME-hard reasoning

In this subsection, we turn to combinations of constraints that yield an EXPTIME-hard implication problem. In particular, we complete the proof of Theorem 18. To this end, we consider the following additional types of constraints:

(G4) node-creating dtgds with two unbounded node variables;
(C3) data-collecting dtgds with two unbounded node variables;
(E3) degds with two unbounded node variables; and,
(E4) degds with three unbounded node variables.

▶ **Theorem 26.** *$\textsc{Imp}(\mathcal{T}_{df})$ is EXPTIME-hard. This statement holds already without comparison atoms and with only the following combinations of constraint types allowed:*
(a) *Node-creating dtgds of Type (G2) and data-collecting dtgds of Type (C3);*
(b) *Node-creating dtgds of Type (G2) and (G4);*
(c) *Node-creating dtgds of Type (G2) and degds of Type (E4);*
(d) *Node-creating dtgds of Types (G2) and (G3), and degds of Type (E3).*
In all cases, schemas with (at most) binary relations suffice.

The four EXPTIME-hard fragments are illustrated in Table 1. The reductions use an alternating Turing machine with *linearly* bounded space.

5.4 Parallel-correctness revisited

We lift parallel-correctness to the setting of distribution constraints. In particular, we say that a query Q is *parallel-correct w.r.t. a set of distribution constraints* Σ if Q is parallel-correct on every database that satisfies Σ.

As parallel-correctness of a conjunctive query can be expressed as a dtgd, the results of the present section lead to the following:

▶ **Corollary 27.** *For a CQ Q and a set of distribution constraints Σ, the complexity of deciding parallel-correctness of Q w.r.t. Σ is in EXPTIME. Furthermore, it is in Π_2^p (or NP, without comparsion atoms) and PSPACE if $\Sigma \subseteq \mathcal{T}_{bc}^b \cup \mathcal{E}_{bc}^b$ and $\Sigma \subseteq \mathcal{T}_{wbc}^b \cup \mathcal{E}_{bc}^b$, respectively, for a fixed b and a fixed bound α on the maximal arity of relation symbols.*

6 Conclusion

In this work, we introduced a novel declarative framework based on classical tgds and egds with comparison atoms to specify and reason about classes of data distributions. We illustrated our framework by various examples and performed an initial study of the complexity of the implication problem. As an application, we derived bounds (in Corollary 27) for the complexity of parallel-correctness of conjunctive queries.

Of course, there are many immediate general directions for extending the line of work started in this paper. For instance, one could study the implication problem for more expressive distribution constraints than data-full ones. There is a plethora of work on fragments of dependencies for improving the complexity of decision problems (e.g., [10, 14, 16, 36]). It could be investigated if any of these or others lead to a decidable implication problem. Another direction for future work is to study parallel-correctness w.r.t. distribution constraints for more expressive query languages than conjunctive queries. Some possibilities are unions of conjunctive queries [7], conjunctive queries with negation [26] or Datalog [29].

Example 9 and Section 3.3.3 illustrate how co-partitioning schemes can be translated into distribution constraints. It would be interesting to investigate the converse direction. That is, by design, distribution constraints specify in a declaratively way which properties a horizontal partitioning should satisfy. They do not provide a direct operational way to compute an actual partitioning. A natural question is to find an optimal partitioning satisfying a given set of distribution constraints.

Section 3 mentions a translation of distribution constraints to classical tgds and egds by increasing the arity of relations by one to take the node variables into account. It would be interesting to see whether the resulting fragment of dependencies is worthwhile to study it on its own in the classical setting.

Table 1 Illustration of restricted classes of dtgds and degds. Node variables that have to be bounded are shaded, others may be unbounded. The columns indicate the complexity of (some) combinations of fragments.

		Π_2^p (NP)		PSPACE	EXPTIME			
(G1)	$\boxed{\lambda_1} \cdots \boxed{\lambda_r} \rightarrow \colorbox{lightgray}{κ}$	✓	✓	✓				
(G2)	$\colorbox{lightgray}{$\lambda_1$} \cdots \colorbox{lightgray}{λ_r} \rightarrow \boxed{\kappa}$		✓	✓	✓		✓	✓
(G3)	$\colorbox{lightgray}{$\lambda_1$} \cdots \colorbox{lightgray}{λ_r}\boxed{\mu} \rightarrow \boxed{\kappa}$			✓				✓
(G4)	$\boxed{\lambda}\boxed{\mu} \rightarrow \boxed{\kappa}$					✓		
Unrestricted data-collecting dtgds		✓						
(C1)	$\colorbox{lightgray}{$\kappa$}\boxed{\lambda_1} \cdots \boxed{\lambda_r} \rightarrow \boxed{\kappa}$		✓	✓				
(C2)	$\boxed{\kappa}\colorbox{lightgray}{λ_1} \cdots \colorbox{lightgray}{λ_r} \rightarrow \boxed{\kappa}$		✓	✓				
(C3)	$\boxed{\kappa}\boxed{\lambda} \rightarrow \boxed{\kappa}$				✓			
Unrestricted degds		✓						
(E1)	$\colorbox{lightgray}{$\kappa$}\colorbox{lightgray}{$\mu$}\boxed{\lambda_1} \cdots \boxed{\lambda_r} \rightarrow \kappa = \mu$		✓	✓				
(E2)	$\boxed{\kappa}\colorbox{lightgray}{μ}\colorbox{lightgray}{λ_1} \cdots \colorbox{lightgray}{λ_r} \rightarrow \kappa = \mu$		✓	✓				
(E3)	$\boxed{\kappa}\boxed{\lambda} \rightarrow \kappa = \lambda$							✓
(E4)	$\boxed{\kappa}\boxed{\lambda}\boxed{\mu} \rightarrow \kappa = \lambda$						✓	
	Theorem	20	23	25	26(a)	26(b)	26(c)	26(d)

The main technical challenge left open from this work is whether the EXPTIME-hardness result in Theorem 26(c) can be extended to rules of Type (E4) that contain two rather than three unbounded node variables.

References

1 TPC-H. URL: `http://www.tpc.org/tpch/`.

2 Serge Abiteboul, Émilien Antoine, and Julia Stoyanovich. The Webdamlog System Managing Distributed Knowledge on the Web. *CoRR*, abs/1304.4187, 2013. `arXiv:1304.4187`.

3 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Émilien Antoine. A rule-based language for web data management. In *PODS*, pages 293–304, 2011.

4 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

5 Foto N. Afrati, Chen Li, and Vassia Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT*, pages 487–498, 2008.

6 Peter Alvaro, William R. Marczak, Neil Conway, Joseph M. Hellerstein, David Maier, and Russell Sears. Dedalus: Datalog in Time and Space. In *Datalog Reloaded*, pages 262–281, 2010.

7 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Transferability for Conjunctive Queries. *J. ACM*, 64(5):36:1–36:38, 2017. `doi:10.1145/3106412`.

8 Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014. URL: `http://www.cambridge.org/9781107016163`.

9 Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Ontology-Mediated Query Answering over Temporal Data: A Survey (Invited Talk). In *TIME*, pages 1:1–1:37, 2017.

10 Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *IJCAI*, pages 712–717, 2011.

11 Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Constraint-Generating Dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999. doi:10.1006/jcss.1999.1632.

12 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *J. ACM*, 64(6):40:1–40:58, 2017.

13 Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In *ICALP*, pages 73–85, 1981.

14 Michael Benedikt. How Can Reasoners Simplify Database Querying (And Why Haven't They Done It Yet)? In *PODS*, pages 1–15, 2018.

15 Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The Space-Efficient Core of Vadalog. *CoRR*, abs/1809.05951, 2018. arXiv:1809.05951.

16 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013. doi:10.1613/jair.3873.

17 Andrea Calì, Georg Gottlob, and Andreas Pieris. Advanced Processing for Ontological Queries. *PVLDB*, 3(1):554–565, 2010. doi:10.14778/1920841.1920912.

18 Marco A. Casanova, Ronald Fagin, and Christos H. Papadimitriou. Inclusion Dependencies and Their Interaction with Functional Dependencies. *J. Comput. Syst. Sci.*, 28(1):29–59, 1984.

19 Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. Embedded Implicational Dependencies and their Inference Problem. In *STOC*, pages 342–354, 1981.

20 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

21 D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. . Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.

22 Deming Dou and Stéphane Coulondre. A sound and complete chase procedure for constrained tuple-generating dependencies. *J. Intell. Inf. Syst.*, 40(1):63–84, 2013. doi:10.1007/s10844-012-0216-5.

23 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

24 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In Catriel Beeri and Alin Deutsch, editors, *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14-16, 2004, Paris, France*, pages 83–94. ACM, 2004. doi:10.1145/1055558.1055572.

25 Shinya Fushimi, Masaru Kitsuregawa, and Hidehiko Tanaka. An Overview of The System Software of A Parallel Relational Database Machine GRACE. In *VLDB*, pages 209–219, 1986.

26 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Containment for Conjunctive Queries with Union and Negation. In *ICDT*, 2016.

27 Gaetano Geck, Frank Neven, and Thomas Schwentick. Distribution constraints: The chase for distributed data, 2020. arXiv:2003.00965.

28 Georg Gottlob, Reinhard Pichler, and Vadim Savenkov. Normalization and optimization of schema mappings. *VLDB J.*, 20(2):277–302, 2011. doi:10.1007/s00778-011-0226-x.

29 Bas Ketsman, Aws Albarghouthi, and Paraschos Koutris. Distribution Policies for Datalog. In *ICDT*, pages 17:1–17:22, 2018.

30 Bas Ketsman and Dan Suciu. A Worst-Case Optimal Multi-Round Algorithm for Parallel Computation of Conjunctive Queries. In *PODS*, pages 417–428, 2017.

31 Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly, 2016. URL: `http://shop.oreilly.com/product/0636920032175.do`.

32 Yi Lu, Anil Shanbhag, Alekh Jindal, and Samuel Madden. AdaptDB: Adaptive Partitioning for Distributed Joins. *PVLDB*, 10(5):589–600, 2017.

33 Michael J. Maher and Divesh Srivastava. Chasing Constrained Tuple-Generating Dependencies. In *PODS*, pages 128–138, 1996.

34 Vera Zaychik Moffitt, Julia Stoyanovich, Serge Abiteboul, and Gerome Miklau. Collaborative Access Control in WebdamLog. In *SIGMOD*, pages 197–211, 2015.

35 Frank Neven, Thomas Schwentick, Christopher Spinrath, and Brecht Vandevoort. Parallel-Correctness and Parallel-Boundedness for Datalog Programs. In *ICDT*, pages 14:1–14:19, 2019.

36 Adrian Constantin Onet. The chase procedure and its applications. PhD Thesis, June 2012. URL: `https://spectrum.library.concordia.ca/974476/`.

37 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition.* Springer, 2011.

38 Wolf Rödiger, Tobias Mühlbauer, Philipp Unterbrunner, Angelika Reiser, Alfons Kemper, and Thomas Neumann. Locality-sensitive operators for parallel main-memory database clusters. In *ICDE*, pages 592–603, 2014.

39 Bart Samwel, John Cieslewicz, Ben Handy, Jason Govig, Petros Venetis, Chanjun Yang, Keith Peters, Jeff Shute, Daniel Tenedorio, Himani Apte, Felix Weigel, David Wilhite, Jiacheng Yang, Jun Xu, Jiexing Li, Zhan Yuan, Craig Chasseur, Qiang Zeng, Ian Rae, Anurag Biyani, Andrew Harn, Yang Xia, Andrey Gubichev, Amr El-Helw, Orri Erling, Zhepeng Yan, Mohan Yang, Yiqun Wei, Thanh Do, Colin Zheng, Goetz Graefe, Somayeh Sardashti, Ahmed M. Aly, Divy Agrawal, Ashish Gupta, and Shivakumar Venkataraman. F1 query: Declarative querying at scale. *PVLDB*, 11(12):1835–1848, 2018. URL: `http://www.vldb.org/pvldb/vol11/p1835-samwel.pdf`, `doi:10.14778/3229863.3229871`.

40 Marco Serafini, Rebecca Taft, Aaron J. Elmore, Andrew Pavlo, Ashraf Aboulnaga, and Michael Stonebraker. Clay: Fine-Grained Adaptive Partitioning for General Database Schemas. *PVLDB*, 10(4):445–456, 2016.

41 Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. F1: A distributed SQL database that scales. *PVLDB*, 6(11):1068–1079, 2013. `doi:10.14778/2536222.2536232`.

42 Bruhathi Sundarmurthy, Paraschos Koutris, and Jeffrey F. Naughton. Exploiting Data Partitioning To Provide Approximate Results. In *BeyondMR@SIGMOD*, pages 5:1–5:5, 2018.

43 Balder ten Cate, Phokion G. Kolaitis, and Walied Othman. Data exchange with arithmetic operations. In *EDBT*, pages 537–548, 2013.

44 Ron van der Meyden. The Complexity of Querying Indefinite Data about Linearly Ordered Domains. *J. Comput. Syst. Sci.*, 54(1):113–135, 1997. `doi:10.1006/jcss.1997.1455`.

45 Erfan Zamanian, Carsten Binnig, and Abdallah Salama. Locality-aware Partitioning in Parallel Database Systems. In *SIGMOD*, pages 17–30, 2015.

Towards Streaming Evaluation of Queries with Correlation in Complex Event Processing

Alejandro Grez
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
ajgrez@uc.cl

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
cristian.riveros@uc.cl

Abstract

Complex event processing (CEP) has gained a lot of attention for evaluating complex patterns over high-throughput data streams. Recently, new algorithms for the evaluation of CEP patterns have emerged with strong guarantees of efficiency, i.e. constant update-time per tuple and constant-delay enumeration. Unfortunately, these techniques are restricted for patterns with local filters, limiting the possibility of using joins for correlating the data of events that are far apart.

In this paper, we embark on the search for efficient evaluation algorithms of CEP patterns with joins. We start by formalizing the so-called partition-by operator, a standard operator in data stream management systems to correlate contiguous events on streams. Although this operator is a restricted version of a join query, we show that partition-by (without iteration) is equally expressive as hierarchical queries, the biggest class of full conjunctive queries that can be evaluated with constant update-time and constant-delay enumeration over streams. To evaluate queries with partition-by we introduce an automata model, called chain complex event automata (chain-CEA), an extension of complex event automata that can compare data values by using equalities and disequalities. We show that this model admits determinization and is expressive enough to capture queries with partition-by. More importantly, we provide an algorithm with constant update time and constant delay enumeration for evaluating any query definable by chain-CEA, showing that all CEP queries with partition-by can be evaluated with these strong guarantees of efficiency.

2012 ACM Subject Classification Information systems → Data streams; Theory of computation → Database query processing and optimization (theory); Theory of computation → Formal languages and automata theory; Theory of computation → Automata extensions

Keywords and phrases Complex event processing, Query languages, Correlation, Constant delay enumeration.

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.14

Funding A. Grez and C. Riveros were partially funded by the Millennium Institute for Foundational Research on Data.

1 Introduction

Streaming query evaluation is the most crucial problem in complex event processing (CEP). Given a CEP query Q, the streaming evaluation of Q over a stream consists in continuously reading events and outputting all complex events (i.e. sets of events) as soon as the last event that fires Q arrives. This streaming evaluation can be divided in two parts: (1) the process that continuously reads events and updates the state of the system whenever a new event arrives and (2) the process that outputs (i.e. enumerates) all complex events that satisfy the query. Both processes are required to run separately in such a way that the update process calls the enumeration process whenever a new output is found [17].

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 14; pp. 14:1–14:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Given the high-throughput data streams in areas like Network Intrusion Detection [27], Industrial Control Systems [19] or Real-Time Analytics [28], the time and space used by these two processes must be severely restricted. As proposed in [7, 17, 22], an efficient streaming evaluation process should satisfy at least the following two ideals: the update process must take constant time per new event and the enumeration process must take constant delay between two consecutive outputs. Intuitively, this is the best that a CEP system can aim for efficiently processing high-throughput data streams in practice. In [17] a streaming evaluation algorithm with constant update time per event and constant delay enumeration was shown for a meaningful core of CEP query languages when only local filters are allowed. Unfortunately, not all relevant queries in CEP can be evaluated with these strong guarantees, which fosters the search of query operators that allow efficient evaluation.

One of the key features in CEP is correlation [12]: to associate different events that might occur arbitrarily far in the input stream. Verifying that two users have the same id, or verifying an increasing sequence of temperature events, are some examples of how correlation is used in CEP. The most basic operator for adding correlation in CEP are equalities, namely, joining two events which have the same data value. Unfortunately, the evaluation of join queries is a difficult task even in a static setting [1], stressing the difficulties of finding efficient evaluation algorithms of CEP queries with equality predicates. One special operator usually included in CEP systems [5, 31, 15] for correlating events is partition-by [5] (also referred as *segmentation-oriented context* in [16] or just *context* in [15]). As the name suggests, this operator breaks up the events of a stream into partitions where all events of the same partition have the same data value. Despite being a useful operator in CEP, there is a lack of research in evaluating partition-by queries with solid efficiency guarantees, and usually this operator is severely restricted in CEP systems [31].

In this paper, we embark on the search for efficient evaluation of CEP queries with correlation when equality and disequality predicates are used. We first formalize the partition-by operator by extending Complex Event Logic (CEL) [17, 18] with a simple and compositional semantics. To motivate the expressive power of partition-by, we show that CEL with partition-by (but without iteration) is equally expressive as hierarchical queries [7, 22], the biggest subclass of conjunctive queries (CQ) that can be evaluated with constant update time and constant delay enumeration [7].

With a well-defined operator for doing correlation, we study the evaluation of partition-by through a machine model that we called chain Complex Event Automata (chain-CEA), an extension of complex event automata with equality and disequality predicates [17]. Although automata models over data words usually do not have good closure properties [29], we show that the chain-CEA model admits determinization and is expressive enough to capture all CEL queries with partition-by. The most important result of the paper is a streaming evaluation algorithm for the full class of chain-CEA, with constant update time and constant delay enumeration. In particular, this shows that all queries with partition-by can be evaluated efficiently in a streaming fashion.

Related work. Streaming query evaluation has been studied in the context of data stream management systems (DSMS) [5] and complex event processing (CEP) [31, 12, 21]. The notion of constant update time per tuple/event and constant delay enumeration has not been considered until recently [25, 20, 7] and, furthermore, in CEP systems these strong guarantees of efficiency have not been adopted yet [17]. Therefore, the algorithmic approach in CEP systems for evaluating queries with correlation is incomparable to our approach.

New techniques in dynamic query evaluation [8, 2, 9] have recently attracted a lot of attention [7, 22, 23]. In [7, 22], the streaming evaluation of CQ is considered but this does not include queries with order. In [23], inequalities over atoms are considered, but only for the case of CQ. Our setting also includes disjunction and iteration (but not conjunction), which makes our work orthogonal to the work in [7, 22, 23].

Register automata [24] have been extensively studied in the context of automata theory and XML [29]. Nevertheless, this model has not been studied in the context of CEP and efficient query evaluation. Recently, in [4] a similar extension of complex event automata with registers was proposed. However, this work does not study the determinization and evaluation of this model with constant update time and constant delay enumeration.

2 Preliminaries

In this section, we recall the formal definitions for streams and complex events [17], and give a simplified version of Complex Event Logic (CEL) [18, 17], originally called SO-CEL in [18]. We will later use CEL as a base language to model the partition-by operator.

Streams and complex events. Let $\mathbf{A}$ be a set of *attribute names* and $\mathbf{D}$ be an infinite set of values. A database schema $\mathcal{R}$ is a finite set of relation names, where each relation name $R \in \mathcal{R}$ is associated to a tuple of attributes denoted by $\text{att}(R)$. If R is a relation name, then an R-tuple is a function $t : \text{att}(R) \to \mathbf{D}$. Given $a \in \text{att}(R)$, we write $t.a$ to denote the value $t(a)$, and $\text{att}(t)$ to denote $\text{dom}(t)$. We say that the type of an R-tuple t is R, and denote this by $\text{type}(t) = R$. For any relation name R, $\text{tuples}(R)$ denotes the set of all possible R-tuples. Similarly, for any database schema $\mathcal{R}$, $\text{tuples}(\mathcal{R}) = \bigcup_{R \in \mathcal{R}} \text{tuples}(R)$. Given a schema $\mathcal{R}$, an $\mathcal{R}$-*stream* S is an infinite sequence $S = t_1 t_2 \ldots$ where $t_i \in \text{tuples}(\mathcal{R})$. When $\mathcal{R}$ is clear from the context, we refer to S simply as a stream. Given a stream $S = t_1 t_2 \ldots$ and a position $i \in \mathbb{N}$, the i-th element of S is denoted by $S[i] = t_i$.

A complex event C is defined as a non-empty and finite set of natural numbers. Intuitively, given a stream $S = t_1 t_2 \ldots$ a complex event $C = \{i_1, \ldots, i_n\}$ determines the set of tuples $\{t_{i_1}, \ldots, t_{i_n}\}$ and, thus, C represents the set of relevant events. We denote by $\min(C)$ and $\max(C)$ the minimum and maximum element of C, respectively. Given two complex events C_1 and C_2, we write $C_1 \cdot C_2$ for their *concatenation*, which is defined as $C_1 \cdot C_2 := C_1 \cup C_2$ whenever $\max(C_1) < \min(C_2)$ and empty otherwise. Given a complex event C we define $S[C] = \{S[i] \mid i \in C\}$, namely, the set of tuples in S positioned at the indices specified by C.

Complex event logic (CEL). Let $\mathbf{X}$ be a finite set of monadic second-order (SO) variables. An SO predicate of arity n is an n-ary relation P over sets of tuples, $P \subseteq (2^{\text{tuples}(\mathcal{R})})^n$. We write $\text{arity}(P) = n$. Let $\mathbf{P}$ be a set of SO predicates. An atom over $\mathbf{P}$ is an expression of the form $P(X_1, \ldots, X_n)$ where $P \in \mathbf{P}$ is a predicate of arity n, and $X_1, \ldots, X_n \in \mathbf{X}$ (we also write $P(\bar{X})$ for $P(X_1, \ldots, X_n)$). A CEL formula is defined by the following syntax:

$$\varphi \ := \ R \ \mid \ \varphi \texttt{ IN } X \ \mid \ \varphi \texttt{ FILTER } P(\bar{X}) \ \mid \ \varphi \texttt{ OR } \varphi \ \mid \ \varphi \,;\, \varphi \ \mid \ \varphi +$$

where R ranges over relation names, X over variables in $\mathbf{X}$ and $P(\bar{X})$ over atoms in $\mathbf{P}$. We say φ is an atomic formula if $\varphi = R$.

A valuation is a function $\mu : \mathbf{X} \to 2^{\mathbb{N}}$ such that $\mu(X)$ is a complex event for every $X \in \mathbf{X}$. We define the support of μ by $\text{supp}(\mu) = \bigcup_{X \in \mathbf{X}} \mu(X)$, and the union between μ_1 and μ_2 as $(\mu_1 \cup \mu_2)(X) = \mu_1(X) \cup \mu_2(X)$ for every $X \in \mathbf{X}$. Given a formula φ and a stream S, we say that a complex event C belongs to the evaluation of φ over S under the valuation μ (denoted by $C \in [\![\varphi]\!](S, \mu)$) if one of the following conditions holds:

- $\varphi = R$, $C = \{i\}$, $\text{type}(S[i]) = R$ and $\mu(X) = \emptyset$ for every X.
- $\varphi = \rho \texttt{ IN } X$, $\mu(X) = C$, and there exists a valuation μ' such that $C \in [\![\rho]\!](S, \mu')$ and $\mu(Y) = \mu'(Y)$ for all $Y \neq X$.

type	T	R	R	R	T	R	T	R	...
id	123	155	165	223	252	352	355	411	...
user-id	11	48	48	48	13	13	33	79	...
tweet-id		123	343	123		252		123	...
post/reply	#vote	#ihate	#ihate	#ihate	#vote	#ihate	#ihate	#stop	...
index	1	2	3	4	5	6	7	8	...

Figure 1 A stream S of events from Twitter. T are tweets with an id, a user-id and a post message, and R are responses with an id, a user-id, a tweet-id, and a reply message. The last line is the index of each event in the stream, respectively.

- $\varphi = \rho\ \texttt{FILTER}\ P(X_1, \dots, X_n)$, $C \in \llbracket \rho \rrbracket(S, \mu)$ and $(S[\mu(X_1)], \dots, S[\mu(X_n)]) \in P$.
- $\varphi = \rho_1\ \texttt{OR}\ \rho_2$ and ($C \in \llbracket \rho_1 \rrbracket(S, \mu)$ or $C \in \llbracket \rho_2 \rrbracket(S, \mu)$).
- $\varphi = \rho_1 \,;\, \rho_2$ and there exist complex events C_1 and C_2, and valuations μ_1 and μ_2 such that $C = C_1 \cdot C_2$, $\mu = \mu_1 \cup \mu_2$, $C_1 \in \llbracket \rho_1 \rrbracket(S, \mu_1)$ and $C_2 \in \llbracket \rho_2 \rrbracket(S, \mu_2)$.
- $\varphi = \rho+$, and $C \in \llbracket \rho \rrbracket(S, \mu)$ or $C \in \llbracket \rho \,;\, \rho+ \rrbracket(S, \mu)$.

We say that C belongs to the evaluation of a CEL formula φ over S at position $n \in \mathbb{N}$, denoted by $C \in \llbracket \varphi \rrbracket_n(S)$, if $C \in \llbracket \varphi \rrbracket(S, \mu)$ for some valuation μ, and $\max(C) = n$.

▶ **Example 1.** As a running example, suppose that we consider the stream from Twitter. For the sake of simplification, suppose that the stream is composed just by tweets (T) and replies (R). A tweet is composed by three attributes: an `id`, a `user-id` and a `post` message. Instead, a reply is composed by four attributes: an `id`, a `user-id`, a `tweet-id` of the replied message, and a `reply` message. Figure 1 shows an example of a stream with this schema.

As an example of a CEL formula, suppose that a journalist wants to detect all pairs of events composed by a tweet followed by a response containing '#voteforjohn' and '#ihatejohn', respectively, representing "hot" debates in Twitter about the election of a candidate called John. This query can easily be defined with the following CEL-formula:

$$\varphi_1 \ := \ (T\ \texttt{IN}\ X; R\ \texttt{IN}\ Y)\ \texttt{FILTER}\ (X.\texttt{post} = \text{`\#vote'}\ \ \texttt{AND}\ \ Y.\texttt{reply} = \text{`\#ihate'})$$

Here we make use of three operators: sequencing (;) to say we want to find complex events consisting of a T-tuple followed by an R-tuple; variable names (`IN`) to assign variables X and Y to T and R, respectively; and `FILTER` to define the conditions that the events must satisfy. We use conjunction (i.e., `AND`) as a syntactic sugar, which is short for applying a `FILTER` operator for each predicate of the conjunction. Predicates $X.\texttt{post} =$ '#vote' and $Y.\texttt{reply} =$ '#ihate' are basically restricting the T and R tuples t and r so that $t.\texttt{post}$ contains '#voteforjohn' and $r.\texttt{reply}$ contains '#ihatejohn'. Given a stream $S = t_1 t_2 \dots$ and a valuation μ, one can easily check that $\llbracket \varphi_1 \rrbracket(S, \mu)$ contains complex events of the form $\{k_1, k_2\}$ with $k_1 < k_2$ such that $t_{k_1}.\texttt{post}$ contains '#voteforjohn' and $t_{k_2}.\texttt{reply}$ contains '#ihatejohn'. For example, $\{1, 2\}$, $\{1, 3\}$ and $\{5, 6\}$ in Figure 1 are some outputs of φ_1 over S. Note that the replies are not necessarily replying to the tweet they are paired with, contrary to what one would like. We address this issue in the next section.

▶ **Example 2.** Suppose now that we want to find all sequences of debates that start with a tweet with '#voteforjohn', are followed by one or more responses with '#ihatejohn', and end with a response containing '#stophating'. This query can easily be defined in CEL using (+):

$$\begin{aligned} \varphi_2 \ = \ \big(T\ \texttt{IN}\ X \ ; \ \big(R+\big)\ \texttt{IN}\ Y \ ; \ R\ \texttt{IN}\ Z\big)\ \texttt{FILTER}\ \big(&X.\texttt{post} = \text{`\#vote'} \\ \texttt{AND}\ Y.\texttt{reply} = \text{`\#ihate'}\ \ &\texttt{AND}\ \ Z.\texttt{reply} = \text{`\#stop'}\big) \end{aligned}$$

In φ_2 we use the $(+)$ operator to extract an unbounded sequence of replies, which are then assigned to Y so that the predicate $Y.\mathtt{reply} =$ '#ihate' filters only the sequences where all tuples contain '#ihatejohn' (i.e. all tuples t in the complex event represented by Y must satisfy $t.\mathtt{reply} =$ '#ihate'). The other predicates are used to ensure that the T-tuple contains '#voteforjohn' and the last R-tuple contains '#stophating'. Finally, one can check that φ_2 defines the desired property. For example, if we evaluate φ_2 over S in Figure 1, then $\{1, 2, 4, 8\}$ and $\{1, 3, 4, 6, 8\}$ will be some outputs in $[\![\varphi_2]\!]_8(S)$.

A relevant feature in CEP is to skip arbitrary events when a formula is evaluated [12]. For example, for φ_1 it would make no sense looking for two contiguous events T and R. For this reason, the sequencing operator allows to skip an arbitrary number of events between two relevant events. The iteration operator has a similar semantics, which results in that for every sequence captured by it, the powerset of events is also captured. To remedy this problem CEL also includes the so-called selection strategies [17, 18], namely, operators for filtering the set of output to a meaningful subset. In this paper, our results do not include the evaluation over selection strategies. We leave this for future work.

CEL fragments and unary predicates. Given a set O of operators (e.g. `OR`, $+$), we define CEL$[O]$ to be the set of CEL formulas constructed from atomic formulas, `IN`, and operators in O. For example, φ_1 is in CEL$[;, \mathtt{FILTER}]$ and φ_2 is in CEL$[;, \mathtt{FILTER}, +]$. Furthermore, we define CEL$+O$ as the set of all CEL formulas extended with O.

Although CEL does not restrict the set of predicates that can be used by `FILTER`, not necessarily all predicates can be evaluated efficiently (or are even computable). For this reason, in [18] the analysis of CEL was restricted to SO-extensions of first-order unary predicates. Formally, let $\mathbf{U}$ be the set of all unary predicates over tuples, i.e. $\mathbf{U} = \{P \subseteq \text{tuples}(\mathcal{R})\}$. Given $P \in \mathbf{U}$ we define the SO-extension $P^{\text{SO}} \subseteq 2^{\text{tuples}(\mathcal{R})}$ of P such that $A \in P^{\text{SO}}$ if, and only if, $t \in P$ for all $t \in A$. We denote by $\mathbf{U}^{\text{SO}}$ the set of all SO-extensions of predicates in $\mathbf{U}$. When writing predicates with SO variables we are referring to the SO extension of the first order predicate. For example, $P := x.\mathtt{post} =$ '#vote' is a predicate in $\mathbf{U}$ such that $t \in P$ iff t has the attribute `post` and $t.\mathtt{post}$ contains #voteforjohn. Then $P^{\text{SO}} := (X.\mathtt{post} =$ '#vote'$)$ is the SO-extension of P that defines all complex events whose tuples satisfy P.

In [17, 18], it was shown that all CEL formulas restricted to predicates in $\mathbf{U}^{\text{SO}}$ can be evaluated efficiently. For this reason, from now on we assume that for any fragment or extension of CEL, all `FILTER` are restricted to predicates in $\mathbf{U}^{\text{SO}}$.

Streaming evaluation with constant-delay enumeration. As it is standard in the literature [7, 22], we consider evaluation algorithms on Random Access Machines (RAM) with addition and uniform cost measure [3]. Furthermore, we assume the existence of a key-value index (e.g. hash index) that allows insertions and deletions in $O(1)$ time and the index uses space linear in the number of insertions. In other words, we assume to have perfect hashing of linear size [10]. Although this is not realistic for practical computers, it can be simulated with a $O(\log(n))$-factor in the evaluation process with n the number of insertions in the index. Our complexity analysis is always in data complexity, namely, we assume that the CEL query φ and the schema $\mathcal{R}$ of the stream are fixed. Finally, we restrict the set $\mathbf{U}$ to unary predicates with constant time evaluation, namely, for every predicate P in $\mathbf{U}$ and every tuple t, we assume that checking whether $t \in P$ takes constant-time.

For efficient evaluation in CEP, we adapt the notion of constant-delay used in [6, 11] for streaming evaluation. Our evaluation process is a streaming algorithm divided in two parts: (1) consuming new events and updating the internal memory of the system and (2) generating complex events from the internal memory of the system. A streaming evaluation algorithm with *constant update time* and *constant-delay enumeration* is an algorithm that reads a stream $S = t_1 t_2 \dots$ sequentially and evaluates a formula φ over S such that (1) the time spent between reading t_i and t_{i+1} is bounded by $\mathcal{O}(|t_i|)$, and (2) it maintains a data structure D in memory, such that after reading t_n, the set $[\![\varphi]\!]_n(S)$ can be enumerated from D with constant-delay. The enumeration requires the existence of a routine Enumerate that enumerates $[\![\varphi]\!]_n(S) = \{C_1, C_2, \dots, C_m\}$ one by one without repetitions. We call $\text{delay}(C_i)$ the time it takes between enumerating C_i and C_{i+1}, and we say Enumerate runs with constant-delay if there exists a constant k depending only on φ such that $\text{delay}(C_i) = k \cdot |C_i|$ for all i. We remark that (1) is a natural restriction for a streaming algorithm, while (2) is the minimum requirement if an arbitrarily large set of arbitrarily large outputs must be produced [30]. Given that our analysis is in data complexity (i.e. φ and $\mathcal{R}$ are fixed), then the update time $\mathcal{O}(|t|)$ has a hidden factor that depends on $|\varphi|$ and $|\mathcal{R}|$.

3 Partition-by: syntax and semantics

Our main motivation in this paper is to study queries with correlation in CEP. One of the main operators for joining multiple events is partition-by [15, 31] (also referred as *segmentation-oriented context* in [16] or just *context* in [15]). Intuitively, events in a stream are usually correlated by an attribute that has the same value, e.g. an id. Then this attribute is "partitioning" the stream in multiple streams, where all events of the same stream contain the same value. In this section, we formally define the PART-BY operator in CEL, and motivate its usefulness by showing that it is expressive enough to define hierarchical queries.

Given two formulas φ_1 and φ_2, we denote by $\varphi_1 \subseteq \varphi_2$ when φ_1 is a subformula of φ_2. Consider a formula φ and variables $X_1, \dots, X_k$ of φ. We say that $X_1, \dots, X_k$ form a *variable cover* of φ if, for every atomic subformula ρ of φ, i.e. $\rho \subseteq \varphi$ and $\rho = R$ for some R, there is some $i \leq k$ and formula $\psi = \psi' \texttt{ IN } X_i$ such that $\rho \subseteq \psi \subseteq \varphi$, namely, all the events captured by atomic subformulas will be captured by some of the variables $X_1, \dots, X_k$ in φ. For example, in Example 2 variables X, Y and Z form a variable cover of φ_2.

We extend the syntax of CEL with the operator PART-BY as follows. A formula φ is in CEL+PART-BY if it satisfies the syntax of CEL, plus the following rule:

$$\varphi \ := \ \varphi \texttt{ PART-BY } [X_1.a_1, \dots, X_k.a_k]$$

where $X_1, \dots, X_k \in \mathbf{X}$ form a variable cover of φ and $a_1, \dots, a_k \in \mathbf{A}$ are attributes. The semantics of the PART-BY operator is defined as follows. Consider a complex event C, a stream $S = t_1 t_2 \dots$ and a valuation μ. Then, $C \in [\![\varphi \texttt{ PART-BY } [X_1.a_1, \dots, X_k.a_k]]\!](S, \mu)$ if $C \in [\![\varphi]\!](S, \mu)$ and for all $i, j \in \mathbb{N}$, $l \in \mu(X_i)$ and $m \in \mu(X_j)$, it holds that $S[l].a_i = S[m].a_j$. Thus, all events must contain the same data value in their attributes. For the case we only want to partition using a single attribute a that is common among all events (e.g. an id), we add the syntactic sugar $\varphi \texttt{ PART-BY } [a]$, which is defined as $\varphi \texttt{ PART-BY } [a] := (\varphi \texttt{ IN } X) \texttt{ PART-BY } [X.a]$, where X is a fresh variable that does not appear in φ. Clearly, X is a variable cover of φ.

▶ **Example 3.** In Example 1 we wanted to extract all pairs of tweets and replies that contain #voteforjohn and #ihatejohn, respectively. Although φ_1 extract these complex events, it fails to relate a reply with the tweet is replying to. For this, we can use the partition-by operator as follows:

$$\begin{aligned}\varphi_1^* \ := \ \big((T \texttt{ IN } X; R \texttt{ IN } Y) \texttt{ FILTER } (X.\text{post} = \text{`\#vote'} \\ \texttt{AND } Y.\texttt{reply} = \text{\#ihate'})\big) \texttt{ PART-BY } (X.\texttt{id}, Y.\texttt{tweet-id})\end{aligned}$$

Clearly, X, Y form a variable cover of φ_1. Furthermore, PART-BY restricts the output to pairs t and r with $t.\texttt{id} = r.\texttt{tweet-id}$. In Figure 1 now only $\{1,2\}$, $\{1,4\}$ and $\{5,6\}$ are in $[\![\varphi_1^*]\!](S)$.

▶ **Example 4.** Now, we want to restrict formula φ_2 in Example 2 in order to correlate tweets and replies in a meaningful way. Suppose that we want to restrict φ_2 such that all replies are replying to T and all #ihatejohn replies are from the same user. Then we can extend φ_2 with PART-BY to impose these restrictions (we omit the filters for the sake of readability):

$$\varphi_2^* \;=\; \Big[(T \texttt{ IN } X \;;\; (R\,+) \texttt{ PART-BY } (\texttt{user-id}) \texttt{ IN } Y \;;\; R \texttt{ IN } Z) \texttt{ FILTER } (\cdots)\Big]$$
$$\texttt{PART-BY } (X.\texttt{id}, Y.\texttt{tweet-id}, Z.\texttt{tweet-id})$$

This formula shows the advantage of using nesting of PART-BY. The internal PART-BY over attribute user-id restricts all #ihatejohn replies to have the same identifier, namely, they come from the same user. Then the external PART-BY forces all replies to have the same tweet-id as the first tweet and, therefore, they are replies of the same tweet. In Figure 1, $\{1,3,4,6,8\}$ is no longer an output but $\{1,2,4,8\}$ still is.

Partition-by and hierarchical queries. Partition-by models a join operator that usually appears in CEP systems [5, 15, 31]. Although this operator can be considered rather restrictive, interestingly, it is related to the class of hierarchical queries [13, 26], the biggest class of conjunctive queries without projection that can be evaluated in a streaming fashion [7, 22]. To formally define hierarchical queries we first introduce some notation. Given a database schema $\mathcal{R}$, we assume an arbitrary total order $<$ over the attribute names $\mathbf{A}$. For $R \in \mathcal{R}$ with $\text{att}(R) = \{a_1, \ldots, a_k\}$ and $a_1 < \ldots < a_k$, we write $R(x_1, \ldots, x_k)$ for variables $x_1, \ldots, x_k$ to denote that x_i is assigned to attribute a_i. We call $R(x_1, \ldots, x_k)$ an atom. A (full) conjunctive query Q is an expression $R_1(\bar{x}_1) \wedge \ldots \wedge R_k(\bar{x}_k)$ where each $R_i(\bar{x}_i)$ is an atom (i.e. we restrict our discussion to CQ without projection). Given a conjunctive query Q with k atoms and a stream $S = t_1 t_2 \ldots$ we say that a complex event C satisfies Q if $|C| \leq k$ and $\{t_i \mid i \in C\} \models Q$. We define $[\![Q]\!]_n(S)$ as all complex events C that satisfy Q and $\max(C) = n$.

From now on, we restrict our analysis to hierarchical conjuctive queries. Specifically, for a variable x define the set $\text{atom}(x)$ of all atoms in Q where x is mentioned. Then Q is hierarchical [13, 26] if for every x and y it holds that either $\text{atom}(x) \subseteq \text{atom}(y)$, $\text{atom}(x) \supseteq \text{atom}(y)$, or $\text{atom}(x) \cap \text{atom}(y) = \emptyset$. For example, the query $R(x) \wedge S(x,y)$ is hierarchical and $R(x) \wedge S(x,y) \wedge T(y)$ is not.

Unfortunately, CEL+PART-BY is not enough to capture the expressiveness of hierarchical queries. The reason is that partition-by combined with sequencing forces all events with correlated values to be "adjacent". On the other hand, hierarchical queries do not impose any order over tuples. For this reason, we consider the ALL-operator, a standard CEP operator studied in [18]. Formally, given formulas φ_1 and φ_2 we define the formula $\varphi_1 \texttt{ ALL } \varphi_2$ such that for a stream S and valuation μ it holds that $C \in [\![\varphi_1 \texttt{ ALL } \varphi_2]\!](S, \mu)$ if there exist complex events C_1, C_2, and valuations μ_1, μ_2 such that $C_1 \in [\![\rho_1]\!](S, \mu_1)$, $C_2 \in [\![\rho_2]\!](S, \mu_2)$, $C = C_1 \cup C_2$ and $\mu = \mu_1 \cup \mu_2$. In other words, ALL makes the pair union of complex events coming from evaluating φ_1 and φ_2, separately. Interestingly, CEL[ALL, PART-BY] captures exactly the expressiveness of hierarchical queries.

▶ **Proposition 5.** *For every hierarchical query Q, there is a formula φ in* CEL[ALL, PART-BY] *such that $[\![Q]\!]_n(S) = [\![\varphi]\!]_n(S)$ for every stream S and position n, and vice versa.*

The previous proposition shows the motivation of partition-by from the perspective of hierarchical CQ. Although CEL+PART-BY is not enough to capture the expressibility of hierarchical CQ, it shows that partition-by is related with a subclass of CQ that can be evaluated efficiently in a streaming fashion.

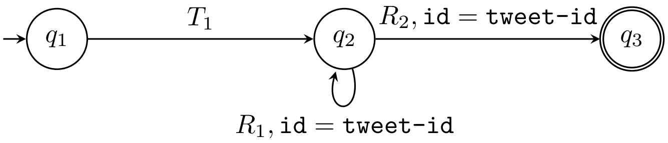

q_1
T_1
q_2
$R_2, \texttt{id} = \texttt{tweet-id}$
q_3
$R_1, \texttt{id} = \texttt{tweet-id}$

Figure 2 An example of chain-CEA with unary predicates $T_1 := \text{type}(T) \wedge \text{post} = \text{'\#vote'}$, $R_1 := \text{type}(R) \wedge \text{reply} = \text{'\#ihate'}$, and $R_2 := \text{type}(R) \wedge \text{reply} = \text{'\#stop'}$.

4 Chain complex event automata

Similarly to [17], we base our evaluation approach on an automata model to represent CEL+`PART-BY`. We present an automata model, called chain Complex Event Automata (chain-CEA), and show that each formula in CEL+`PART-BY` can be represented by this model.

In order to express the `PART-BY` operator, the automata model needs to be able to handle equality predicates. Given attributes $a, b \in \mathbf{A}$ define the equality and disequality predicates as $P_{a=b} = \{(t_1, t_2) \mid a \in \text{att}(t_1) \wedge b \in \text{att}(t_2) \wedge t_1.a = t_2.b\}$ and $P_{a \neq b} = \text{tuples}(R) \setminus P_{a=b}$. A *conjunctive binary predicate*, or binary predicate for short, is a predicate B that is a conjunction of equality and disequality predicates, i.e., $B = \bigcap_{i=1}^{n}(P_{a_i \sim b_i})$, where $a_i, b_i \in \mathbf{A}$ and $\sim_i \in \{=, \neq\}$. For simplicity, we usually drop the predicate notation and denote B simply as $\bigwedge_{i=1}^{n}(a_i \sim b_i)$. For example, $(a = b \wedge c \neq d)$ represents the predicate $B = P_{a=b} \cap P_{c \neq d}$, and thus $(t_1, t_2) \in B$ if $t_1.a = t_2.b$ and, if $c \in \text{att}(t_1)$ and $d \in \text{att}(t_2)$, then $t_1.c \neq t_2.d$. To separate equalities and disequalities from B, we will usually denote $B = B_= \wedge B_{\neq}$ where $B_=$ and $B_{\neq}$ are binary predicates composed only by equalities and disequalities, respectively. We denote by $\mathbf{B}$ the set of all binary predicates.

A *chain complex event automaton* (chain-CEA) is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of states, the transition relation Δ is a set of tuples (p, P, B, q), where $p, q \in Q$, $P \in \mathbf{U}$ and $B \in \mathbf{B}$, and $I, F \subseteq Q$ are the initial and final set of states, respectively. A configuration of $\mathcal{A}$ is defined by a state and a position in the stream, i.e. a pair $(q, i) \in Q \times \mathbb{N}$. An initial configuration is a pair (q, i) where $q \in I$ and $i = 0$. A run ρ of $\mathcal{A}$ over a stream $S = t_1 t_2 \ldots$ is a sequence of configurations: $(q_0, i_0) \xrightarrow{P_1/B_1} (q_1, i_1) \xrightarrow{P_2/B_2} \ldots \xrightarrow{P_n/B_n} (q_n, i_n)$ such that (q_0, i_0) is an initial configuration and, for every $j \leq n$: $i_{j-1} < i_j$, $(q_{j-1}, P_j, B_j, q_j) \in \Delta$, $t_{i_j} \in P_j$ and $(t_{i_{j-1}}, t_{i_j}) \in B_j$, where we consider t_0 being the empty tuple with no attributes. Further, the run ρ above induces the complex event $C_\rho = \{i_j \mid j > 0\}$. We say that ρ is an accepting run if $q_n \in F$. We define the set of complex events of $\mathcal{A}$ over S ending at position n as $[\![\mathcal{A}]\!]_n(S) = \{C_\rho \mid \rho \text{ is an accepting run of } \mathcal{A} \text{ and } \max\{C\} = n\}$.

It is worth noting that, even though only conjunctions and negations of equality predicates are allowed, in practice every logical combination (i.e. $\wedge$, $\vee$ and $\neg$) can be managed by simulating disjunction using multiple transitions. However, we need this restricted definition to later simplify the evaluation algorithm in Section 5.

▶ **Example 6.** Recall our complex events in Example 2 of a tweet with #voteforjohn, followed by one or more responses with #ihatejohn, and ending with a response saying #stophating. Suppose now that instead of correlating all responses with the first tweet, we want to extract a chain of responses, namely, for each contiguous responses r_1 and r_2 it holds that $r_1.\texttt{id} = r_2.\texttt{tweet-id}$ (i.e. r_2 is a reply of r_1). In Figure 2 we show a chain-CEA defining this query. If the automaton is in the initial state q_1 and receives a tweet t event containing #voteforjohn, it moves to q_2 and stores t. Then for each response r containing #ihatejohn whose `tweet-id` is equivalent to the id of the stored event, it forgets that event and stores r. Finally, when it receives an R-event containing #stophating which is responding the stored event, it reaches a final state.

The previous example shows a meaningful CEP query definable by a chain-CEA. This type of queries are very useful in practice (see for example query (7) in [12]). The next result shows that chain-CEA are expressive enough to cover the class of CEL+PART-BY formulas.

▶ **Proposition 7.** *For any formula φ in* CEL+PART-BY*, there exists a chain-CEA $\mathcal{A}$ such that $[\![\varphi]\!]_n(S) = [\![\mathcal{A}]\!]_n(S)$ for every S and n.*

On the other hand, one can show that the chain-CEA from Example 6 cannot be defined by any CEL+PART-BY formula. This, together with Proposition 7, shows that that CEL+PART-BY is strictly included in the queries defined by chain-CEA.

Like in [17] for CEA, here the determinization of chain-CEA is a crucial property for having efficient streaming evaluation and necessary property for removing duplicate runs that produce the same output. We start by defining our notion of deterministic chain-CEA. Similarly to [17], a deterministic chain-CEA must be "deterministic" with respect to the input and output, namely, given a stream S and a complex event C, there exists at most one run over S that produces C. Formally, we say that a chain-CEA $\mathcal{A} = (Q, \Delta, I, F)$ is I/O deterministic (or just deterministic) if $|I| = 1$ and, for every pair of transitions $(p, P_1, B_1, q_1) \neq (p, P_2, B_2, q_2)$, it holds that $(P_1 \cap B_1[t]) \cap (P_2 \cap B_2[t]) = \emptyset$ for every tuple t, where $B_i[t]$ is the set of all t' such that $(t, t') \in B_i$. In other words, the conditions (P_1, B_1) and (P_2, B_2) must be disjoint. One can easily check that the chain-CEA from Example 6 is deterministic.

▶ **Theorem 8.** *Chain-CEA admit determinization, namely, for any chain-CEA $\mathcal{A}$ there exists a deterministic chain-CEA $\mathcal{A}'$ such that $[\![\mathcal{A}]\!]_n(S) = [\![\mathcal{A}']\!]_n(S)$ for every S and n.*

A natural question that arises from the definition of chain-CEA is whether disequalities are strictly necessary in an automata model for CEP. For example, one can easily see that disequalities are not necessary for defining CEL+PART-BY formulas, since the partition-by operator only requires to check that the same value is used through a contiguous subsequence of the output. In the next result, we show that disequalities are indeed necessary if we want to find an automata model that admits determinization. More precisely, let chain-CEA$^=$ be the class of chain-CEA where all transitions are restricted to equalities.

▶ **Proposition 9.** *There exists a chain-CEA$^=$ $\mathcal{A}$ such that there exists no I/O deterministic chain-CEA$^=$ equivalent to $\mathcal{A}$.*

We are ready to state the main result of the paper.

▶ **Theorem 10.** *For every chain-CEA, there exists a streaming evaluation algorithm with constant update time and constant delay enumeration.*

By combining Proposition 7 and Theorem 10, we get that for any formula in CEL+PART-BY there exists a streaming evaluation algorithm with constant update time per tuple and constant delay enumeration. It is important to stress that chain-CEA is more general than CEL+PART-BY, in particular, the chain-CEA in Figure 2 cannot be defined by a CEL+PART-BY formula, but it can still be evaluated efficiently. We leave open whether there exists a set of predicates $\mathcal{P}$ (like PART-BY) such that CEL+$\mathcal{P}$ characterizes what is definable by chain-CEA.

5 A streaming evaluation algorithm for chain-CEA

In this section we show how to evaluate a chain-CEA over a stream with constant update time and constant-delay enumeration. We explain first the main data structures used by the algorithm to later show how to evaluate a chain-CEA.

The run DAG. In our algorithms, we compactly represent sets of runs by using a directed acyclic graph (DAG) annotated with configurations. Formaly, let $\mathcal{A} = (Q, \Delta, q_0, F)$ be a deterministic chain-CEA. A run DAG G of $\mathcal{A}$ (or just run DAG) is a tuple $G = (V, E, \bot, \kappa)$ consisting of a finite set of vertices V, a set of edges $E \subseteq V \times V$, a special vertex $\bot \in V$, and a function κ that maps every $v \in V$ to a configuration $\kappa(v) \in Q \times \mathbb{N}$ of $\mathcal{A}$. It is required that the graph (V, E) is acyclic, $\kappa(\bot) = (q_0, 0)$, and for every $v \in V$ there is a directed path from v to $\bot$. Furthermore, it is also required that for every $(u, v) \in E$ with $\kappa(u) = (q_1, i_1)$ and $\kappa(v) = (q_2, i_2)$, it holds that $i_1 > i_2$.

Intuitively, a vertex v labeled by $\kappa(v) = (q, i)$ is encoding the last configuration of a run over a stream S. Moreover, by the last two conditions every path starting in v and ending in $\bot$ is representing a run where configurations are listed in decreasing order. We make this intuition more precise as follows. Let $\pi = v_n, \ldots, v_1, \bot$ be a path from $v = v_n$ to $\bot$ in G and $\kappa(v_j) = (q_j, i_j)$ for $j \leq n$. Then $\kappa(\bot), \kappa(v_1), \ldots, \kappa(v_n)$ represents a run of $\mathcal{A}$ and $\mathrm{CE}(\pi) = \{i_1, \ldots, i_n\}$ the complex event defined by π. We denote by $\mathrm{CE}(v)$ the set of all complex events defined by paths from v to $\bot$ in G, and $\mathrm{CE}(U) = \bigcup_{v \in U} \mathrm{CE}(v)$ for $U \subseteq V$.

Note that there could be two paths starting from v in G that define the same complex event in $\mathrm{CE}(v)$. We say that a run DAG G is *safe* if $\mathrm{CE}(v_1) \cap \mathrm{CE}(v_2) = \emptyset$ for every $v_1, v_2 \in V$. Indeed, the safety property allows to enumerate all complex events in G without repetitions.

▶ **Lemma 11.** *Let $G = (V, E, \bot, \kappa)$ be a safe run DAG such that there is a procedure that, given any vertex $v \in V$, enumerates its neighborhood $\{u \mid (v, u) \in E\}$ with constant-delay. Then there exists a procedure that, given $U \subseteq V$, it enumerates $\mathrm{CE}(U)$ with constant delay.*

Therefore, by the previous lemma we can use a safe run DAG to encode the outputs of our evaluation algorithm for chain-CEA and enumerate these outputs with constant delay.

An index for binary predicates. In our evaluation algorithm we will need a special index over vertices of a run DAG to efficiently evaluate the binary predicates of a chain-CEA. Given a new event t and a state p, we want to quickly retrieve all configurations (p, i) that have reached p and such that $(t_i, t) \in B$ for some $e = (p, P, B, q) \in \Delta$. The run DAG will encode configurations (p, i), but we will need an index to store t_i and quickly "check" $(t_i, t) \in B$.

To define this index, we first need to introduce some notation. Let $B = \bigwedge_{i=1}^{n} (a_i \sim_i b_i)$ be a binary predicate with $\sim_i \in \{=, \neq\}$. Without loss of generality, we assume that all conditions $a_i \sim_i b_i$ in B are different. Let $\{(a_i, b_i)\}_i$ be a set of fresh attribute names not used in the schema $\mathcal{R}$. Given a tuple t, we define the left projection and right projection of t with respect to B as the tuples $\overleftarrow{\pi}_B(t)$ and $\overrightarrow{\pi}_B(t)$, respectively, with attributes in $\{(a_i, b_i)\}_i$ such that $\overleftarrow{\pi}_B(t).(a_i, b_i) = t.a_i$ whenever $a_i \in \mathrm{att}(t)$ and $\overrightarrow{\pi}_B(t).(a_i, b_i) = t.b_i$ whenever $b_i \in \mathrm{att}(t)$. Otherwise, if $a_i \notin \mathrm{att}(t)$ or $b_i \notin \mathrm{att}(t)$, then $\overleftarrow{\pi}_B(t).(a_i, b_i)$ and $\overrightarrow{\pi}_B(t).(a_i, b_i)$ are not defined, respectively. The left and right projections extract the relevant information of a tuple t to define $B[t]$. To see this, we say that t_1 and t_2 are *totally different*, denoted by $t_1 \not\equiv t_2$, if and only if $t_1.a \neq t_2.a$ for every $a \in \mathrm{att}(t_1) \cap \mathrm{att}(t_2)$, that is, they are different point-wise.

▶ **Lemma 12.** *Let $B = B_= \wedge B_{\neq}$ be a binary predicate. Then $(t, t') \in B$ if, and only if, $\overleftarrow{\pi}_{B_=}(t) = \overrightarrow{\pi}_{B_=}(t')$ and $\overleftarrow{\pi}_{B_{\neq}}(t) \not\equiv \overrightarrow{\pi}_{B_{\neq}}(t')$.*

With the previous notation, we are ready to define our index of a transition, called the *equality-disequality index* or ED-index for short. Let $G = (V, E, \bot, \kappa)$ be a run DAG and let $e = (p, P, B_= \wedge B_{\neq}, q)$ be a transition. We define the ED-index Index_e as a set of triples $(v, t_=, t_{\neq})$ where $v \in V$ and $t_=$, $t_{\neq}$ are left projections with respect to $B_=$ and $B_{\neq}$, respectively. Intuitively, Index_e will keep all configurations that are at state p and are

Algorithm 1 Evaluation of a det. chain-CEA $\mathcal{A} = (Q, \Delta, q_0, F)$ and a stream $S = t_1 t_2 \ldots$.

```
1:  procedure Evaluation($\mathcal{A}$, $S$)
2:      Init()
3:      for $i := 1$ to $\infty$ do
4:          FireTransitions($i$)
5:          UpdateIndices($i$)
6:          Enumerate($\cup_{q \in F} U_q^i$)
7:  procedure Init( )
8:      $G \leftarrow$ NewMappingGraph($q_0$)
9:      $U_{q_0}^0 \leftarrow \{\perp\}$
10:     for all $e_0 = (q_0, P, \emptyset, q) \in \Delta$ do
11:         $\text{Index}_{e_0}^0 \leftarrow \{(\perp, t_\emptyset, t_\emptyset)\}$
12: procedure FireTransitions($i$)
13:     for all $e = (p, P, B_= \wedge B_{\neq}, q) \in \Delta$ do
14:         $(t_=, t_{\neq}) \leftarrow (\vec{\pi}_{B_=}(t_i), \vec{\pi}_{B_{\neq}}(t_i))$
15:         if $t_i \in P \wedge \text{Index}_e^{i-1}[t_=, t_{\neq}] \neq \emptyset$ then
16:             $v \leftarrow$ AddNewVertex($G, q, i$)
17:             Connect($G, v, \text{Index}_e^{i-1}[t_=, t_{\neq}]$)
18:             $U_q^i \leftarrow U_q^i \cup \{v\}$
19: procedure UpdateIndices($i$)
20:     for all $e = (p, P, B_= \wedge B_{\neq}, q) \in \Delta$ do
21:         $\text{Index}_e^i \leftarrow \text{Index}_e^{i-1}$
22:         $(t_=, t_{\neq}) \leftarrow (\overleftarrow{\pi}_{B_=}(t_i), \overleftarrow{\pi}_{B_{\neq}}(t_i))$
23:         for all $v \in U_p^i$ do
24:             $\text{Index}_e^i \leftarrow \text{Index}_e^i \cup \{(v, t_=, t_{\neq})\}$
```

"waiting" to trigger e. More specifically, given a stream $S = t_1 t_2 \ldots$ if $(v, t_=, t_{\neq}) \in \text{Index}_e$ then $\kappa(v) = (p, i)$ and $t_= = \overleftarrow{\pi}_{B_=}(t_i)$ and $t_{\neq} = \overleftarrow{\pi}_{B_{\neq}}(t_i)$. Thanks to Lemma 12, whenever we want to check if $(t_i, t) \in B_= \wedge B_{\neq}$ for a new tuple t, we only need to obtain the tuple $(v, t_=, t_{\neq})$ from Index_e and check whether $t_= = \vec{\pi}_{B_=}(t)$ and $t_{\neq} \not\equiv \vec{\pi}_{B_{\neq}}(t)$. This motivates the following main query of an ED-index: given a pair of tuples $t'_=$ and $t'_{\neq}$:

$$\text{Index}_e[t'_=, t'_{\neq}] = \{v \in V \mid (v, t_=, t_{\neq}) \in \text{Index}_e \ \wedge t_= = t'_= \ \wedge \ t_{\neq} \not\equiv t'_{\neq}\} \qquad (1)$$

That is, $\text{Index}_e[t'_=, t'_{\neq}]$ returns all vertices v representing configurations $\kappa(v) = (p, i)$ such that there is a tuple t' with $t'_= = \vec{\pi}_{B_=}(t')$ and $t'_{\neq} = \vec{\pi}_{B_{\neq}}(t')$ and $(t_i, t') \in B_= \wedge B_{\neq}$. We will use the ED-index to store configurations and to quickly return them when e is fired.

The streaming evaluation algorithm. In Algorithm 1 we show how to evaluate a deterministic chain-CEA over a stream. The main procedure is Evaluation that receives as input a deterministic chain-CEA $\mathcal{A} = (Q, \Delta, q_0, F)$ and a stream $S = t_1 t_2 \ldots$. This procedure is composed of four subprocedures: Init for initializing the main data structures, FireTransitions(i) for firing the transitions in Δ given a new tuple t_i, UpdateIndices(i) for updating each Index_e given the previous tuple t_i, and, finally, Enumerate for enumerating all complex events ending at position i. For the sake of presentation, instead of having a yield function that provides each next tuple in the stream, we explicitly index each new phase by i (i.e. associated to tuple t_i) and iterate from 1 to "infinity" (the main for-loop at line 3). Then, given the next tuple t_i, in each i-phase we fire the transitions and update the indices with t_i, and enumerate all complex events at position i. In the sequel, we will first explain the data structures used by the algorithm to later describe each subprocedure.

Algorithm 1 maintains three structures that are used by all subprocedures: the run DAG $G = (V, E, \perp, \kappa)$, the ED-indices Index_e for each $e \in \Delta$, and set of vertices $U_q \subseteq V$ for each $q \in Q$. As it was explained before, G will encode runs of $\mathcal{A}$ and Index_e will allow us to quickly evaluate the binary predicate at e. Moreover, for each $q \in Q$ the set U_q will keep the new vertices v (i.e. configurations) at q. These sets will be useful for updating the indices and enumerating all new results. For the sake of presentation, we assume that G, Index_e, and U_q are defined globally and accessible by all subprocedures.

In each i-phase, the algorithm will update G to represent all runs of $\mathcal{A}$ over S until position i. To that end, it will use the following methods on run DAGs. The first method, `NewMappingGraph`(q_0), creates a new event DAG G containing only the vertex $\perp$ with

$\kappa(\bot) = (q_0, 0)$ and empty sets of vertices V and edges E. The second method, `AddNewVertex`, receives an event DAG G and a configuration (q, i), and creates a fresh vertex v with $\kappa(v) = (q, i)$, and adds it to V. Finally, the method returns the vertex v. The last method, `Connect`, receives as input a run DAG G, a vertex v on G, and a nonempty set of vertices $U \subseteq V$, and connects v with each vertex in U, namely, (v, u) is added to E for every $u \in U$. Although `AddNewVertex` and `Connect` could temporary break the properties of G (e.g. acyclicity), we will use it one after the other and it will be clear that the properties of G are always preserved.

For the structures Index_e and U_q, the reader might have noticed that in Algorithm 1 we use a superscript Index_e^i and U_q^i. This i is denoting the "version" of Index_e and U_q at phase i. We assume that each new i-version is always initialized as empty (i.e. $U_q^i = \emptyset$ and $\text{Index}_e^i = \emptyset$). It is important to note that for U_q^i we use the index i just to simplify the presentation (i.e. we could have reuse a set U_q in each phase). However, for Index_e^i the superscript is crucial to denote the version of Index_e when, for example, a vertex v is connected with the set $\text{Index}_e^i[t_=, t_{\neq}]$ (see line 17). As it will be discussed later (see Section 6), Index_e is a (partially) persistent data structure [14] and the superscript is denoting the i-version of the structure.

We are ready to describe each subprocedure in Algorithm 1. The algorithm starts with Init that is in charge of initializing G, Index_e^0, and U_q^0 before phase 1. For this, a new event DAG G is created and the vertex with the initial configuration $\bot$ is assigned to $U_{q_0}^0$ (recall that $U_q^i = \emptyset$ for $i \geq 0$ by assumption). Intuitively, this represents that the initial configuration is ready to start. For initializing Index_e, we assume without loss of generality that all outgoing transitions from q_0 use trivial predicates, namely, $B = \emptyset$ for every $e_0 = (q_0, P, B, q) \in \Delta$. Then $(\bot, t_\emptyset, t_\emptyset)$ is the only triple that must contain $\text{Index}_{e_0}^0$ with $t_\emptyset$ the empty tuple.

For each new phase i, we call FireTransitions(i) that check for each transition $e = (p, P, B_= \wedge B_{\neq}, q)$ whether it can be fired or not given the new tuple t_i (line 13). For this, we extract from t_i its right-projections $t_=$ and $t_{\neq}$ with respect to $B_=$ and $B_{\neq}$, respectively. Then we check if t_i satisfies P and whether there exists a previous configuration (p, j) such that (t_j, t_i) satisfies $B_= \wedge B_{\neq}$. We do this through Index_e, $t_=$, and $t_{\neq}$ by checking if $\text{Index}_e^{i-1}[t_=, t_{\neq}] \neq \emptyset$. If this is the case, all pairs of configurations (p, j) and (q, i) with $(p, j) \in \text{Index}_e^{i-1}[t_=, t_{\neq}]$ satisfy e and we must extend G with a new configuration (q, i) that represents all these new runs. For this, we create a new node v in G for configuration (q, i) and connect v with each vertex in $\text{Index}_e^{i-1}[t_=, t_{\neq}]$ (lines 16-17). Finally, the new vertex v is added to the set U_q^i of new vertices in state q at phase i.

The next step in phase i is to update Index_e^{i-1} to its new version Index_e^i given t_i. For this, we use the set U_p^i to update each transition $e = (p, P, B_= \wedge B_{\neq}, q)$. More specifically, in UpdateIndices(i) we iterate over each transition $e = (p, P, B_= \wedge B_{\neq}, q)$ and make Index_e^i equal to its previous version. Then, we extract from t_i its left-projections $t_=$ and $t_{\neq}$ with respect to $B_=$ and $B_{\neq}$, respectively, and add $(v, t_=, t_{\neq})$ to Index_e^i for each $v \in U_p^i$. Recall that U_p^i contains all the new vertices added during FireTransitions(i) and, in particular, $\kappa(v) = (p, i)$ for each $v \in U_p^i$. After UpdateIndices(i) is done, the ED-index Index_e^i contains all the relevant information for checking $B_= \wedge B_{\neq}$ in the next phases.

Up to this point, it is straightforward to prove the following invariant after each phase i, which leads to the correctness proof of Algorithm 1.

▶ **Lemma 13.** *Consider $\{U_q^i\}_{q \in Q}$ and G after the end of the i-phase. Then, for every run $(q_0, 0), (q_1, i_1) \ldots, (q_n, i_n)$ of $\mathcal{A}$ over S with $i_n = i$, there exist $v \in U_{q_n}^i$ and a path $v_n, \ldots, v_0$ in G with $v_n = v$ and $v_0 = \bot$ such that $\kappa(v_j) = (q_j, i_j)$ for every $j \leq n$. Conversely, for every $v \in U_q^i$ and every path $v_n, \ldots, v_0$ in G with $v_n = v$ and $v_0 = \bot$, it holds that $\kappa(v_0), \ldots, \kappa(v_n)$ is a run of $\mathcal{A}$ over S. Moreover, if $\mathcal{A}$ is deterministic, then G is safe.*

The final step at phase i is to enumerate all complex events of accepting runs. For this, we call the subprocedure Enumerate over the set of vertices $\cup_{q\in F} U_q^i$. By Lemma 13, we know that G correctly encodes all runs of $\mathcal{A}$ until the i-th tuple of S and, moreover, G is safe (i.e each complex event is represented by exactly one path in G). Therefore, we can easily enumerate all complex events $[\![\mathcal{A}]\!]_i(S)$ one-by-one and without repetitions, by enumerating all paths in G starting at vertices in $\cup_{q\in F} U_q^i$ and ending at $\bot$.

It is only left to show that Algorithm 1 satisfies constant update time and constant-delay enumeration. To do this, we have to dig deeper into the implementation of Index_e, which is the goal of the last section.

6 A persistent index structure for equalities and disequalities

Fix a transition $e = (p, P, B_= \wedge B_{\neq}, q)$. Let $(v_0, t_0, r_0), (v_1, t_1, r_1), \ldots$ be a sequence of triples such that v_i is a vertex and t_i, r_i are tuples for all $i \in \mathbb{N}$. Furthermore, define $\text{Index}_e^0 = \emptyset$ and $\text{Index}_e^i = \text{Index}_e^{i-1} \cup \{(v_i, t_i, r_i)\}$. Call (v_i, t_i, r_i) an insertion and i the version of Index_e.

To have constant update time and constant-delay enumeration, Index_e must satisfy the following properties, for every pair of tuples t, r and point in time $i \in \mathbb{N}$:

1. every new insertion in Index_e takes constant time, and
2. for all $j \leq i$, $\text{Index}_e^j[t, r]$ can be can be enumerated with constant-delay.

The last condition implies that Index_e is a persistent data structure [14], namely, it preserves the previous version (i.e. Index_e^j) of itself whenever it is modified.

We claim that, if Index_e satisfies the above three properties, then Algorithm 1 runs with constant update time and constant-delay enumeration. First, given that $\mathcal{A}$ is fixed, then it is clear that every step of Algorithm 1 can be done in constant time, except lines 15, 17, and 24. Checking whether $\text{Index}_e[t, r] \neq \emptyset$ (line 15) or doing an insertion in Index_e (line 24) can be done in constant time by properties (2) and (1), respectively. Furthermore, one can execute $\texttt{Connect}(G, v, \text{Index}_e^i[t, r])$ (line 17) in constant time if, instead of coding the graph G with adjacency lists, we represent the neighborhood of each vertex v by storing t, r, and i in v and, because of (2), we can later call $\text{Index}_e^i[t, r]$ whenever needed. Finally, from Lemma 11 we know that, if the neighborhood of each vertex from a safe run DAG can be enumerated with constant delay, then $\text{CE}(U)$ can also be enumerated with constant delay. Given that $\text{Index}_e^i[t, r]$ allows to enumerate the neighborhood of each vertex, then the enumeration with constant-delay follows.

In the sequel, we show how to implement Index_e in order to satisfy properties (1) and (2).

Case without disequalities. If e does not have disequalities (i.e. $B_{\neq}$ is trivial), then for every $(v, t, r) \in \text{Index}_e$, we can drop r and keep only (v, t). To satisfy (1) and (2) we use a key-value index $\texttt{DS}$ where keys are tuples t and each value $\texttt{DS}[t]$ is a list of pairs $(u_0, i_0), \ldots, (u_n, i_n)$ where each u_k is a vertex and i_k is a "timestamp", namely, the phase when u_k was inserted. Then, for every new insertion (u_i, t_i) in phase i, we go to $\texttt{DS}[t_i]$ and insert (u_i, i) at the end of the list. Finally, for every query of the form $\text{Index}_e^j[t]$ we can go into $\texttt{DS}[t]$, jump into the pair (u_k, i_k) with $i_k = j$ and enumerate $(u_k, i_k), \ldots, (u_0, i_0)$ with constant-delay. Recall that by our RAM model of computation, we can find the list $\texttt{DS}[t]$ and find the pair (u_k, i_k) inside $\texttt{DS}[t]$ in constant time (in the latter case, we need another key-value index for $\texttt{DS}[t]$ that, given j, finds $(\texttt{DS}[t])[j] = (u_k, j)$). Furthermore, by keeping $\texttt{DS}[t]$ as a linked list, one can easily enumerate $(u_k, i_k), \ldots, (u_0, i_0)$ with constant-delay.

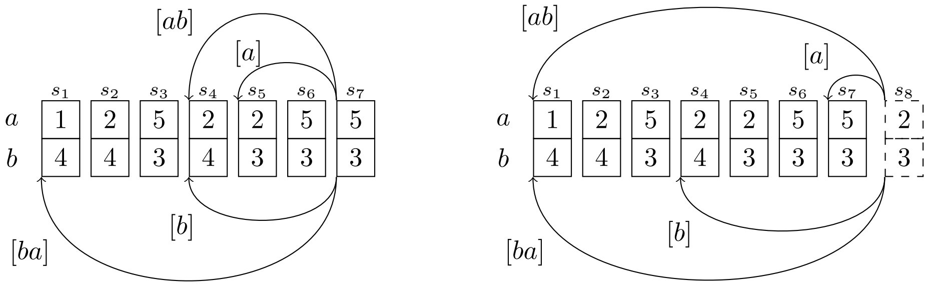


Figure 3 A list of tuples $s_1 \dots s_7$ with the additional bookkeeping to support disequalities.

Case with disequalities. If e includes disequalities (i.e. $B_{\neq}$ is non-trivial), then we need to extend our lists $\texttt{DS}[t]$ to support insertions (v_i, t_i, s_i) and queries $\text{Index}_e^i[t, r]$. For this, extend $\texttt{DS}[t]$ as a list of triples $(u_0, s_0, i_0), \dots, (u_n, s_n, i_n)$ where u_k and i_k are as before, and s_k is the tuple for supporting disequalities. Similar to the case without disequalities, for every new insertion (v_i, t_i, s_i) at phase i we go into the list $\texttt{DS}[t_i]$ and insert the triple (v_i, s_i, i) at the end of the list. Then for every query $\text{Index}_e^i[t, r]$ we can jump into the list $\texttt{DS}[t]$, jump into the triple (u_k, s_k, i_k) with $i_k = i$ and enumerate all u_l with $l \leq k$ such that $s_l \not\equiv r$ (i.e. s_l and r are totally different). Of course, this last enumeration step cannot be done with constant delay, unless some extra bookkeeping is added to the data structure. The rest of this section is then devoted to do this.

For the sake of simplification, from now on assume that each list $\texttt{DS}[t]$ is composed only by tuples $s_1, \dots, s_n$. Then the problem is reduced to, given a tuple r and position i, enumerate the set $\{s_k \mid k \leq i \wedge s_k \not\equiv r\}$. Without loss of generality, assume also that all $s_1, \dots, s_n$ have the same set of attributes A, i.e. $\text{att}(s_k) = A$, and define $d = |A|$. If not, complete each tuple s_k with the missing attributes and a fresh value for each new attribute. For example, at the left of Figure 3 we give a list $s_1, \dots, s_7$ with attributes $A = \{a, b\}$ and $d = 2$ where each column is a tuple (over integers) and each row is an attribute.

Let $\bar{a} = a_1 a_2 \dots a_m$ be a sequence of non-repeating attributes of A, and define $\bar{A}$ to be the set of all $\bar{a}$. For each tuple s_k and each $\bar{a}$, we define a tuple $s_k[\bar{a}] = s_j$ with $j < k$. Strictly speaking, $s_k[\bar{a}]$ will be a (backward) pointer from s_k to s_j that allows us to jump to $s_k[\bar{a}]$ in constant time. Given that our analysis is in data complexity, $|\bar{A}|$ is of constant size, so we only store a constant number of pointers in each tuple s_k (although exponential in d). In Figure 3, the pointers $[a]$, $[b]$, $[ab]$, and $[ba]$ of s_7 are displayed with arrows.

Now, for each s_k in the list $\texttt{DS}[t] = s_1, \dots, s_n$, the tuple $s_k[\bar{a}]$ is defined recursively as follows. First, for every attribute $a \in A$, $s_k[a]$ points to the maximum $j < k$ such that $s_k.a \neq s_j.a$. Next, for each sequence $\bar{a} = a_1 a_2 \dots a_m$, $s_k[\bar{a}]$ points to the maximum $j < k$ such that, for all $1 \leq l \leq m$, $s_j.a_l \neq s_k[a_1 \dots a_{l-1}].a_l$ where $s_k[\epsilon] = s_k$ (ϵ is the empty sequence in $\bar{A}$). In the case that there is no such tuple s_j, then $s_k[\bar{a}]$ is not defined, which means we reached the beginning of $\texttt{DS}[t]$.

▶ **Example 14.** Consider the list $s_1, \dots, s_7$ at the left of Figure 3 and consider tuple s_7. Then $s_7[a] = s_5$ is the last tuple before s_7 with a value different than 5, and $s_7[ab] = s_4$ is the last before s_7 with $s_4.a = 2 \neq 5 = s_7.a$ and $s_4.b = 4 \neq 3 = s_5.b$. Similarly, $s_7[b] = s_4$ is the last node before s_7 with $s_4.b = 4 \neq 3 = s_7.b$, and $s_7[ab] = s_1$ is the last before s_7 with $s_1.b = 4 \neq 3 = s_7.b$ and $s_1.a = 1 \neq 2 = s_4.a$.

With the previous structure over $s_1, \ldots, s_n$, we show how to enumerate with constant delay the set $\{s_k \mid k \leq i \wedge s_k \not\cong r\}$ given a tuple r and index i. For this, we define a procedure $\texttt{findNext}(s_k, r)$ that returns the last tuple s_j with $j < k$ such that $s_j \not\cong r$ (and false if s_j does not exist). Note that, if `findNext` runs in constant time, then we can enumerate the set $\{s_k \mid k \leq i \wedge s_k \not\cong r\}$ with constant delay: first, if $s_i \not\cong r$ then we enumerate s_i; then for every last node s_k we enumerated, we call $\texttt{findNext}(s_k, r)$ to get the next one, until `findNext` returns false. For computing $\texttt{findNext}(s_k, r)$, let $s := s_{k-1}$ be the node immediately before s_k in $\texttt{DS}[t]$. In the first step we check if $s[\epsilon]$ fulfills the condition, namely, if $s \not\cong r$. If so, we return $s[\epsilon]$; otherwise, there must be some attribute a_1 such that $s[\epsilon].a_1 = r.a_1$. In the next step we consider $s[a_1]$ and check if $s[a_1].a \neq r.a$ for each $a \in \text{att}(R) \setminus \{a_1\}$; if so, we return $s[a_1]$. Notice we do not need to compare r with all tuples between $s[a_1]$ and $s[\epsilon]$ because, by definition, each tuple s' between both satisfy $s'.a_1 = s[\epsilon].a_1 = r.a_1$. Furthermore, we no longer need to check the value of a_1 in $s[a_1]$ because $s[a_1].a_1 \neq s[\epsilon].a_1 = r.a_1$. We repeat this procedure inductively. If we are in step $1 \leq m < d$ and failed in all previous steps, then for $\bar{a} = a_1 \ldots a_m \in \bar{A}$, assume $s[a_1 \ldots a_{l-1}].a_l = r.a_l$ for every $l \leq m$. If $s[\bar{a}] \not\cong r$, return $s[\bar{a}]$; otherwise consider some attribute $a_{m+1} \in A \setminus \{a_1, \ldots, a_m\}$ such that $s[\bar{a}].a_{m+1} = r.a_{m+1}$. Then we consider $s[\bar{a} \cdot a_{m+1}]$ in the next step. Again, we do not need to compare r with all elements between $s[\bar{a} \cdot a_{m+1}]$ and $s[\bar{a}]$: each tuple s' between both satisfies $s'.a_{m+1} = s[\bar{a}].a_{m+1} = r.a_{m+1}$. Also we do not need to compare $s[\bar{a} \cdot a_{m+1}]$ with r on $\{a_1, \ldots, a_{m+1}\}$ given that, by induction, $s[\bar{a} \cdot a_{m+1}].a_{m+1} \neq s[\bar{a}].a_{m+1} = r.a_{m+1}$ and $s[\bar{a} \cdot a_{m+1}].a_l \neq s[a_1 \ldots a_{l-1}].a_l = r.a_l$. At some point we will find some tuple that fulfills the conditions; in the worst-case scenario we iterate d times, in which case we are sure by definition that $s[a_1 \ldots a_d]$ satisfies the condition or is undefined (i.e. it does not exists). All in all, the procedure takes $O(d)$ steps, which is constant. Moreover, this procedure does not use the pointers of s_k, but the ones of s_{k-1}. This is an important property that we use next when we want to insert a new node in $\texttt{DS}[t]$.

It is left only to show how to update $\texttt{DS}[t] = s_1, \ldots, s_n$ when we read a new tuple s_{n+1}. For this, we add s_{n+1} to the end of the list and define $s_{n+1}[\bar{a}]$ for each $\bar{a} \in \bar{A}$ in the following way. If the list is empty, then $s_{n+1}[\bar{a}]$ is undefined for all $\bar{a} \in \bar{A}$. Otherwise, for each $\bar{a} = a_1 \ldots a_m$ we define $s_{n+1}[\bar{a}]$ incrementally over the length m. Suppose that, $s_{n+1}[a_1 \ldots a_l]$ is already defined for every $l < m$. Define the tuple r such that $r.a_l = s_{n+1}[a_1 \ldots a_{l-1}].a_l$ for all $l < m$. Then, define $s_{n+1}[a_1 \ldots a_m] := \texttt{findNext}(s_{n+1}, r)$. In other words, we collect all values $c_1 = s_{n+1}[\epsilon].a_1$, $c_2 = s_{n+1}[a_1].a_2, \ldots, c_m = s_{n+1}[a_1 \ldots a_m].a_m$ and find the last tuple s such that $s.a_l \neq c_l$ for every $l \leq m$. As it was mentioned above, since `findNext` only uses the pointers of s_n, and not of s_{n+1} itself, the function is well-defined. Moreover, given that $\texttt{findNext}(s_{n+1}, r)$ can be found in constant time, then $s_{n+1}[a_1 \ldots a_m]$ is computed in constant time as well.

▶ **Example 15.** Suppose that we want to add the node $s_8 = \{a \to 2, b \to 3\}$ to the list on the left of Figure 3. The result is shown on the right of Figure 3 where s_8 is the last dashed column. We define $s_8[\bar{a}]$ incrementally using `findNext`. For a, we call $\texttt{findNext}(\texttt{n}_8, \{a \to 2\})$, which tries with the last tuple s_7 and, because $s_7.a \neq 2$, we set $s_8[a] := s_7$. For b, we call $\texttt{findNext}(s_8, \{b \to 3\})$, which first tries with s_7, but $s_7.b = 3$, so it tries with $s_7[b] = s_4$; since $s_4.b \neq s_7.b$, we set $s_8[b] = s_4$. For sequence ab, we have $s_8.a = 2$ and $s_8[a].b = 3$, so we call $\texttt{findNext}(s_8, \{a \to 2, b \to 3\})$. As s_7 conflicts in b, it tries with $s_7[b] = s_4$, but this time it conflicts with a, so it tries with $s_7[ba] = s_1$. As $s_1.a \neq 2$ and $s_1.b \neq 3$, we set $s_8[ab] = s_1$. The same procedure is done for ba, resulting in $s_8[ba] = s_1$.

By combining the key-value index `DS` where the keys are tuples and the values are the extended list with the additional bookkeeping mentioned above, we get properties (1) and(2) needed for Algorithm 1 to have constant update time and constant-delay enumeration.

7 Future work

This work rises several research opportunities regarding streaming evaluation of queries with correlation in CEP. The first problem is to find a unified class of queries that includes chain-CEA and hierarchical queries. Indeed, there are simple hierarchical queries (e.g. $R(x) \wedge S(y) \wedge T(x)$) that are not definable by chain-CEA. Another relevant question is whether partition-by queries with projection can be evaluated efficiently. Chain-CEA forbid the use of projection and it is not clear how to extend Algorithm 1 to support it. In particular, it is not clear how to extend this algorithm to support selection strategies [17], an important operator in CEP to filter the number of outputs. Finally, this work studies the streaming evaluation of equality and disequality predicates in CEP, but leaves open the evaluation of other predicates for correlation, like inequalities.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases: the logical level.* Addison-Wesley, 1995.

2 Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. *Proceedings of the VLDB Endowment*, 5(10):968–979, 2012.

3 A. Aho and J. Hopcroft. *The design and analysis of computer algorithms.* Pearson Education India, 1974.

4 E. Alevizos, A. Artikis, and G. Paliouras. Symbolic Automata with Memory: a Computational Model for CEP. *arXiv preprint arXiv:1804.09999*, 2018.

5 A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal*, 2006.

6 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *International Workshop on Computer Science Logic*, pages 167–181. Springer, 2006.

7 C. Berkholz, J. Keppeler, and N. Schweikardt. Answering conjunctive queries under updates. In *PODS*, pages 303–318, 2017.

8 Stefano Ceri and Jennifer Widom. Deriving Production Rules for Incremental View Maintenance. In *VLDB*, 1991.

9 Rada Chirkova, Jun Yang, et al. Materialized views. *Foundations and Trends® in Databases*, 4(4):295–405, 2012.

10 T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms.* MIT press, 2009.

11 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

12 G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 2012.

13 Nilesh N. Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 293–302, 2007.

14 James R Driscoll, Neil Sarnak, Daniel D Sleator, and Robert E Tarjan. Making data structures persistent. *Journal of computer and system sciences*, 38(1):86–124, 1989.

15 Esper Enterprise Edition website. `http://www.espertech.com/`. Accessed: 2018-12-21.

16 Opher Etzion, Peter Niblett, and David C Luckham. *Event processing in action.* Manning Greenwich, 2011.

17 A. Grez, C. Riveros, and M. Ugarte. A formal framework for Complex Event Processing. In *ICDT*, 2019.

18 A. Grez, C. Riveros, M. Ugarte, and S. Vansummeren. A Second-Order Approach to Complex Event Recognition. *arXiv preprint arXiv:1712.01063*, 2017.

19 M. Groover. *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall, 2007.

20 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30. ACM, 2015.

21 Martin Hirzel, Guillaume Baudart, Angela Bonifati, Emanuele Della Valle, Sherif Sakr, and Akrivi Akrivi Vlachou. Stream processing languages in the big data era. *ACM SIGMOD Record*, 47(2):29–40, 2018.

22 M. Idris, M. Ugarte, and S. Vansummeren. The dynamic Yannakakis algorithm: Compact and efficient query processing under updates. In *SIGMOD*, 2017.

23 M. Idris, M. Ugarte, S. Vansummeren, H. Voigt, and W. Lehner. Conjunctive queries with inequalities under updates. *VLDB*, 11(7):733–745, 2018.

24 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

25 Christoph Koch. Incremental query evaluation in a ring of databases. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 87–98. ACM, 2010.

26 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 223–234. ACM, 2011.

27 B. Mukherjee, T. Heberlein, and K. Levitt. Network intrusion detection. *IEEE network*, 1994.

28 B. Sahay and J. Ranjan. Real time business intelligence in supply chain analytics. *Information Management & Computer Security*, 2008.

29 L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, 2006.

30 L. Segoufin. Enumerating with constant delay the answers to a query. In *Proceedings of the 16th International Conference on Database Theory*, pages 10–20. ACM, 2013.

31 E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, 2006.

On the Expressiveness of Languages for Complex Event Recognition

Alejandro Grez
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
ajgrez@uc.cl

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
cristian.riveros@uc.cl

Martín Ugarte
Millennium Institute for Foundational Research on Data, Santiago, Chile
martin.ugarte@imfd.cl

Stijn Vansummeren
Université Libre de Bruxelles, Brussels, Belgium
stijn.vansummeren@ulb.ac.be

Abstract

Complex Event Recognition (CER for short) has recently gained attention as a mechanism for detecting patterns in streams of continuously arriving event data. Numerous CER systems and languages have been proposed in the literature, commonly based on combining operations from regular expressions (sequencing, iteration, and disjunction) and relational algebra (e.g., joins and filters). While these languages are naturally first-order, meaning that variables can only bind single elements, they also provide capabilities for filtering sets of events that occur inside iterative patterns; for example requiring sequences of numbers to be increasing. Unfortunately, these type of filters usually present ad-hoc syntax and under-defined semantics, precisely because variables cannot bind sets of events. As a result, CER languages that provide filtering of sequences commonly lack rigorous semantics and their expressive power is not understood.

In this paper we embark on two tasks: First, to define a denotational semantics for CER that naturally allows to bind and filter sets of events; and second, to compare the expressive power of this semantics with that of CER languages that only allow for binding single events. Concretely, we introduce Set-Oriented Complex Event Logic (SO-CEL for short), a variation of the CER language introduced in [17] in which all variables bind to sets of matched events. We then compare SO-CEL with CEL, the CER language of [17] where variables bind single events. We show that they are equivalent in expressive power when restricted to unary predicates but, surprisingly, incomparable in general. Nevertheless, we show that if we restrict to sets of binary predicates, then SO-CEL is strictly more expressive than CEL. To get a better understanding of the expressive power, computational capabilities, and limitations of SO-CEL, we also investigate the relationship between SO-CEL and Complex Event Automata (CEA), a natural computational model for CER languages. We define a property on CEA called the *-property and show that, under unary predicates, SO-CEL captures precisely the subclass of CEA that satisfy this property. Finally, we identify the operations that SO-CEL is lacking to characterize CEA and introduce a natural extension of the language that captures the complete class of CEA under unary predicates.

2012 ACM Subject Classification Information systems → Data streams; Information systems → Query operators; Theory of computation → Formal languages and automata theory

Keywords and phrases Query languages, Complex Event Recognition, Logics, Automata theory

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.15

Funding A. Grez, C. Riveros and M. Ugarte were partially funded by the Millennium Institute for Foundational Research on Data.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 15; pp. 15:1–15:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

type	T	H	H	T	H	T	H	H	T	H	...
value	-2	30	20	-1	27	2	45	50	-2	65	...
index	0	1	2	3	4	5	6	7	8	9	...

Figure 1 A stream S of events measuring temperature (T) in Celsius degrees and humidity (H) as a percentage of water in the air.

1 Introduction

The timely processing of data streams, where new data is continuously arriving, is a key ingredient of many contemporary Big Data applications. Examples of such applications include the recognition of: attacks in computer networks [10, 9]; human activities in video content [18]; traffic incidents in smart cities [4]; and opportunities in the stock market [20]. Numerous systems for processing streaming data have been proposed over the decades (see, e.g., [19, 12] for surveys). Complex Event Recognition (CER for short) systems are specialized stream processing systems that allow to detect higher-level *complex events* from streams of simple *events*. In CER systems, users write so-called *patterns* that describe the sequences of simple events that trigger the recognition of a complex event.

To support the above-mentioned application scenarios, numerous CER systems and languages have been proposed in the literature – see e.g., the surveys [12, 3] and the references therein. Most notably, CER is supported by several contemporary Big Data streaming engines, such as Trill [8] and Flink [6]. However, as noted in [12], the literature focuses mostly on the practical aspects of CER, resulting in many heterogeneous implementations with fundamentally different capabilities. As a result, little is known on the formal foundations of CER and, in contrast to the situation for relational databases, we currently lack a common understanding of the trade-offs between expressiveness and complexity in the design of CER languages, as well as an established theory for optimizing CER patterns.

Towards a better understanding of the formal foundations of CER, a subset of the authors has recently proposed and studied a formal logic that captures the core features found in most CER languages [17]. This logic, denoted CEL, combines the regular expression operators (sequencing, to require that some pattern occurs before another somewhere in a stream; iteration, to recognize a pattern a number of times; and disjunction) with data filtering features as well as limited data outputting capabilities. CEL follows the approach that seems to be taken by most of the CER literature (e.g., [14, 15, 1, 24, 11], see also [19, 12]) in that data filtering is supported by binding variables to individual events in the stream, which can later be inspected by means of one or more predicates. In this respect, variables in CEL are *first order* variables, since they bind to individual events.

One of the main contributions of CEL is to provide formal semantics for a language that combines filtering capabilities with iteration. In particular, a challenging yet common task in CER systems is to filter variables occurring inside a Kleene closure in a wider scope, stating properties that involve all the events *captured* in different iterations. For this reason, first order variables interact rather awkwardly with Kleene closure. Indeed, if a variable is bound inside Kleene closure, what does the variable refer to when used outside of a Kleene closure? In many of the practical CER languages, first order variables are used inside Kleene Closure to express properties on sequences of events rather than on individual events. In other words, first order variables are used to express properties on *second order* objects.

To illustrate this semantics issue, let us introduce the following running example. Suppose that sensors are positioned in a farm to detect freezing plantations. Sensors detect temperature and humidity, generating a stream of events of two types, T and H, both of which have a *value* attribute that contains the measured temperature or humidity, respectively. We encode each event as a relational tuple, and an event stream is an infinite sequence of events. Furthermore, events are assumed to appear in generation order in the stream. Figure 1 shows an example, where *index* marks the position of the event in the stream.

Suppose now that a farmer is interested in checking events of freezing plantations. One possible specification representing freezing plantations events could be the following:

> "*after having a temperature below 0 degrees, there is a period where humidity increases until humidity is over* 60%".

To motivate the semantics mismatch between first order variables and second order properties in CER languages, let us consider how one can define this complex event with two of the most influential CER languages in the literature [12], namely Cayuga [15, 14, 13] and SASE [24, 1, 23]. The CER languages of contemporary big data systems such as Trill [8] and Flink [6] are based on the former, and are therefore prone to the same mismatch.

1. In Cayuga, this complex event can be defined as follows:

$$\begin{aligned}&\texttt{FILTER}\{value < 0\}\ T\\ &\qquad\qquad \texttt{FOLD}\Big\{\$2.value < \$.value,\ \ \$2.value \geq \$.value \texttt{ AND } \ \$2.value \geq 60\Big\}\ H \qquad (1)\end{aligned}$$

   Here, the subexpression ($\texttt{FILTER}$ $\{value < 0\}$ T) takes the stream of all events of type T and produces a new stream only with those events satisfying $value < 0$. Then, this output stream is processed by the FOLD operator. A stream expression of the form $S1$ $\texttt{FOLD}\{\mathit{filter\_next}, \mathit{filter\_stop}\}$ $S2$ is processed as follows[1]. Every time you receive an event from the stream $S1$, start collecting all elements from the stream $S2$ that satisfy *filter_next*, until you see an element from the stream $S2$ satisfying *filter_stop*. This allows to perform some incremental computations and variables '\$2' and '\$' refer to the previous and current iteration, respectively. In our Cayuga query, $\$2.value < \$.value$ is checking that we see an increasing sequence of H values, until we see the last non-increasing H value that is over 60% (i.e. $\$2.value \geq \$.value$ AND $\$2.value \geq 60$).
2. In SASE, we can define the complex event more directly with the following query:

$$\begin{aligned}&\texttt{PATTERN}\,\mathrm{SEQ}(T\ t,\ \ H^{+}\ h1[],\ \ H\ h2)\\ &\qquad\qquad \texttt{WHERE}\ t.value < 0 \ \ \texttt{AND} \ \ h1[i-1].value < h1[i].value \texttt{ AND } \ h2.value \geq 60 \qquad (2)\end{aligned}$$

   The query looks for a T event (t), followed by one or more H events (collectively called $h1[]$), followed by a final H event ($h2$). The query then states that t's value is below zero, that subsequent events in $h1[]$ have increasing values, and $h2$'s value is above 60.

The two previous queries motivate the aforementioned ad-hoc semantics where first- and second-order objects are mixed. Cayuga uses first order variables (e.g. \$ and \$2) to define, in a procedural way, a property over a second order object: the set of H events. Instead, the SASE query implicitly combines first order variables (e.g. t and $h2$) with second order variables (e.g. $h1[]$). Indeed, SASE uses the ad-hoc notation $h1[i-1].value < h1[i].value$ to declare a predicate over the second order object $h1[]$. It is important to mention that in both languages the semantics of the second order objects is not formally defined and,

[1] Cayuga's FOLD operator actually also takes a third parameter that specifies the event to be output once the termination condition is met; for the sake of simplification we omit this parameter here.

moreover, second order objects are actually not acknowledged as such. As a consequence, CER languages are designed without any understanding of what the expressive power is of using first order versus second order variables, or how to compare them with other formalisms proposed in the literature. Moreover, while both Cayuga and SASE propose a computational model based on automata for evaluating queries, the exact relationship in expressive power between the CER language and their computational models has never been studied before.

In this paper, we embark on the task of understanding the expressive power of CER languages that only allow binding and filtering individual events versus those that allow binding and filtering sets of events, as well as their corresponding computational models. Concretely we consider CEL [17] as a model of the former class of languages, and we introduce Set Oriented Complex Event Logic (SO-CEL for short) as a model for the second class of languages. Variables in SO-CEL can only bind and filter sets of matched events. Specifically, we compare CEL against SO-CEL and show that they are equivalent in expressive power when equipped with the same unary predicates but, surprisingly, incomparable when equipped with n-ary predicates, $n > 1$. In particular, when equipped with sets of binary predicates, SO-CEL is strictly more expressive than CEL. However, when equipped with sets of ternary predicates, the languages are incomparable. The intuition behind this is that SO-CEL cannot distinguish the events captured by a single variable inside a Kleene closure, while this is possible in CEL by using a clever trick that relies on ternary filters (Section 4).

Since CEL and SO-CEL coincide when they are restricted to unary predicates, we study the expressiveness of this core CER language and compare it with a computational model for detecting complex events called *Complex Event Automata* [17] (CEA for short). We show that, in this setting, CEL and SO-CEL are strictly weaker than CEA, but capture the subclass of CEA that satisfy the so-called $*$-property. Intuitively, this property indicates that the CEA can only make decisions based on events that are part of the output. As a by-product of our development we are able to show that certain additional CER operators that have been proposed in the literature, such as `AND` and `ALL`, do not add expressive power to CEL and SO-CEL while others, such as `UNLESS`, provide the languages with new capabilities (Section 5).

Finally, we identify the operations that SO-CEL lacks to capture CEA and introduce a natural extension that captures the complete class of CEA under unary predicates. This is the first time that a CER language is proposed to capture the full expressive power of its underlying computational model. As a result we are also able to give insight into the `STRICT` selection policy and strict operator that are usually supported by CER languages (Section 6).

Related Work. As already mentioned, the focus in the majority of the CER literature is on the systems aspects of CER rather than on the foundational aspects, and there is little formal study of the expressiveness of CER languages. A notable exception is the work by Zhang et al on SASE$^+$ [24], which considers the descriptive complexity of a core CER language. It is unfortunate, however, that this paper lacks a formal definition of the language under study; and ignores in particular the aforementioned issues related to the scoping of variables under Kleene closure, as well as the data output capabilities.

Extensions of regular expressions with data filtering capabilities have been considered outside of the CER context. *Extended regular expressions* [2, 5, 7] extend the classical regular expressions operating on strings with variable binding expressions of the form $x\{e\}$ (meaning that when the input is matched, the substring matched by regular expression e is bound to variable x) and variable backreference expression of the form $\&x$ (referring to the last binding of variable x). Variables binding expressions can occur inside a Kleene closure, but when referred to, a variable always refers to the last binding. Extended regular expressions differ from SO-CEL and CEL in that they operate on finite strings over a finite alphabet rather than infinite streams over an infinite alphabet of possible events; and use variables only to filter the input rather than also using them to construct the output. Regular expressions with variable bindings have also been considered in the so-called spanners approach to information extraction [16]. There, however, variables are only used to construct the output and cannot be used to inspect the input. In addition, variable binding inside Kleene closures is prohibited.

Languages with second-order variables, such as monadic second order logic (MSO), are standard in logic and databases [21]. However, to the best of our knowledge we are not aware of any CER language such as SO-CEL that combines regular operators with variables that bind sets of events.

2 Preliminaries

In this section we introduce the formal definitions for streams and complex events, and recall the definition of CEL, as introduced in [17].

Schemas, Tuples and Streams. Let $\mathbf{A}$ be an infinite set of *attribute names* and $\mathbf{D}$ an infinite set of values. A database schema $\mathcal{R}$ is a finite set of relation names, where each relation name $R \in \mathcal{R}$ is associated to a tuple of attributes denoted by $\text{att}(R)$. If R is a relation name, then an R-tuple is a function $t : \text{att}(R) \to \mathbf{D}$. We say that the type of an R-tuple t is R, and denote this by $\text{type}(t) = R$. For any relation name R, $\text{tuples}(R)$ denotes the set of all possible R-tuples. Similarly, for any database schema $\mathcal{R}$, $\text{tuples}(\mathcal{R}) = \bigcup_{R \in \mathcal{R}} \text{tuples}(R)$. Given a schema $\mathcal{R}$, an $\mathcal{R}$*-stream* S is an infinite sequence $S = t_0 t_1 \dots$ where $t_i \in \text{tuples}(\mathcal{R})$. When $\mathcal{R}$ is clear from the context, we refer to S simply as a stream. Given a stream $S = t_0 t_1 \dots$ and a position $i \in \mathbb{N}$, the i-th element of S is denoted by $S[i] = t_i$, and the sub-stream $t_i t_{i+1} \dots$ is denoted by S_i. We consider in this paper that the time of each event is given by its index, and defer a more elaborated time model (like [22]) to future work.

CEL syntax. Let $\mathbf{X}$ be a set of first order variables. Given a schema $\mathcal{R}$, an FO predicate of arity n is an n-ary relation P over $\text{tuples}(\mathcal{R})$, $P \subseteq \text{tuples}(\mathcal{R})^n$. If $\mathcal{P}$ is a set of FO predicates then an atom over $\mathcal{P}$ is an expression $P(x_1, \dots, x_n)$ with $P \in \mathcal{P}$ of arity n and $x_1, \dots, x_n$ FO variables in $\mathbf{X}$. The set of formulas of $\text{CEL}(\mathcal{P})$ over schema $\mathcal{R}$ is given by the grammar:

$$\varphi := R \texttt{ AS } x \mid \varphi \texttt{ FILTER } P(\bar{x}) \mid \varphi \texttt{ OR } \varphi \mid \varphi \,;\, \varphi \mid \varphi + .$$

Here, R ranges over relation names in $\mathcal{R}$, x over variables in $\mathbf{X}$ and $P(\bar{x})$ over $\mathcal{P}$.

CEL semantics. For the semantics of CEL we first need to introduce the notion of *complex event.* A complex event C is defined as a non-empty and finite set of natural numbers. We denote by $\min(C)$ and $\max(C)$ the minimum and maximum element of C, respectively. Given two complex events C_1 and C_2, we write $C_1 \cdot C_2$ for their *concatenation*, which is defined as $C_1 \cdot C_2 := C_1 \cup C_2$ whenever $\max(C_1) < \min(C_2)$ and empty otherwise. Given an CEL formula φ, we denote by $\text{vdef}(\varphi)$ all variables defined in φ by a clause of the form $R \texttt{ AS } x$ and by $\text{vdef}_+(\varphi)$ all variables in $\text{vdef}(\varphi)$ that are defined outside the scope of all $+$-operators. For example, in the formula:

$$\varphi = (T \texttt{ AS } x \,;\, (H \texttt{ AS } y \texttt{ FILTER } y.id = x.id)+; (T \texttt{ AS } z)+) \texttt{ FILTER } (u.id = 1)$$

we have $\text{vdef}(\varphi) = \{x, y, z\}$ and $\text{vdef}_+(\varphi) = \{x\}$. A valuation is a function $\nu : \mathbf{X} \to \mathbb{N}$. Given a finite subset $U \subseteq \mathbf{X}$ and two valuations ν_1 and ν_2, we define the valuation $\nu_1[\nu_2/U]$ by $\nu_1[\nu_2/U](x) = \nu_2(x)$ whenever $x \in U$ and $\nu_1[\nu_2/U](x) = \nu_1(x)$ otherwise.

Now we are ready to define the semantics of CEL. Given an CEL-formula φ, we say that a complex event C belongs to the evaluation of φ over a stream S starting at position i, ending at position j, and under the valuation ν (denoted by $C \in [\![\varphi]\!](S, i, j, \nu)$) if $i \leq j$ and one of the following conditions holds:

- $\varphi = R \texttt{ AS } x$, $C = \{\nu(x)\}$, $\text{type}(S[\nu(x)]) = R$ and $i \leq \nu(x) = j$.[2]
- $\varphi = \rho \texttt{ FILTER } P(x_1, \ldots, x_n)$, and both $C \in [\![\rho]\!](S, i, j, \nu)$ and $(S[\nu(x_1)], \ldots, S[\nu(x_n)]) \in P$ hold.
- $\varphi = \rho_1 \texttt{ OR } \rho_2$, and $C \in [\![\rho_1]\!](S, i, j, \nu)$ or $C \in [\![\rho_2]\!](S, i, j, \nu)$.
- $\varphi = \rho_1 ; \rho_2$ and there exists $k \in \mathbb{N}$ and complex events C_1 and C_2 such that $C = C_1 \cdot C_2$, $C_1 \in [\![\rho_1]\!](S, i, k, \nu)$ and $C_2 \in [\![\rho_2]\!](S, k+1, j, \nu)$.
- $\varphi = \rho+$ and $C \in \bigcup_{k=1}^{\infty} [\![\rho[k]]\!](S, i, j, \nu)$ where $C \in [\![\rho[k]]\!](S, i, j, \nu)$ if there exists a valuation ν' such that $C \in [\![\rho]\!](S, i, j, \nu[\nu'/U])$ if $k = 1$ or $C \in [\![\rho ; \rho[k-1]]\!](S, i, j, \nu[\nu'/U])$ otherwise, where $U = \text{vdef}_+(\rho)$.

We say that C belongs to the evaluation of φ over S at position $n \in \mathbb{N}$, denoted by $C \in [\![\varphi]\!]_n(S)$, if $C \in [\![\varphi]\!](S, 0, n, \nu)$ for some valuation ν. Notice that the definition of CEL in [17] did not use the bounds i and j. We use them here just for consistency with the other definitions in the paper (SO-CEL and SO-CEL+).

▶ **Example 1.** Consider that we want to use CEL to see how temperature changes at some location whenever there is an increase of humidity from below 30 to above 60. Assume, for this example, that the location of an event (i.e. the location of a sensor) is recorded in its *id* attribute. Then, using a self-explanatory syntax for FO predicates, we would write:

$$[H \texttt{ AS } x ; (T \texttt{ AS } y \texttt{ FILTER } y.id = x.id)+ ; H \texttt{ AS } z] \\ \texttt{FILTER } (x.value < 30 \ \wedge z.value > 60 \wedge x.id = z.id)$$

Inside the Kleene closure, y is always bound to the current event being inspected. The filter $y.id = x.id$ ensures that the inspected temperature events are of the same location as the first humidity event x. Note that, in this case, the output is a complex event, and includes in particular the positions of the inspected T events.

3 Second-Order Complex Event Logic

In this section, we formally define SO-CEL, a core complex event recognition language in which all variables bind complex events instead of individual events. Before giving the formal definition, we first give a gentle introduction to SO-CEL and the design decisions behind its syntax and semantics.

As discussed in the Introduction, practical CER languages use variables that bind both single events (e.g. ‘\$’ in (1) and t in (2)) and complex events (e.g. $h[]$ in (2)). In SO-CEL variables bind to complex events, and predicates are over complex events instead of individual events. As an example, recall our statement for detecting freezing plantations: “*after having*

[2] The fact that $\nu(x) = j$ implies that the event in position j will always be part of the matched complex event. The reason behind this design decision is that one does not desire to wait for future events to decide whether or not a complex event matches a certain pattern.

a temperature below 0 degrees, there is a period where humidity increases until humidity is over 60%". This statement can be defined in SO-CEL with the following formula:

$$\varphi \ = \ T; (H+ \ \texttt{IN} \ HS); (H \ \texttt{IN} \ LH) \ \texttt{FILTER} \ (T.value < 0 \wedge \mathrm{incr}(HS) \wedge LH.value \geq 60) \quad (3)$$

To understand the meaning of this formula, note that T and H are relation names while HS (Humidity Sequence) and LH (Last Humidity) are variables. These two variables, HS and LH, are assigned to the complex events defined by the subformulas $H+$ and H, respectively, by using the IN-operator. For example, if $\{4, 6, 7\}$ is a complex event defined by $H+$ (i.e. a sequence of one or more H-events) then HS will be assigned to $\{4, 6, 7\}$. We denote this as $HS \to \{4, 6, 7\}$. Similarly, if the subformula H (i.e. only one H event) defines the complex event $\{9\}$, then $LH \to \{9\}$. Strictly speaking LH represents a complex event, although because of the pattern it will always contain only a single event. Note that T is not assigned to any variable in φ despite that we later used T in the filter clause. In SO-CEL we use relational names themselves also as variables; this generally decreases the number of variables in a formula and aids readability. Thus, T is also used as a variable in φ and $T \to \{3\}$ is a valid assignment of the T-events.

Now that all variables are assigned to complex events, we can check that they respect the order imposed by the sequencing operator (;): $T \to \{3\}$ is followed by the sequence $HS \to \{4, 6, 7\}$, which is followed by $LH \to \{9\}$. All together they form the complex event $C = \{3, 4, 6, 7, 9\}$. Indeed, variables T, HS and LH are assigned to the relevant part C which are used in the filter clause to check through built-in predicates that they satisfy the required conditions: (1) the temperature is below 0, (2) the humidity forms an increasing sequence and (3) the last humidity is over 60%. The first and third properties are naturally checked with the predicates $T.value < 0$ and $LH.value \geq 60$. The second property can be checked through a SO-CEL predicate that restricts the complex event in HS to form an increasing sequence of humidity values (similar to the predicate $h1[i-1].value < h1[i].value$ in (2)).

In SO-CEL we allow to use arbitrary predicates over complex events. This might seem too relaxed at first, as predicates could specify arbitrary properties. However, the goal of this approach is to separate what is inherent to a CER framework and what is particular to an application. In particular, each application is free to choose any set of predicates that can be useful and meaningful for users, as well as the algorithms and evaluation strategies to evaluate them. Next, we give the syntax and semantics of SO-CEL.

SO-CEL syntax. Let $\mathbf{L}$ be a finite set of SO (Set-Oriented) variables containing all relation names (i.e. $\mathcal{R} \subseteq \mathbf{L}$). An SO predicate of arity n is an n-ary relation P over sets of tuples, $P \subseteq (2^{\mathrm{tuples}(\mathcal{R})})^n$. We write arity($P$) for the arity of P. Let $\mathcal{P}$ be a set of SO predicates. An atom over $\mathcal{P}$ is an expression of the form $P(A_1, \ldots, A_n)$ where $P \in \mathcal{P}$ is a predicate of arity n, and $A_1, \ldots, A_n \in \mathbf{L}$ (we also write $P(\bar{A})$ for $P(A_1, \ldots, A_n)$). The set of formulas in SO-CEL($\mathcal{P}$) is given by the following syntax:

$$\varphi \ := \ R \ \mid \ \varphi \ \texttt{IN} \ A \ \mid \ \varphi \ \texttt{FILTER} \ P(\bar{A}) \ \mid \ \varphi \ \texttt{OR} \ \varphi \ \mid \ \varphi \, ; \varphi \ \mid \ \varphi+$$

where R ranges over relation names, A over labels in $\mathbf{L}$ and $P(\bar{A})$ over $\mathcal{P}$.

SO-CEL semantics. An SO valuation (or just valuation if clear from the context) is a function $\mu : \mathbf{L} \to 2^{\mathbb{N}}$ such that $\mu(A)$ is a finite set for every $A \in \mathbf{L}$. The support of such a valuation is defined as $\mathrm{supp}(\mu) = \bigcup_{a \in \mathbf{L}} \mu(A)$. Given two valuations μ_1 and μ_2, their union is defined by $(\mu_1 \cup \mu_2)(A) = \mu_1(A) \cup \mu_2(A)$ for every $A \in \mathbf{L}$. Finally, given a complex event C we define $S[C] = \{S[i] \mid i \in C\}$, namely, the set of tuples in S positioned at the indices specified by C.

Now we are ready to define the semantics of SO-CEL formulas. Given a SO-CEL formula φ, a stream S, and positions $i \leq j$, we say that a complex event C belongs to the evaluation of φ over a stream S starting at position i and ending at position j, and under the SO valuation μ (denoted by $C \in [\![\varphi]\!](S, i, j, \mu)$) if one of the following conditions holds:

- $\varphi = R$, $C = \mu(R) = \{j\}$, $\text{type}(S[j]) = R$ and $\mu(A) = \emptyset$ for every $A \neq R$.
- $\varphi = \rho \texttt{ IN } A$, $\mu(A) = C$, and there exists a valuation μ' such that $C \in [\![\rho]\!](S, i, j, \mu')$ and $\mu(B) = \mu'(B)$ for all $B \neq A$.
- $\varphi = \rho \texttt{ FILTER } P(A_1, \ldots, A_n)$, and both $C \in [\![\rho]\!](S, i, j, \mu)$ and $(S[\mu(A_1)], \ldots, S[\mu(A_n)]) \in P$ hold.
- $\varphi = \rho_1 \texttt{ OR } \rho_2$ and $C \in [\![\rho_1]\!](S, i, j, \mu)$ or $C \in [\![\rho_2]\!](S, i, j, \mu)$.
- $\varphi = \rho_1 \, ; \, \rho_2$ and there exists $k \in \mathbb{N}$, complex events C_1 and C_2, and valuations μ_1 and μ_2 such that $C = C_1 \cdot C_2$, $\mu = \mu_1 \cup \mu_2$, $C_1 \in [\![\rho_1]\!](S, i, k, \mu_1)$ and $C_2 \in [\![\rho_2]\!](S, k+1, j, \mu_2)$.
- $\varphi = \rho+$, and $C \in \bigcup_{k=1}^{\infty} [\![\rho^k]\!](S, i, j, \mu)$ where $\rho^k = \rho; \cdots ; \rho$ k-times.

Observe that, by definition, if $C \in [\![\varphi]\!](S, i, j, \mu)$ then C is a subset of $\{i, \ldots, j\}$ and $j \in C$. Furthermore, one can easily show by induction over the size of φ that the support of μ is equal to C, namely, $C = \text{supp}(\mu)$. Similar to CEL we say that C belongs to the evaluation of a SO-CEL formula φ over S at position $n \in \mathbb{N}$, denoted by $C \in [\![\varphi]\!]_n(S)$, if $C \in [\![\varphi]\!](S, 0, n, \mu)$ for some SO valuation μ.

▶ **Example 2.** Consider the formula φ in (3) that detects possible freezing plantations. We illustrate the semantics of φ over the stream S depicted in Figure 1 where event types T and H have both a *value* attribute and an *index* attribute recording their index in the stream.

First, note that although conjunction of predicates is not directly supported in SO-CEL, this can be easily simulated by a nesting of filter operators. Then, for the sake of simplification, we can analyze φ by considering each filter separately. For the subformula $\varphi_T = T \texttt{ FILTER } T.value < 0$ we can see that (i) $\{3\} \in [\![\varphi_T]\!](S, 0, 3, \mu_1)$ with $\mu_1(T) = \{3\}$. On the other hand, the last event (i.e. 9) is the only event that satisfies $\varphi_H = (H \texttt{ IN } LH) \texttt{ FILTER } LH.value \geq 60$ and then (ii) $\{9\} \in [\![\varphi_H]\!](S, 8, 9, \mu_2)$ with $\mu_2(LH) = \mu_2(H) = \{9\}$.

Now, the intermediate formula $\varphi_+ = (H+ \texttt{ IN } HS) \texttt{ FILTER } \text{incr}(HS)$ captures a sequence of one or more H-events representing an increasing sequence of humidities. Because Kleene closure allows for arbitrary events to occur between iterations, these sequences can be selected from the powerset of all H-events that produced an increasing sequence like, for example, $\{4, 6, 7\}$ or $\{2, 4\}$. In particular, we have that (iii) $\{4, 6, 7\} \in [\![\varphi_+]\!](S, 4, 7, \mu_3)$ with $\mu_3(LH) = \mu_2(H) = \{4, 6, 7\}$. Putting together (i), (ii) and (iii) and noticing that $\varphi = \varphi_T; \varphi_+; \varphi_H$, we have that $\{3, 4, 6, 7, 9\} \in [\![\varphi]\!](S, 0, 9, \mu)$ with $\mu = \mu_1 \cup \mu_2 \cup \mu_3$. Finally, we remove μ and $\{3, 4, 6, 7, 9\}$ is a complex event in $[\![\varphi]\!]_9(S)$.

The reader might find the semantics of SO-CEL more flexible and simpler than the one of CEL: the assignment of variables is more flexible and the semantics of iteration simpler (since variables are not re-assigned). We argue that the reason for this relies on the use of first-order variables in order to manage second-order objects (i.e. complex events). For example, in CEL variables can only be assigned at the event definition, with the atomic formula $R \texttt{ AS } x$. In contrast, second-order variables can manage complex events, allowing to use the IN-operator anywhere in a formula. Another more interesting example is iteration. In order to use first-order variables in a formula of the form $\varphi+$ we are forced to reassign these variables every time the subformula φ is evaluated (i.e. the use of the valuation $\nu_1[\nu_2/U]$). On the other hand, SO valuations can naturally be merged by union (i.e. $\mu_1 \cup \mu_2$) and, therefore, the iteration is just a simple generalization of the sequencing operator (;).

It is important to notice that it is possible to define a more general language CEL that includes first-order and second-order variables. Given that in this paper our expressiveness analysis is always between CEL and SO-CEL, we decide to present both language separately. We leave for future work the study of a CER query language that includes both approaches.

4 The Expressiveness of SO variables versus FO variables

In this section, we compare the expressiveness of CEL and SO-CEL. Since in traditional logics second-order languages can encode everything a first-order language can, this could suggest that SO-CEL is more expressive than CEL. We show that this is only partially true: SO-CEL includes CEL for binary predicates but they are incomparable in general.

In order to make a fair comparison between CEL and SO-CEL we first need to agree on how we relate the FO predicates that can be used in CEL to the SO predicates that can be used in SO-CEL. Indeed, the expressive power of both languages inherently depends on the allowed predicates, and we need to put them on equal ground in this respect. In particular, without any restrictions on the predicates of SO-CEL we can easily express formulas that are beyond the scope of CEL. For this reason, we will restrict ourselves to SO predicates created as *extensions* of FO predicates. Given an FO predicate $P(x_1, \ldots, x_n)$, we define its *SO-extension* P^{SO} to be the SO predicate of the same arity as P such that $(S_1, \ldots, S_n) \in P^{\mathrm{SO}}$ iff $\forall x_1 \in S_1, \ldots, x_n \in S_n$ it is the case that $(x_1, \ldots, x_n) \in P$. We extend this definition to sets of predicates: if $\mathcal{P}$ is a set of FO predicates, $\mathcal{P}^{\mathrm{SO}}$ is the set $\{P^{\mathrm{SO}} \mid P \in \mathcal{P}\}$. In what follows we will compare CEL($\mathcal{P}$) to SO-CEL($\mathcal{P}^{\mathrm{SO}}$).

▶ **Example 3.** Using the SO-extensions of the unary FO predicates (e.g. $X.value < 30 := \forall x \in X.\ x.value < 30$) and the binary id-comparison predicate (e.g. $X.id = Y.id := \forall x \in X. \forall y \in Y.\ x.id = y.id$), the CEL expression of Example 1 can be written in SO-CEL as:

$$(H \texttt{ IN } X; (T+ \texttt{ IN } Y); H \texttt{ IN } Z) \texttt{ FILTER}$$
$$(X.value < 30 \wedge Z.value > 60 \wedge X.id = Y.id \wedge X.id = Z.id).$$

One could ask why do we focus on *universal* extensions of FO predicates. After all, one could also consider *existential* extensions of the form $P^{\exists}$ where $(S_1, \ldots, S_n) \in P^{\exists}$ iff $\exists x_1 \in S_1, \ldots, x_n \in S_n.\ (x_1, \ldots, x_n) \in P$. Under this notion, SO-CEL cannot meaningfully filter events captured by a Kleene closure. For example, if $X.\mathtt{id} = Y.\mathtt{id}$ is used with an existential semantics in Example 3, it would include in Y the T events occurring between the first H event and the second H event, as long as there is one such T event with the corresponding id. Therefore, although existential extensions could be useful in some particular CER use-cases, we compare CEL with SO-CEL by considering only universal extensions.

We now compare both languages, considering the arity of the allowed predicates. We start by showing that if $\mathcal{U}$ is a set of unary FO predicates, CEL($\mathcal{U}$) and SO-CEL($\mathcal{U}^{\mathrm{SO}}$) have the same expressive power. Formally, we say that two formulas ψ and φ are equivalent, denoted by $\psi \equiv \varphi$, if $[\![\psi]\!]_n(S) = [\![\varphi]\!]_n(S)$ for every stream S and position n.

▶ **Theorem 4.** *Let $\mathcal{U}$ be any set of unary FO predicates. For every formula $\psi \in$ CEL($\mathcal{U}$) there exists a formula $\varphi \in$ SO-CEL($\mathcal{U}^{\mathrm{SO}}$) such that $\psi \equiv \varphi$, and vice versa.*

The previous theorem is of particular relevance since it shows that both languages coincide in a well-behaved core. CEL with unary predicates was extensively studied in [17] showing efficient evaluation algorithms and it is part of almost every CER language [12].

Now we show that if we go beyond unary predicates there are SO-CEL formulas that cannot be equivalently defined in CEL (under the same set of predicates). Let $\mathcal{P}_=$ be the smallest set of FO predicates that allows to express equality between attributes of tuples and is closed under boolean operations.

▶ **Theorem 5.** *There is a formula in SO-CEL($\mathcal{P}_=^{\mathrm{SO}}$) that cannot be expressed in CEL($\mathcal{P}_=$).*

An example of a formula that can be defined in SO-CEL($\mathcal{P}_=^{\mathrm{SO}}$) but cannot be defined in CEL($\mathcal{P}_=$) is $\varphi := (R+\,;\,T+)$ `FILTER` $R.id \neq T.id$, where $X.id \neq Y.id$ is defined as $\forall x \in X \forall y \in Y(x(id) \neq y(id))$. Intuitively, an equivalent formula in CEL($\mathcal{P}_=$) for φ would need to compare every element in R with every element in T, which requires a quadratic number of comparisons. The proof establishes that the number of comparisons in the evaluation of an CEL formula is at most linear in the size of the output and, thus, φ cannot be defined by any formula in CEL($\mathcal{P}_=$). It is important to note that this result shows the limitations of a CER language based on FO variables and what can be gained if SO variables are used.

A natural question at this point is whether SO-CEL can define every CEL formula. For binary predicates (e.g. $x.\mathtt{id} = y.\mathtt{id}$) the answer is positive, as the following result shows.

▶ **Theorem 6.** *Let $\mathcal{B}$ be any set of FO binary predicates closed under complement. Then for every formula $\psi \in$ CEL($\mathcal{B}$) there exists a formula $\varphi \in$ SO-CEL($\mathcal{B}^{\mathrm{SO}}$) such that $\psi \equiv \varphi$.*

It is important to notice that closedness under complement is a mild restriction over $\mathcal{B}$. In particular, if the set $\mathcal{B}$ is closed under boolean operations (as usually every CER query language supports), the condition trivially holds.

Interestingly, it is not true that SO-CEL is always more expressive than CEL. In particular, there exists an CEL formula with ternary predicates that cannot be defined by any SO-CEL formula. For the next result, consider the smallest set of FO predicates $\mathcal{P}_+$ containing the sum predicate $x = y + z$ that is closed under boolean operations.

▶ **Theorem 7.** *There is a formula in CEL($\mathcal{P}_+$) that cannot be expressed in SO-CEL($\mathcal{P}_+^{\mathrm{SO}}$).*

The formula R `AS` $x\,;(S$ `AS` $y\,;T$ `AS` z `FILTER` $(x = y + z))+$ cannot be defined in SO-CEL($\mathcal{P}_+^{\mathrm{SO}}$). This formula *injects* the x-variable inside the Kleene closure in order to check that each pair (y, z) sums x. This capability of injecting variables inside Kleene closure cannot be simulated in SO-CEL given that in SO-CEL a sub-formula cannot filter variables outside its own scope. It is important to recall that this does not occur if binary predicates are used (Theorem 6), which are of common use in CER.

5 On the Expressiveness of Unary Formulas

What is the expressiveness of CEL($\mathcal{P}$) or SO-CEL($\mathcal{P}$)? To obtain more insight into the the expressive power of the fundamental operators of these languages, we will study this question in the setting where $\mathcal{P}$ is limited to the class $\mathcal{U}$ of unary FO predicates. As we showed in Section 4, CEL($\mathcal{U}$) and SO-CEL($\mathcal{U}^{\mathrm{SO}}$) are equally expressive in this setting, suggesting that this is a robust subfragment of CER query languages. In this section, we compare CEL and SO-CEL with complex event automata (CEA), a computational model proposed in [17] for efficiently evaluating CEL with unary FO predicates. We show that the so-called $*$-property of CEA captures the expressiveness of CEL and SO-CEL with unary predicates. Furthermore, we use this property to understand the expressiveness of CEL and SO-CEL under the extension with new CER operators.

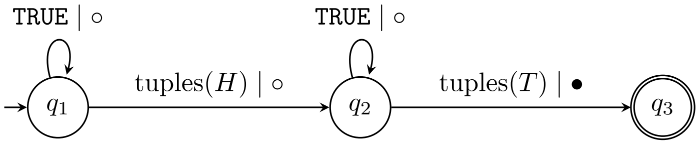


Figure 2 A complex event automaton that has no equivalent formula in SO-CEL.

Let $\mathcal{R}$ be a schema and $\mathcal{U}$ be a set of unary FO predicates over $\mathcal{R}$. We denote by $\mathcal{U}^+$ the closure of $\mathcal{U} \cup \{\text{tuples}(R) \mid R \in \mathcal{R}\}$ under conjunction. A *complex event automaton* [17] (CEA) over $\mathcal{R}$ and $\mathcal{U}$ is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of states, $\Delta \subseteq Q \times \mathcal{U}^+ \times \{\circ, \bullet\} \times Q$ is a finite transition relation, and $I, F \subseteq Q$ are the set of initial and final states, respectively. Intuitively, the elements $\{\circ, \bullet\}$ indicate whether or not the element used to take the transition will be part of the output. Given an $\mathcal{R}$-stream $S = t_0 t_1 \ldots$, a run ρ of length n of $\mathcal{A}$ over S is a sequence of transitions $\rho : q_0 \xrightarrow{P_0/m_0} q_1 \xrightarrow{P_1/m_1} \cdots \xrightarrow{P_n/m_n} q_{n+1}$ such that $q_0 \in I$, $t_i \in P_i$ and $(q_i, P_i, m_i, q_{i+1}) \in \Delta$ for every $i \leq n$. ρ is *accepting* if $q_{n+1} \in F$. $\text{Run}_n(\mathcal{A}, S)$ denotes the set of accepting runs of $\mathcal{A}$ over S of length n. Further, we define the complex event $C \subseteq 2^{\mathbb{N}}$ induced by ρ as $C_\rho = \{i \in [0, n] \mid m_i = \bullet\}$. Given a stream S and $n \in \mathbb{N}$, we define the set of complex events of $\mathcal{A}$ over S at position n as $[\![\mathcal{A}]\!]_n(S) = \{C_\rho \mid \rho \in \text{Run}_n(\mathcal{A}, S)\}$.

In [17], it was shown that for every formula $\varphi \in \text{CEL}(\mathcal{U})$ there exists an equivalent CEA $\mathcal{A}$ such that $[\![\varphi]\!]_n(S) = [\![\mathcal{A}]\!]_n(S)$ for every stream S and position n. By Theorem 4, it follows that for every formula $\varphi \in \text{SO-CEL}(\mathcal{U}^{\text{SO}})$ there is an equivalent CEA $\mathcal{A}$ such that $[\![\varphi]\!]_n(S) = [\![\mathcal{A}]\!]_n(S)$ for every stream S and position n. It is then natural to ask whether the converse also holds, namely, if every CEA $\mathcal{A}$ over $\mathcal{U}$ has an equivalent formula in $\text{SO-CEL}(\mathcal{U}^{\text{SO}})$ (and thus in $\text{CEL}(\mathcal{U})$). Here, however, the answer is negative because CEA can make decisions based on tuples that are not part of the output complex event, while formulas cannot. Consider for example the CEA of Figure 2. This automaton will output complex events of the form $C = \{i\}$, provided that $S[i]$ is of type T and there is a previous position $j < i$ such that $S[j]$ is of type H. It is straightforward to prove that this cannot be achieved by SO-CEL formulas because such a formula would either not check that the H event occurs, or include the position j of H in C – which the automaton does not.

In order to capture the exact expressiveness of CEL or SO-CEL formulas with unary predicates, we restrict CEA to a new semantics called the $*$-*semantics*. Formally, let $\mathcal{A} = (Q, \Delta, I, F)$ be a complex event automaton and $S = t_1, t_2, \ldots$ be a stream. A $*$-run ρ^* of $\mathcal{A}$ over S ending at n is a sequence of transitions: $\rho^* : (q_0, 0) \xrightarrow{P_1/\bullet} (q_1, i_1) \xrightarrow{P_2/\bullet} \cdots \xrightarrow{P_k/\bullet} (q_k, i_k)$ such that $q_0 \in I$, $0 < i_1 < \ldots < i_k = n$ and, for every $j \geq 1$, $(q_{j-1}, P_j, \bullet, q_j) \in \Delta$ and $S[i_j] \in P_j$. We say that ρ^* is an accepting $*$-run if $q_k \in F$. Furthermore, we denote by $C_\rho \subseteq 2^{\mathbb{N}}$ the complex event induced by ρ^* as $C_{\rho^*} = \{i_j \mid j \leq k\}$. The set of all complex events generated by $\mathcal{A}$ over S under the $*$-semantics is defined as: $[\![\mathcal{A}]\!]^*_n(S) = \{C_{\rho^*} \mid \rho^* \text{ is an accepting } *\text{-run of } \mathcal{A} \text{ over } S \text{ ending at } n\}$. Notice that under this semantics, the automaton no longer has the ability to verify a tuple without marking it but it is allowed to skip an arbitrary number of tuples between two marking transitions.

We can now effectively capture the expressiveness of unary formulas as follows.

▶ **Theorem 8.** *For every set $\mathcal{U}$ of unary FO predicates, SO-CEL($\mathcal{U}^{\text{SO}}$) has the same expressive power as CEA($\mathcal{U}$) under the $*$-semantics, namely, for every formula φ in SO-CEL($\mathcal{U}^{\text{SO}}$), there exists a CEA $\mathcal{A}$ over $\mathcal{U}$ such that $[\![\varphi]\!]_n(S) = [\![\mathcal{A}]\!]^*_n(S)$ for every S and n, and vice versa.*

For every stream S and complex event C, let $S[C\rangle$ refer to the subsequence of S induced by C. An interesting property of the $*$-semantics is that, for every CEA $\mathcal{A}$, stream S, and complex event $C \in [\![\mathcal{A}]\!]^*(S)$, we can arbitrarily modify, add and remove tuples in S that are not mentioned in $S[C\rangle$, and the original tuples in $S[C\rangle$ would still form a complex event of $\mathcal{A}$ over the new stream. To formalize this, we need some additional definitions. A *stream-function* f is a function $f : \text{streams}(\mathcal{R}) \to 2^{\mathbf{C}}$, where streams$(\mathcal{R})$ is the set of all $\mathcal{R}$-streams and $\mathbf{C}$ is the set of all complex events. Although f can be any function that returns a set of complex events on input streams, we are interested in the processing-functions f that can be described either by a SO-CEL formula φ (i.e. $f = [\![\varphi]\!]$) or by a CEA $\mathcal{A}$ (i.e. $f = [\![\mathcal{A}]\!]$). Let S_1, S_2 be two streams and C_1, C_2 be two complex events. We say that S_1 and C_1 are *$*$-related* with S_2 and C_2, written as $(S_1, C_1) =_* (S_2, C_2)$, if $S_1[C_1\rangle = S_2[C_2\rangle$.

Consider now a stream-function f. We say that f has the *$*$-property* if, for every stream S and complex event $C \in f(S)$, it holds that $C' \in f(S')$ for every S' and C' such that $(S, C) =_* (S', C')$. A way to understand the $*$-property is to see S' as the result of fixing the tuples in S that are part of $S[C]$ and adding or removing tuples arbitrarily, and defining C' to be the complex event that has the same original tuples of C. The following proposition states the relation that exists between the $*$-property and the $*$-semantics over CEA.

▶ **Proposition 9.** *If the stream-function defined by a CEA $\mathcal{A}$ has the $*$-property, then there exists a CEA $\mathcal{A}'$ such that $[\![\mathcal{A}]\!]_n(S) = [\![\mathcal{A}']\!]^*_n(S)$ for every S and n.*

By combining Theorem 8 and Proposition 9 we get the following result.

▶ **Corollary 10.** *Let f be a stream-function. Then f can be defined by a CEA over $\mathcal{U}$ and has the $*$-property iff there exists a formula φ in SO-CEL$(\mathcal{U}^{\mathrm{SO}})$ such that $f = [\![\varphi]\!]$.*

With the previous corollary we have captured the exact expressiveness of CEL$(\mathcal{U})$ and SO-CEL$(\mathcal{U}^{\mathrm{SO}})$ based on a restricted subclass of CEA. Interestingly, we can use this characterization to show that other operators for CER that have been proposed in the literature [12] can be captured by SO-CEL$(\mathcal{U}^{\mathrm{SO}})$. Some languages include additional useful operators like AND, ALL and UNLESS, which have the following semantics in SO-CEL. Given a complex event C, a stream S, a valuation μ, and $i, j \in \mathbb{N}$:

- $C \in [\![\rho_1 \texttt{ AND } \rho_2]\!](S, i, j, \mu)$ iff $C \in [\![\rho_1]\!](S, i, j, \mu) \cap [\![\rho_2]\!](S, i, j, \mu)$.
- $C \in [\![\rho_1 \texttt{ ALL } \rho_2]\!](S, i, j, \mu)$ if and only if there are $i_1, i_2, j_1, j_2 \in \mathbb{N}$, complex events C_1, C_2, and valuations μ_1, μ_2 such that $C_k \in [\![\rho_k]\!](S, i_k, j_k, \mu_k)$, $C = C_1 \cup C_2$, $\mu = \mu_1 \cup \mu_2$, $i = \min\{i_1, i_2\}$ and $j = \max\{j_1, j_2\}$.
- $C \in [\![\rho_1 \texttt{ UNLESS } \rho_2]\!](S, i, j, \mu)$ iff $C \in [\![\rho_1]\!](S, i, j, \mu)$ and, for every complex event C', valuation μ', and $i', j' \in \mathbb{N}$ such that $i \leq i' \leq j' \leq j$, it holds that $C' \notin [\![\rho_2]\!](S, i', j', \mu')$.

The AND operator selects those matches produced by both formulas. Although this is natural for sets, it is restrictive for capturing events. On the contrary, ALL is more flexible and allows to combine complex events. In this sense, ALL is similar to sequencing but allows the complex events to occur at any point in time, even overlapping or intersecting. For example, suppose that we want to capture a temperature below 0 degrees and a humidity over 60% that can occur in any order. This can be written as $(T \texttt{ ALL } H) \texttt{ FILTER } (T.value < 0 \wedge H.value \geq 60)$. The motivation for introducing UNLESS in CER languages is to have some sort of negation [12]. It is important to mention that the *negated* formula (the right-hand side) is restricted to complex events between the start and end of complex events for the formula in the left-hand side. This is motivated by the fact that a complex event should not depend on objects that are distant in the stream. For example, consider that we want to see a drastic increase in temperature, i.e., a sequence of a low temperature (less than 20 degrees) followed by a high temperature (more than 40 degrees), where no other temperatures occur in between. This can be expressed by the following pattern with the `UNLESS` operator:

$$[(T \texttt{ IN } \mathit{TF} \,;\, T \texttt{ IN } \mathit{TL}) \texttt{ FILTER } (\mathit{TF.value} < 20 \wedge \mathit{TL.value} > 40)]$$
$$\texttt{UNLESS } [T \texttt{ FILTER } (T.\mathit{value} >= 20 \wedge T.\mathit{value} <= 40)]$$

Interestingly, from a language design point of view, the operators `AND` and `ALL` are redundant in the sense that `AND` and `ALL` do not add expressive power in the unary case. Indeed, `AND` and `ALL` can be defined by CEA and both satisfy the $*$-property..

▶ **Corollary 11.** *Let $\mathcal{U}$ be a set of unary FO predicates. For every expression φ of the form φ_1 `OP` φ_2, with `OP` $\in \{$`AND`, `ALL`$\}$ and φ_i in SO-CEL($\mathcal{U}^{\mathrm{SO}}$), there is a SO-CEL($\mathcal{U}^{\mathrm{SO}}$) formula φ' such that $[\![\varphi]\!]_n(S) = [\![\varphi']\!]_n(S)$ for every S and n.*

In contrast, the `UNLESS` operator can be defined by CEA but one can show that there are formulas mentioning `UNLESS` that do not satisfy the $*$-property. Then, by Corollary 10, `UNLESS` is not expressible in SO-CEL($\mathcal{U}^{\mathrm{SO}}$) with $\mathcal{U}$ unary FO predicates. This shows that `UNLESS` adds expressibility to unary SO-CEL formulas while remaining executable by CEA.

6 Capturing the Expressive Power of Complex Event Automata

As discussed in Section 5, given a set $\mathcal{U}$ of unary FO predicates, SO-CEL($\mathcal{U}^{\mathrm{SO}}$) captures the class of CEA over $\mathcal{U}$ that have the $*$-property (Corollary 10). However, in [17] it was shown that all CEA can be evaluated efficiently, and not only those satisfying the $*$-property. It makes sense then to study the origin of this lack of expressive power and extend the language to precisely capture the expressiveness of the automata model.

6.1 Expressibility of CEA and Unary SO-CEL

By looking at the characterization of SO-CEL in terms of the $*$-property, one can easily distinguish three shortcomings of SO-CEL. First, every event that is relevant for capturing a complex event must be part of the output. Although this might be a desired property in some cases, it disallows projecting over a subset of the relevant events. This limitation is explained by the $*$-property, and suggests that to capture CEA we need an operator that allows to remove or, in other words, project events that must appear in the stream but are irrelevant for the output. Although projection is one of the main operators in relational databases, it is rarely used in the context of CER, possibly because of the difficulties encountered when trying to define a consistent semantics that combines projection with operators like Kleene closure. Interestingly, we show below that by using second-order variables it is straightforward to introduce a simple projection operator in SO-CEL.

The second shortcoming of SO-CEL is that it cannot express contiguous sequences. The sequencing operators (; and +) allow for arbitrary *irrelevant* events in between. While this is a typical requirement in CER, a user could want to capture contiguous events, which has been considered in some CER language before [23] as a *selection operator* that keeps contiguous sequences of events in the output (see Section 6.2 for further discussion). Given that this can be naturally achieved by CEA and has been previously proposed in the literature, it is reasonable to include some operators that allow to declare contiguous sequence of events.

A final feature that is clearly supported by CEA but not by SO-CEL is specifying that a complex event starts at the beginning of the stream. This feature is not particularly interesting in CER, but we include it as a new operator with the simple objective of capturing the computational model. Actually, this operator is intensively used in the context of regular expression programing where an expression of the form "$^\wedge R$" marks that R must be evaluated starting from the beginning of the document. Therefore, it is not at all unusual in query languages to include an operator that recognizes events from the beginning of the stream.

Given the discussion above, we propose to extend SO-CEL with the following operators:

$$\varphi \;:=\; \varphi : \varphi \;\mid\; \varphi\oplus \;\mid\; \pi_L(\varphi) \mid \; \texttt{START}(\varphi)$$

where $L \subseteq \mathbf{L}$. Recall that for a valuation μ, $\text{supp}(\mu)$ is defined as $\text{supp}(\mu) = \bigcup_{A\in\mathbf{L}} \mu(A)$. Given a formula φ of one of the forms above, a complex event C, a stream S, a valuation μ, and positions i, j, we say that $C \in [\![\varphi]\!](S, i, j, \mu)$ if one of the following conditions holds:

- $\varphi = \rho_1 : \rho_2$ and there exists two non-empty complex events C_1 and C_2 and valuations μ_1 and μ_2 such that $C = C_1 \cdot C_2$, $\mu = \mu_1 \cup \mu_2$, $C_1 \in [\![\rho_1]\!](S, i, \max(C_1), \mu_1)$, $C_2 \in [\![\rho_2]\!](S, \min(C_2), j, \mu_2)$ and $\max(C_1) = \min(C_2) - 1$.
- $\varphi = \rho\oplus$ and $C \in \bigcup_{k=1}^{\infty} [\![\rho^k]\!](S, i, j, \mu)$ where $\rho^k = \rho : \cdots : \rho$ k-times.
- $\varphi = \pi_L(\rho)$, $C = \text{supp}(\mu)$ and there is $C' \in [\![\rho]\!](S, i, j, \mu')$ for some valuation μ' such that $\mu(A) = \mu'(A)$ if $A \in L$ and $\mu(A) = \emptyset$ otherwise.
- $\varphi = \texttt{START}(\rho)$, $C \in [\![\varphi']\!](S, i, j, \mu)$, and $\min(C) = i$.

To denote the extension of SO-CEL with a set of operators $\mathcal{O}$ we write SO-CEL $\cup\, \mathcal{O}$. For readability, we use the special notation SO-CEL+ to denote SO-CEL $\cup \{ :, \oplus, \pi, \texttt{START}\}$.

The idea behind : and $\oplus$ is to simulate ; and +, respectively, but imposing that *irrelevant* events cannot occur in between. This allows us to recognize, for example, the occurrence of an event of type R immediately after an event of type T ($\varphi = R : T$), or an unbounded series of consecutive events of type R ($\varphi = R\oplus$). Note, however, that the operator $\oplus$ does not impose that intermediate events are contiguous. For example the formula $(R; S)\oplus$ imposes that the last event S of one iteration occurs right before the first event R of the next iteration, but in one iteration the R event and the S event do not need to occur contiguously.

▶ **Example 12.** Following the schema of our running example, suppose that we want to detect a period of temperatures below 0° and humidities below 40%, followed by a sudden increase of humidity (above 45%). Naturally, we do not expect to skip *irrelevant* temperatures or humidities, as this would defy the purpose of the pattern. Assuming that we are only interested in retrieving the humidity measurements, this pattern would be written as follows:

$$\pi_H[((H \texttt{ IN } X) \texttt{ OR } T)\oplus : (H \texttt{ IN } Y) \texttt{ FILTER } (X.value < 40 \wedge T.value < 0 \wedge Y.value > 45)].$$

Having defined the previous operators, we proceed to show that for every set $\mathcal{U}$ of unary predicates, SO-CEL+ ($\mathcal{U}^{\text{SO}}$) captures the full expressive power of CEA over $\mathcal{U}$. To this end, we say that a formula φ in SO-CEL+ ($\mathcal{U}^{\text{SO}}$) is equivalent to a CEA $\mathcal{A}$ over $\mathcal{U}$ (denoted by $\varphi \equiv \mathcal{A}$) if for every stream S and $n \in \mathbb{N}$ it is the case that $[\![\mathcal{A}]\!]_n(S) = [\![\varphi]\!]_n(S)$.

▶ **Theorem 13.** *Let $\mathcal{U}$ be a set of unary FO predicates. For every CEA $\mathcal{A}$ over $\mathcal{U}$, there is a formula $\varphi \in$ SO-CEL+ ($\mathcal{U}^{\text{SO}}$) such that $\varphi \equiv \mathcal{A}$. Conversely, for every formula $\varphi \in$ SO-CEL+ ($\mathcal{U}^{\text{SO}}$) there exists a CEA $\mathcal{A}$ over $\mathcal{U}$ such that $\varphi \equiv \mathcal{A}$.*

This result is particularly relevant because, as shown in [17], for every stream S and CEA $\mathcal{A}$, we can evaluate $\mathcal{A}$ by consuming the stream S using constant time to process every new event, and after consuming the n^{th} event of S the set $[\![\mathcal{A}]\!]_n(S)$ is enumerated with *constant delay.* Although the constants here are measured under data complexity and might depend exponentially on the size of the automaton, these are useful efficiency guarantees for CER in practice, and therefore extending SO-CEL to a language that precisely captures the class of CEA gives more expressive power while maintaining these efficiency guarantees.

6.2 Strict Sequencing versus Strict Selection

For recognizing events that occur contiguously we introduced the strict-sequencing operators (i.e. : and $\oplus$) that locally check this condition. These operators are the natural extension of ; and $+$, and they resemble the standard operators of concatenation and Kleene star from regular expressions. However, to the best of our knowledge strict-sequencing has not been proposed before in the context of CER, possibly because adding this feature to a language might complicate the semantics, specially when combined with other non-strict operators. To avoid this interaction, the strict-contiguity selection (or strict-selection) has been previously introduced in [23] by means of a unary predicate that basically forces a complex event C to capture a contiguous set of events. Formally, for any formula φ in SO-CEL let $\texttt{STRICT}(\varphi)$ be the syntax for the strict-selection operator previously mentioned. Given a stream S, a valuation μ, and two position $i, j \in \mathbb{N}$, we say that $C \in [\![\texttt{STRICT}(\varphi)]\!](S, i, j, \mu)$ if $C \in [\![\varphi]\!](S, i, j, \mu)$ and C is an interval (i.e. there are no $i, k \in C$ and $j \notin C$ s.t. $i < j < k$).

A reasonable question is whether the same expressiveness results of Theorem 13 could be obtained with STRICT. We answer this by giving evidence that our decision of including strict-sequencing operators instead of strict-selection was correct. We show that strict-sequencing and strict-selection coincide if we restrict our comparison to unary predicates. Surprisingly, if we move to binary predicates, strict-selection is strictly less expressive than strict-sequencing.

At a first sight, the strict-sequencing operators and the strict-selection predicates seem equally expressive since each allows to force contiguity between pairs of events. At least, this intuition holds whenever we restrict to unary predicates.

▶ **Proposition 14.** *Let $\mathcal{U}$ be a set of unary SO predicates. For every φ in SO-CEL$\cup\{\,:\,,\oplus\}(\mathcal{U})$, there exists a formula ψ in SO-CEL $\cup$ $\{\texttt{STRICT}\}(\mathcal{U})$ such that $\varphi \equiv \psi$, and vice-versa.*

The connection between both operators change if we move to predicates of higher arity. Note, however, that STRICT can always be simulated by the sequencing operators : and $\oplus$.

▶ **Proposition 15.** *Let $\mathcal{P}$ be a set of SO predicates. Given a formula $\varphi \in$ SO-CEL $\cup$ $\{\texttt{STRICT}\}(\mathcal{P})$ there exists $\psi \in$ SO-CEL $\cup$ $\{\,:\,,\oplus\}(\mathcal{P})$ such that $\varphi \equiv \psi$.*

To explain our decision of including the operators : and $\oplus$ instead of STRICT, we study the opposite direction. First, it is not hard to see that the operator : can indeed be simulated by means of the operator STRICT. Actually, for any formula $\varphi_1 : \varphi_2$ we can isolate the rightmost and leftmost event definition of φ_1 and φ_2 respectively, change : by ; and surround it by a STRICT operator. Now, if we include the operator $\oplus$, the situation becomes more complex. In particular, for binary predicates, STRICT is not capable of simulating the $\oplus$-operator.

▶ **Theorem 16.** *For any set $\mathcal{P}$ of SO predicates and for any formula $\varphi \in$ SO-CEL $\cup$ $\{:\}(\mathcal{P})$ there is a formula $\psi \in$ SO-CEL $\cup$ $\{\texttt{STRICT}\}(\mathcal{P})$ such that $\varphi \equiv \psi$. In contrast, there exists a set $\mathcal{P}$ containing a single binary SO predicate and a formula $\varphi \in$ SO-CEL $\cup$ $\{\oplus\}(\mathcal{P})$ that is not equivalent to any formula in SO-CEL $\cup$ $\{\texttt{STRICT}\}(\mathcal{P})$.*

This last theorem concludes our discussion on the operators for contiguity, and allows us to argue that including the operators : and $\oplus$ is better than including the unary operator STRICT. It is worth noting that the proof of Theorem 16 is a non-trivial result that requires a version of the pumping lemma for CEA; the proof can be found in the Appendix.

7 Discussion and future work

There are several future research directions regarding the relation between CER languages, logics, and streaming evaluation. For example, one relevant problem is to understand the connection between SO-CEL and monadic second-order logic (MSO). For unary filters, we conjecture that SO-CEL+ has the same expressive power as MSO over unary filters. Another natural question is to compare the expressiveness of SO-CEL+ and MSO extended with binary predicates. Furthermore, a more fundamental question is what fragments of SO-CEL or MSO (with binary predicates) can be evaluated with strong guarantees like constant-delay enumeration. We believe that understanding the relation between SO-CEL, formal logics (e.g. MSO), and constant delay algorithms is fundamental for the design of CER languages and the implementation of CER systems.

References

1 Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient Pattern Matching over Event Streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 147–160, 2008.

2 Alfred V. Aho. Algorithms for Finding Patterns in Strings. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 255–300. North Holland, 1990.

3 Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex Event Recognition Languages: Tutorial. In *International Conference on Distributed and Event-based Systems, DEBS 2017,*, pages 7–10, 2017.

4 Alexander Artikis, Marek J. Sergot, and Georgios Paliouras. An Event Calculus for Event Recognition. *IEEE Trans. Knowl. Data Eng.*, 27(4):895–908, 2015.

5 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A Formal Study Of Practical Regular Expressions. *Int. J. Found. Comput. Sci.*, 14(6):1007–1018, 2003. doi:10.1142/S012905410300214X.

6 Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015. URL: http://sites.computer.org/debull/A15dec/p28.pdf.

7 Benjamin Carle and Paliath Narendran. On Extended Regular Expressions. In *LATA 2009*, volume 5457 of *Lecture Notes in Computer Science*, pages 279–289, 2009. doi:10.1007/978-3-642-00982-2_24.

8 Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Robert DeLine, John C. Platt, James F. Terwilliger, and John Wernsing. Trill: A High-Performance Incremental Query Processor for Diverse Analytics. *PVLDB*, 8(4):401–412, 2014. doi:10.14778/2735496.2735503.

9 Charles D. Cranor, Yuan Gao, Theodore Johnson, Vladislav Shkapenyuk, and Oliver Spatscheck. Gigascope: high performance network monitoring with an SQL interface. In *SIGMOD*, page 623, 2002.

10 Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *SIGMOD*, pages 647–651, 2003.

11 Gianpaolo Cugola and Alessandro Margara. Complex Event Processing with T-REX. *The Journal of Systems and Software*, 2012.

12 Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 2012.

13 Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. A general algebra and implementation for monitoring event streams. Technical report, Cornell University, 2005.

14 Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. Towards expressive publish/subscribe systems. In *EDBT*, 2006.

15 Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. Cayuga: A General Purpose Event Monitoring System. In *CIDR 2007*, pages 412–422, 2007.

16 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. *J. ACM*, 62(2):12:1–12:51, 2015. `doi:10.1145/2699442`.

17 Alejandro Grez, Cristian Riveros, and Martin Ugarte. A formal framework for Complex Event Processing. In *ICDT*, 2019.

18 Mahmudul Hasan, Jonghyun Choi, Jan Neumann, Amit K. Roy-Chowdhury, and Larry S. Davis. Learning Temporal Regularity in Video Sequences. In *2016 Conference on Computer Vision and Pattern Recognition*, pages 733–742, 2016.

19 Martin Hirzel, Guillaume Baudart, Angela Bonifati, Emanuele Della Valle, Sherif Sakr, and Akrivi Vlachou. Stream Processing Languages in the Big Data Era. *SIGMOD Record*, 47(2):29–40, 2018. `doi:10.1145/3299887.3299892`.

20 Ilya Kolchinsky, Izchak Sharfman, and Assaf Schuster. Lazy evaluation methods for detecting complex events. In *International Conference on Distributed Event-Based Systems, DEBS '15*, pages 34–45, 2015.

21 Leonid Libkin. *Elements of finite model theory.* Springer Science & Business Media, 2013.

22 Walker M. White, Mirek Riedewald, Johannes Gehrke, and Alan J. Demers. What is "next" in event processing? In *PODS*, pages 263–272, 2007.

23 Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD*, 2006.

24 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*, 2014.

Infinite Probabilistic Databases

Martin Grohe
RWTH Aachen University, Germany
grohe@informatik.rwth-aachen.de

Peter Lindner
RWTH Aachen University, Germany
lindner@informatik.rwth-aachen.de

Abstract

Probabilistic databases (PDBs) are used to model uncertainty in data in a quantitative way. In the standard formal framework, PDBs are finite probability spaces over relational database instances. It has been argued convincingly that this is not compatible with an open-world semantics (Ceylan et al., KR 2016) and with application scenarios that are modeled by continuous probability distributions (Dalvi et al., CACM 2009).

We recently introduced a model of PDBs as infinite probability spaces that addresses these issues (Grohe and Lindner, PODS 2019). While that work was mainly concerned with countably infinite probability spaces, our focus here is on uncountable spaces. Such an extension is necessary to model typical continuous probability distributions that appear in many applications. However, an extension beyond countable probability spaces raises nontrivial foundational issues concerned with the measurability of events and queries and ultimately with the question whether queries have a well-defined semantics.

It turns out that so-called finite point processes are the appropriate model from probability theory for dealing with probabilistic databases. This model allows us to construct suitable (uncountable) probability spaces of database instances in a systematic way. Our main technical results are measurability statements for relational algebra queries as well as aggregate queries and Datalog queries.

2012 ACM Subject Classification Mathematics of computing → Probabilistic representations; Theory of computation → Incomplete, inconsistent, and uncertain databases; Theory of computation → Database query languages (principles)

Keywords and phrases Probabilistic Databases, Possible Worlds Semantics, Query Measurability, Relational Algebra, Aggregate Queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.16

Related Version Full version available on arXiv: `https://arxiv.org/abs/1904.06766`.

Funding *Peter Lindner*: This author is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Research Training Group 2236 UnRAVeL.

Acknowledgements We are grateful to Sam Staton for insightful discussions related to this work, and for pointing us to point processes. We also thank Peter J. Haas for discussions on the open-world assumption and the math behind the MCDB system.

1 Introduction

Probabilistic databases (PDBs) are used to model uncertainty in data. Such uncertainty could be introduced by a variety of reasons like, for example, noisy sensor data, the presence of incomplete or inconsistent information, or because the information is gathered from unreliable sources [3, 63]. In the standard formal framework, probabilistic databases are finite probability spaces whose sample spaces consist of database instances in the usual sense, referred to as "possible worlds". However, this framework has various shortcomings due

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 16; pp. 16:1–16:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to its inherent *closed-world assumption* [16] – in particular, any event outside of the finite scope of such probabilistic databases is treated as an impossible event. There is also work on PDBs that includes continuous probability distributions and hence goes beyond the formal framework of finite probability space. Yet, these continuous PDBs lack a general formal basis in terms of a possible worlds semantics [20]. While both open-world PDBs and continuous probability distributions in PDBs have received some attention in the literature, there is no systematic joint treatment of these issues with a sound theoretical foundation. In [38], we introduced an extended model of PDBs as arbitrary (possibly infinite) probability spaces over finite database instances. However, the focus there was on countably infinite PDBs. An extension to continuous PDBs, which is necessary to model probability distributions appearing in many applications that involve real-valued measurement data, raises new fundamental questions concerning the measurability of events and queries.

In this paper, we lay the foundations of a systematic and sound treatment of infinite, even uncountable, probabilistic databases, and we prove that queries expressed in standard query languages have a well-defined semantics.

Our treatment is based on the mathematical theory of finite point processes [53, 49, 18]. Adopting this theory to the context of relational databases, we give a suitable construction of measurable spaces over which our probabilistic databases can then be defined. The only assumption that we need to make is that the domains of all attributes satisfy certain topological assumptions (they need to be Polish spaces; all standard domains such as integers, strings, reals, satisfy this assumption). For queries and views to have a well-defined open-world semantics, we need them to be measurable mappings between probabilistic databases. Our main technical result states that indeed all queries and views that can be expressed in the relational algebra, even equipped with arbitrary aggregate operators (satisfying some mild measurability conditions) are measurable mappings. The result holds for both a bag-based and set-based relational algebra. We also prove the measurability of Datalog queries.

Measurability of queries may seem like an obvious minimum requirement, but one needs to be very careful. We give an example of a simple, innocent looking "query" that is not measurable (see Example 8). The proofs of the measurability results are not trivial, which may already be seen from the fact that they depend on the topological assumption that the attribute domains are Polish spaces (most importantly, they are complete topological spaces and have a countable dense subset). At their core, the proofs are based on finding suitable "countable approximations" of the queries.

In the last section of this paper, we briefly discuss queries for probabilistic databases that go beyond "standard" database queries lifted to probabilistic databases via an open-world semantics. Examples of such a queries are probabilistic threshold queries and rank queries. Such queries refer not only to the facts in a database, but also to their probabilities, and hence are inherently probabilistic.

Related Work. Early work on models for probabilistic databases dates back to the 1980s [69, 35, 15] and 1990s [8, 56, 26, 34, 71]. These models may be seen as special cases or variations of the now-acclaimed formal model of probabilistic databases that features a usually finite set of database instances (the "possible worlds") together with a probability distribution among them [3, 63].

The work [45] presents a formal definition of the probabilistic semantics of relational algebra queries as it is used in the MayBMS system [46]. A probabilistic semantics for Datalog has already been proposed in the mid-90s [33]. More recently, a version of Datalog was considered in which rules may fire probabilistically [25]. Aggregate queries in probabilistic databases were first treated systematically in [59] and reappear in various works concerning particular PDB systems [54, 28].

The models of possible worlds semantics mentioned above are the mathematical backbone of existing probabilistic database prototype systems such as MayBMS [46], Trio [68] and MystiQ [12]. Various subsequent prototypes feature uncountable domains as well, such as Orion [60], MCDB [41, 42], new versions of Trio [4] and PIP [44]. The MCDB system in particular allows programmers to specify probabilistic databases with infinitely many possible worlds with database instances that can grow arbitrarily large [42] and is therefore probably the most general existing system. Its system-driven description does not feature a general formal, measure theoretic account of its semantics though. In a spirit that is similar to our presentation here, the work [64] introduced a measure theoretic semantics for probabilistic data stream systems with probability measures composed from Gaussian mixture models but (to our knowledge) on a per tuple basis and without the possibility of inter-tuple correlations. Continuous probabilistic databases have already been considered earlier in the context of sensor networks [27, 17, 24]. The first work to formally introduce continuous possible worlds semantics (including aggregation) is [1] for probabilistic XML. However, the framework has an implicit restriction bounding the number of tuples in a PDB.

Models similar in expressivity to the one we present have also been suggested in the context of probabilistic modeling languages and probabilistic programming [51, 52, 58, 22, 7]. In particular notable are the measure theoretic treatments of Bayesian Logic (BLOG) [51] in [70] and Markov Logic Networks (MLNs) [58] in [61]. While these data models are relational, it is unclear, how suitable they are for general database applications and in particular, the investigation of typical database queries is beyond the scope of these works.

Problems raised by the closed-world assumption [57] in probabilistic databases was discussed initially by Ceylan et al. in [16] where they suggest the model of OpenPDBs. In [10], the authors make a more fine-grained distinction between an *open-world* and *open-domain assumption*, the latter of which does not assume the attribute values of the database schema to come from a known finite domain. The work [31] considers semantic constraints on open worlds in the OpenPDB framework. The semantics of OpenPDBs can be strengthened towards an open-domain assumption by the means of ontologies [9, 10, 11].

The classification of views we discuss towards the end of this paper shares similarities with previous classifications of queries such as [17] in the sense that it distinguishes *how* aggregation is involved. The work [66] suggests a distinction between “traditional” and “out-of-world aggregation” quite similar to the one we present.

2 Preliminaries

Throughout the paper, we denote the set of nonnegative integers by $\mathbb{N}$, the set of rational numbers by $\mathbb{Q}$ and the set of real numbers by $\mathbb{R}$. We write $\mathbb{N}_+$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ for the restrictions of these sets to strictly positive numbers.

If M is a set and $k \in \mathbb{N}$, then $\binom{M}{k}$ denotes the set of subsets of M of cardinality k. The set of all finite subsets of M is then given by $\bigcup_{k\geq 0} \binom{M}{k} =: \binom{M}{<\omega}$.

A *bag* (also called *multiset*) over a set U is an unordered collection of elements of U, possibly with repetitions. In order to distinguish sets and bags, we use double curly braces $\{\!\{\cdots\}\!\}$ when explicitly denoting bags. Similarly to the notation for sets, we let $\left(\!\binom{M}{k}\!\right)$ denote the set of bags over the set M of cardinality $k \in \mathbb{N}$ (that is, containing k elements, counting copies). The set of all finite bags over M is given by $\bigcup_{k\geq 0}\left(\!\binom{M}{k}\!\right) =: \left(\!\binom{M}{<\omega}\!\right)$.

There are multiple equivalent ways to formalize the notion of bags. We introduce two such definitions that we use interchangeably later:

Multiplicity perspective A *bag* B over some set U is a function $\#_B\colon U \to \mathbb{N}$ assigning a *multiplicity* to every element of U. The cardinality of B is $|B| \coloneqq \sum_{u\in U} \#_B(u)$.

Quotient perspective For all $a, b \in U^k$, let $a \sim b$ if b is a permutation of a. A *bag* B of cardinality $|B| = k$ is a $\sim$-equivalence class on U^k.

While the multiplicity perspective better matches the intuitive semantics of bags, the quotient view later has a closer connection to the probability spaces we are going to construct.

2.1 Relational Databases

We follow the general terminology and notions of the *named perspective* of databases, see for example [2]. We fix two countably infinite, disjoint sets **Attributes** and **Relations** of *attribute names* and *relation names*, respectively. As usual, we drop the distinction between names of attributes and relations and their model-theoretic interpretation. A *database schema* is a pair $\mathcal{S} = (\mathcal{A}, \mathcal{R})$ with the following properties:

- $\mathcal{A}$ and $\mathcal{R}$ are finite subsets of **Attributes** resp. **Relations**.
- For every attribute $A \in \mathcal{A}$ there exists a set $\mathsf{dom}_{\mathcal{S}}(A)$, called its *domain*.
- For every relation symbol $R \in \mathcal{R}$ there exists an associated k-tuple of distinct attributes from $\mathcal{A}$ for some k, called its *type* $\mathsf{type}_{\mathcal{S}}(R)$.

Implicitly, every relation $R \in \mathcal{R}$ has an *arity* $\mathsf{ar}_{\mathcal{S}}(R) \coloneqq |\mathsf{type}_{\mathcal{S}}(R)|$ and a *domain* $\mathsf{dom}_{\mathcal{S}}(R) \coloneqq \prod_{A\in\mathsf{type}_{\mathcal{S}}(R)} \mathsf{dom}_{\mathcal{S}}(A)$. Elements of the domain of $R \in \mathcal{R}$ are called *R-tuples.* Whenever a pair $(\mathcal{A}, \mathcal{R})$ is given, we assume that all of the aforementioned mappings are given as well, unless it is specified otherwise. Given a database schema $\mathcal{S} = (\mathcal{A}, \mathcal{R})$ and a relation $R \in \mathcal{R}$, the set of *R-facts* in $\mathcal{S}$ is formally defined as $\mathsf{facts}_{\mathcal{S}}(R) = \{R\} \times \mathsf{dom}_{\mathcal{S}}(R)$. The set of *all* facts of schema $\mathcal{S}$ is given as $\mathsf{facts}_{\mathcal{S}}(\mathcal{R}) \coloneqq \bigcup_{R\in\mathcal{R}} \mathsf{facts}_{\mathcal{S}}(R)$.

As usual, we denote R-facts in the fashion of $R(a_1, \dots, a_k)$ rather than $(R, a_1, \dots, a_k)$. If $U \subseteq \mathsf{dom}_{\mathcal{S}}(R)$ for $R \in \mathcal{R}$, we let $R(U) \coloneqq \{R(u)\colon u \in U\}$. If U is a Cartesian product involving singletons, like for example $U = \{a\} \times V$, we may omit the braces of the singletons and replace crosses with commas so that $R(a, U) = \{R(a, u)\colon u \in U\}$.

Finally, a *database instance* D of schema $\mathcal{S} = (\mathcal{A}, \mathcal{R})$ is a *finite* bag of facts from $\mathsf{facts}_{\mathcal{S}}(\mathcal{R})$, that is, an element of the set $\mathbb{D}_{\mathcal{S}} \coloneqq \left(\!\!\left(\begin{smallmatrix}\mathsf{facts}_{\mathcal{S}}(\mathcal{R})\\ <\omega\end{smallmatrix}\right)\!\!\right)$. We want to emphasize that in particular we allow single facts to appear two or more times within an instance. That is, we use bag semantics in our database instances.

2.2 Topology and Measure Theory

We assume that the reader is familiar with the basic notions of point set topology such as open and closed sets and continuous mappings. For a more detailed introduction to the concepts we refer to standard text books such as [14, 13]. In the following, we concentrate on the background from measure theory. The definitions and statements are based upon [62] and Chapter 1 of [43].

In topological terms, the spaces we use as our attribute domains later on are called Polish spaces - complete, separable metrizable spaces. Such spaces are the default choice for probability theory in a general setting, as they are quite general while still exhibiting the nice behavior of closed intervals of the real line, in particular the ability to approximate points by converging sequences of a countable collection of open sets.

▶ **Example 1** (see [32, ch. 18] and [62, pp. 52 et seqq.]).

- All finite and countably infinite spaces (with the discrete topology) are Polish.
- The spaces $\mathbb{R}$ and $\mathbb{R} \cup \{\pm\infty\}$ are Polish.
- Closed subspaces of Polish spaces are Polish.
- Countable disjoint unions and countable products of Polish spaces are Polish.

These examples already capture the most relevant cases for standard database applications. Nevertheless we stick to the abstract notion of Polish spaces in order to keep the framework as general as possible. When we work with Polish spaces, we will later always assume that we work with a fixed metric on the space (turning it into a complete separable metric space). In particular, we will use the standard notation $B_\varepsilon(x)$ for the *ball of radius* ε around the point x (with respect to said metric).

Let $\mathbb{X}$ be some set. A *σ-algebra* on $\mathbb{X}$ is a family $\mathfrak{X}$ of subsets of $\mathbb{X}$ such that $\mathbb{X} \in \mathfrak{X}$ and $\mathfrak{X}$ is closed under complementation and *countable* unions. If $\mathfrak{G}$ is a family of subsets of $\mathbb{X}$, then the *σ-algebra generated by* $\mathfrak{G}$ is the smallest σ-algebra $\mathfrak{X}$ on $\mathbb{X}$ containing $\mathfrak{G}$. A *measurable space* is a pair $(\mathbb{X}, \mathfrak{X})$ where $\mathbb{X}$ is an arbitrary set and $\mathfrak{X}$ is a σ-algebra on $\mathbb{X}$. Subsets of $\mathbb{X}$ are called *$\mathfrak{X}$-measurable* (or *measurable* if $\mathfrak{X}$ is clear from context) if they belong to $\mathfrak{X}$. A *probability measure* on $\mathbb{X}$ is a countably additive function $P\colon \mathfrak{X} \to [0,1]$ with $P(\emptyset) = 0$ and $P(\mathbb{X}) = 1$. (P being countably additive means $P\big(\bigcup_i \mathcal{X}_i\big) = \sum_i P(\mathcal{X}_i)$ for any sequence $\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2, \dots$ of disjoint measurable sets.) A measurable space equipped with a probability measure is called a probability space. If Ξ is a probability space $(\mathbb{X}, \mathfrak{X}, P)$, we also write $\mathsf{Pr}_{X \sim \Xi}(X \in \mathcal{X}) = P(\mathcal{X})$ or even omit the subscript $X \sim \Xi$, if the underlying probability space is clear from context.

Let $(\mathbb{X}, \mathfrak{X})$ and $(\mathbb{Y}, \mathfrak{Y})$ be measurable spaces. A mapping $\varphi\colon \mathbb{X} \to \mathbb{Y}$ is called $(\mathfrak{X}, \mathfrak{Y})$-*measurable* (or simply *measurable* if the involved σ-algebras are clear from context) if the preimage under φ of every $\mathfrak{Y}$-measurable set is $\mathfrak{X}$-measurable. That is, if

$$\varphi^{-1}(\mathcal{Y}') = \{X \in \mathbb{X}\colon \varphi(X) \in \mathcal{Y}'\} \in \mathfrak{X} \qquad \text{for all } \mathcal{Y}' \in \mathfrak{Y}.$$

▶ **Fact 2** (cf. [43, Lemmas 1.4, 1.7 & 1.10]). *Let $(\mathbb{X}, \mathfrak{X})$, $(\mathbb{Y}, \mathfrak{Y})$, $(\mathbb{Z}, \mathfrak{Z})$ be measurable spaces.*

- *Let $\mathfrak{G}$ generate $\mathfrak{Y}$. If $\varphi\colon \mathbb{X} \to \mathbb{Y}$ satisfies $\varphi^{-1}(\mathcal{G}) \in \mathfrak{X}$ for all $\mathcal{G} \in \mathfrak{G}$, then φ is measurable.*
- *If $\varphi\colon \mathbb{X} \to \mathbb{Y}$ and $\psi\colon \mathbb{Y} \to \mathbb{Z}$ are measurable, then $\psi \circ \varphi\colon \mathbb{X} \to \mathbb{Z}$ is $(\mathfrak{X}, \mathfrak{Z})$-measurable.*
- *If $\mathbb{Y}$ is a metric space and $(\varphi_n)_{n \geq 0}$ is a sequence of measurable functions $\varphi_n\colon \mathbb{X} \to \mathbb{Y}$ with $\lim_{n\to\infty} \varphi_n = \varphi$, then φ is measurable as well.*

If $(\mathbb{X}, \mathfrak{T}_{\mathbb{X}})$ is a topological space, the *Borel σ-algebra* $\mathfrak{Bor}_{\mathbb{X}}$ on $\mathbb{X}$ is the σ-algebra generated by $\mathfrak{T}_{\mathbb{X}}$. Sets in the Borel σ-algebra are also called *Borel*.

▶ **Fact 3** (cf. [43, Lemma 1.5]). *Any continuous function between the topological spaces $(\mathbb{X}, \mathfrak{T}_{\mathbb{X}})$ and $(\mathbb{Y}, \mathfrak{T}_{\mathbb{Y}})$ is $(\mathfrak{Bor}_{\mathbb{X}}, \mathfrak{Bor}_{\mathbb{Y}})$-measurable .*

Two measurable spaces $(\mathbb{X}, \mathfrak{X})$ and $(\mathbb{Y}, \mathfrak{Y})$ are called *isomorphic* if there exists a bijection $\varphi\colon \mathbb{X} \to \mathbb{Y}$ such that both φ and φ^{-1} are measurable. The mapping φ is then called an *isomorphism* between the measurable spaces. If $\mathfrak{X} = \mathfrak{Bor}_{\mathbb{X}}$ and $\mathfrak{Y} = \mathfrak{Bor}_{\mathbb{Y}}$, then φ is called a *Borel isomorphism* and the measurable spaces are called *Borel isomorphic*. Measurable spaces that are isomorphic to some Polish space with its Borel σ-algebra are called *standard Borel spaces.*

If $\mathfrak{X}_i$ is a σ-algebra on X_i for all $i \in I$, the *product σ-algebra* $\bigotimes_{i\in I} \mathfrak{X}_i$ of $(\mathfrak{X}_i)_{i\in I}$ is the σ-algebra on $\prod_{i\in I} \mathbb{X}_i$ that is generated by the sets $\{\pi_j^{-1}(\mathcal{X})\colon \mathcal{X} \in \mathfrak{X}_j\}_{j\in I}$ where π_j is the canonical projection map $\pi_j\colon \prod_{i\in I} \mathbb{X}_i \to \mathbb{X}_j$.

▶ **Fact 4** (cf. [43, Lemma 1.2]). *Let* $(\mathbb{X}_i)_{i\in I}$ *be a* countable *sequence of Polish spaces and let* $\mathfrak{Bor}_i$ *be the Borel* σ*-algebra of* $\mathbb{X}_i$*. Then* $\mathbb{X} = \prod_{i\in I} \mathbb{X}_i$ *is Polish and* $\mathfrak{Bor}_{\mathbb{X}} = \bigotimes_{i\in I} \mathfrak{Bor}_i$*. That is, countable products of standard Borel spaces are standard Borel.*

2.3 (Finite) Point Processes

Point processes are a well-known concept in probability theory that is used to model distributions of a discrete (but unknown or even infinite) number of points in some abstract "state space", say the Euclidean space $\mathbb{R}^n$ [18]. They are used to model a variety of both practical and theoretical problems and appear in a broad field of applications such as, for example, particle physics, ecology, geostatistics, astronomy and tracking [53, 18, 6, 50, 23]. A concrete collection of points that is obtained by a draw from such a distribution model is called a *realization* of the point process. If all realizations are finite, we speak of a finite point process [18]. We proceed to construct a finite point process over a Polish state space, following the classic constructions of [53, 49]. While modern point process theory is much more evolved by casting point processes in the more general framework of random measures [19], the seminal model of [53, 49] suffices for our studies due to our restriction to finite point processes.

Let $(\mathbb{X}, \mathfrak{X})$ be a standard Borel space. Then for every n, the product measurable space $(\mathbb{X}^n, \mathfrak{X}^{\otimes n})$ with $\mathfrak{X}^{\otimes n} := \mathfrak{X} \otimes \cdots \otimes \mathfrak{X}$ (n times) is standard Borel as well (Fact 4). Letting $\sim_n$ denote the equivalence relation on $\mathbb{X}^n$ with $(x_1, \dots, x_n) \sim_n (y_1, \dots, y_n)$ if there exists a permutation π of $\{1, \dots, n\}$ with $(y_1, \dots, y_n) = (x_{\pi(1)}, \dots, x_{\pi(n)})$, then elements of $\mathbb{X}^n / \sim_n$ are basically unordered collections of n (not necessarily different) points, that is, *bags* (or *multisets*). Formally, we identify $\mathbb{X}^n / \sim_n$ with the space $\left(\!\binom{\mathbb{X}}{n}\!\right)$ of all n-element bags from X. The space of all possible realizations is then naturally defined as

$$\left(\!\binom{\mathbb{X}}{<\omega}\!\right) = \bigcup_{n\in\mathbb{N}} \left(\!\binom{\mathbb{X}}{n}\!\right) = \bigcup_{n\in\mathbb{N}} \mathbb{X}^n / \sim_n .$$

This is the canonical sample space for a finite point process [18, 53], but we need to define a σ-algebra on this space. The original construction of [53] considers the *symmetrization* transformation sym from $\mathbb{X}^{<\omega}$ to $\left(\!\binom{\mathbb{X}}{<\omega}\!\right)$ where $\mathsf{sym}(x_1, \dots, x_n) = [(x_1, \dots, x_n)]_{\sim_n} = \{\!\{x_1, \dots, x_n\}\!\}$ and $\mathsf{sym}(\mathcal{X}) = \{\mathsf{sym}(\bar{x}) : \bar{x} \in \mathcal{X}\}$ and defines the σ-algebra on $\mathbb{X}$ to be the set of all subsets of $\left(\!\binom{\mathbb{X}}{<\omega}\!\right)$ whose preimage under sym is measurable with respect to the σ-algebra on $\mathbb{X}^{<\omega}$ that is generated using $(\mathfrak{X}^{\otimes n})_{n\in\mathbb{N}}$ (pursuing the idea to lift probability measures from well-known product spaces to the new, in terms of measure theory inconvenient "bag-space" – note that the construction above indeed yields a σ-algebra on $\left(\!\binom{\mathbb{X}}{<\omega}\!\right)$, see [43, Lemma 1.3]). An equivalent, but technically more convenient construction (see [49]) is motivated by an interpretation of point processes as "random counting measures" [53, 49, 19]: for $\mathcal{X} \in \mathfrak{X}$ and $n \in \mathbb{N}$, the set $\mathcal{C}(\mathcal{X}, n) \subseteq \left(\!\binom{\mathbb{X}}{<\omega}\!\right)$ is the set of bags C over $\mathbb{X}$ with $\#_C(\mathcal{X}) := \sum_{X\in\mathcal{X}} \#_C(X) = n$ (that is, with exactly n "hits" in $\mathcal{X}$) is called the *counting event* of $\mathcal{X}$ and n. We define $\mathfrak{C}_{\mathbb{X}}$ to be the σ-algebra that is generated by the family of counting events $\mathcal{C}(\mathcal{X}, n)$ where $\mathcal{X}$ is Borel in $\mathbb{X}$ and n is a nonnegative integer. The family $\mathfrak{C}_{\mathbb{X}}$ is known as the *counting σ-algebra* on $\left(\!\binom{\mathbb{X}}{<\omega}\!\right)$. It can be shown that the σ-algebra generated by the counting events is the same as the σ-algebra defined from product σ-algebras and the symmetrization operation (see [53, 49]).

▶ **Definition 5** (cf. [49, Def. 1]). *Let* $(\mathbb{X}, \mathfrak{X})$ *be a standard Borel space and let* P *be a probability measure on* $\left(\left(\!\binom{\mathbb{X}}{<\omega}\!\right), \mathfrak{C}_{\mathbb{X}}\right)$*. Then* $\left(\left(\!\binom{\mathbb{X}}{<\omega}\!\right), \mathfrak{C}_{\mathbb{X}}, P\right)$ *is called a* finite point process *with* state space $(\mathbb{X}, \mathfrak{X})$*.*

A finite point process $(\mathbb{Y}, \mathfrak{Y}, P)$ *with state space* $(\mathbb{X}, \mathfrak{X})$ *is called* simple*, if any realization is almost surely a set, i. e. if* $\mathsf{Pr}\left(\#_Y(\{X\}) \in \{0,1\} \text{ for all } X \in \mathbb{X}\right) = 1$.

3 Probabilistic Databases

In [38], we introduced a general notion of infinite probabilistic databases as probability spaces of database instances, that is, probability spaces $(\mathbb{D}, \mathfrak{D}, P)$, where $\mathbb{D} \subseteq \mathbb{D}_{\mathcal{S}}$ for some database schema $\mathcal{S}$. Here $\mathbb{D}$ may be infinite, even uncountable. In fact, in [38] we only considered instances that are sets rather than bags, but this does not make much of a difference here. We left it open, however, how to construct such probability spaces, and in particular how to define a suitable measurable spaces $(\mathbb{D}, \mathfrak{D})$, which is nontrivial for uncountable $\mathbb{D}$. In this section, we provide a general construction for constructing such measurable spaces.

3.1 Probabilistic Databases as Finite Point Processes

Throughout this paper, we only consider database schemas $\mathcal{S}$ where for every attribute A the domain $\mathsf{dom}_{\mathcal{S}}(A)$ is a Polish space. This is no real restriction; all domains one might typically find, such as the sets of integers, reals, or strings over a finite or even countable alphabet have this property.

In the following, we fix a database schema $\mathcal{S} = (\mathcal{A}, \mathcal{R})$. It follows from Fact 4 that not only the domains $\mathsf{dom}_{\mathcal{S}}(A)$ of the attributes $A \in \mathcal{A}$, but also the spaces $\mathsf{dom}_{\mathcal{S}}(R)$ and $\mathsf{facts}_{\mathcal{S}}(R)$ for all $R \in \mathcal{R}$ are Polish. We equip all of these spaces with their respective Borel σ-algebras and note that $\mathsf{dom}_{\mathcal{S}}(R)$ and $\mathsf{facts}_{\mathcal{S}}(R)$ are Borel-isomorphic from the point of view of measurable spaces. Thus, they can be used interchangeably when discussing measurability issues with respect to a single relation. For the set $\mathsf{facts}_{\mathcal{S}}(R)$ of facts *using relation symbol* $R \in \mathcal{R}$, let $\mathfrak{F}_{\mathcal{S}}(R)$ denote its (Borel) σ-algebra. We equip $\mathsf{facts}_{\mathcal{S}}(\mathcal{R})$, the set of *all facts of schema* $\mathcal{S}$ with the σ-algebra

$$\mathfrak{F}_{\mathcal{S}}(\mathcal{R}) = \{F \subseteq \mathsf{facts}_{\mathcal{S}}(\mathcal{R}) \colon F \cap \mathsf{facts}_{\mathcal{S}}(R) \in \mathfrak{F}_{\mathcal{S}}(R) \text{ for all } R \in \mathcal{R}\}.$$

Note that this is indeed a σ-algebra and, moreover, turns $(\mathsf{facts}_{\mathcal{S}}(\mathcal{R}), \mathfrak{F}_{\mathcal{S}}(\mathcal{R}))$ into a standard Borel space (cf. [30, p. 39] and [29, p. 166]).

Now a probabilistic database of schema $\mathcal{S}$ is supposed to be a probability space $(\mathbb{D}, \mathfrak{D}, P)$ where $\mathbb{D} \subseteq \mathbb{D}_{\mathcal{S}}$. Without loss of generality we may assume that actually $\mathbb{D} = \mathbb{D}_{\mathcal{S}} = \left(\!\!\binom{\mathsf{facts}_{\mathcal{S}}(\mathcal{R})}{<\omega}\!\!\right)$, because we can adjust the probability measure to be 0 on instances we are not interested in. Thus a probabilistic database is a probability space over finite sets of facts. This is exactly what a finite point process over the state space consisting of facts is. We still need to define the σ-algebra $\mathfrak{D}$, but the theory of point processes gives us a generic way of doing this: we let $\mathfrak{D}_{\mathcal{S}} = \mathfrak{C}_{\mathsf{facts}_{\mathcal{S}}(\mathcal{R})}$ be the counting σ-algebra of $\mathbb{D}_{\mathcal{S}}$ (cf. Section 2.3).

▶ **Definition 6.** *A* standard probabilistic database *of schema $\mathcal{S}$ is a probability space $(\mathbb{D}_{\mathcal{S}}, \mathfrak{D}_{\mathcal{S}}, P)$.*

That is, a standard probabilistic database of schema $\mathcal{S}$ is a finite point process over the state space $(\mathsf{facts}_{\mathcal{S}}(\mathcal{R}), \mathfrak{F}_{\mathcal{S}})$.

The reason we speak of "standard" PDBs in the definition is to distinguish them from the more general PDBs introduced in [38, Definition 3.1]. In [38], we left the σ-algebra unspecified and only required the (mild) property, that the occurrence of measurable sets of facts is themselves measurable. This requirement corresponds to a set version of the counting events defined above and is thus given by default in a standard probabilistic database.

Even though the construction of counting σ-algebras for point processes is nontrivial, we are convinced that it is a natural generic construction of σ-algebras over spaces of finite (or countable) sets and the extensive usage of these constructions throughout mathematics for more than fifty years now indicates their suitability for such tasks. *Throughout this paper, all probabilistic databases are standard. Therefore, we omit the qualifier "standard" in the following and just speak of probabilistic databases (PDBs).*

We defined instances of PDBs to be bags of facts. However, if a PDB, that is, a finite point process is simple (see Section 2.3), then it may be interpreted as a PDB with set-instances.

▶ **Example 7.** Every finite probabilistic database (as introduced, for example, in [63]) can be viewed as a standard PDB: Let $\tilde{\mathbb{D}}$ be a finite set of set-valued database instances over some schema $\mathcal{S} = (\mathcal{A}, \mathcal{R})$ and let $\tilde{P}\colon \tilde{\mathbb{D}} \to [0,1]$ a probability measure on $\tilde{\mathbb{D}}$ (equipped with the power set as its σ-algebra). Then $(\tilde{\mathbb{D}}, \tilde{P})$ corresponds to the simple finite point process $(\mathbb{D}, \mathfrak{D}, P)$ on the instance measurable space of $\mathcal{S}$ with state space $(\mathsf{facts}_{\mathcal{S}}(\mathcal{R}), \mathfrak{F}_{\mathcal{S}}(\mathcal{R}))$ where $P(\mathcal{D}) = \tilde{P}(\mathcal{D} \cap \tilde{\mathbb{D}})$ (interpreting $\tilde{\mathbb{D}}$ with a (finite) collection of bags with $\{0,1\}$-valued multiplicities).

3.2 The Possible Worlds Semantics of Queries and Views

In the traditional database setting, *views* are mappings from database instances of an *input schema* (or *source schema*) $\mathcal{S} = (\mathcal{A}, \mathcal{R})$ to database instances of some *output schema* (or *target schema*) $\mathcal{S}' = (\mathcal{A}', \mathcal{R}')$. Views, whose output schema $\mathcal{S}'$ consists of a single relational symbol only are called *queries*. Queries and views are usually given by syntactic expressions in some *query language*. As it is common, we will blur the distinction between a query (or view) and its syntactic representation.

Let $\Delta = (\mathbb{D}_{\mathcal{S}}, \mathfrak{D}_{\mathcal{S}}, P)$ be a probabilistic database of schema $\mathcal{S} = (\mathcal{A}, \mathcal{R})$ and let V be a view of input schema $\mathcal{S}$ and output schema $\mathcal{S}' = (\mathcal{A}', \mathcal{R}')$. The image of a set $\mathcal{D} \subseteq \mathfrak{D}$ of instances is $V(\mathcal{D}) = \{V(D)\colon D \in \mathcal{D}\} \subseteq \mathbb{D}_{\mathcal{S}'}$.

Now we would like to define a probability measure on the output space $(\mathbb{D}_{\mathcal{S}'}, \mathfrak{D}_{\mathcal{S}'})$ by

$$P'(\mathcal{D}') := P\big(V^{-1}(\mathcal{D}')\big) = P\big(\{D \in \mathbb{D}\colon V(D) \in \mathcal{D}'\}\big) \tag{1}$$

for all $\mathcal{D}' \in \mathfrak{D}_{\mathcal{S}'}$. Then V would map Δ to $\Delta' := (\mathbb{D}_{\mathcal{S}'}, \mathfrak{D}_{\mathcal{S}'}, P')$. This semantics of views over PDBs is known as the *possible worlds semantics* of probabilistic databases [36, 3, 63, 65].

However, P' (as defined in (1)) is only well-defined if for all $\mathcal{D}' \in \mathfrak{D}_{\mathcal{S}'}$ the set $V^{-1}(\mathcal{D}')$ is in $\mathfrak{D}_{\mathcal{S}}$, that is, if V is a *measurable* mapping from $(\mathbb{D}_{\mathcal{S}}, \mathfrak{D}_{\mathcal{S}})$ to $(\mathbb{D}_{\mathcal{S}'}, \mathfrak{D}_{\mathcal{S}'})$.

Measurability is not just a formality, but an issues that requires attention. The following example shows that there are relatively simple "queries" that are not measurable.

▶ **Example 8.** Let $\mathcal{S} = \mathcal{S}'$ be the schema consisting of a singe unary relation symbol R with attribute domian $\mathbb{R}$ (equipped with the Borel σ-algebra), and let B be some Borel set in $\mathbb{R}^2$.

We define a mapping $Q_B\colon \mathbb{D}_{\mathcal{S}} \to \mathbb{D}_{\mathcal{S}}$, our "query", by

$$Q_B(D) := \begin{cases} D & \text{if } D \text{ is a singleton } \{\!\{R(x)\}\!\} \text{ and there exists } \ y \in \mathbb{R} \text{ s. t. } (x,y) \in B, \\ \emptyset & \text{otherwise.} \end{cases}$$

Observe that $Q_B^{-1}(\mathbb{D}_{\mathcal{S}}) = \{\{\!\{R(x)\}\!\}\colon x \in \mathsf{proj}_1(B)\}$, where $\mathsf{proj}_1(B) = \{x \in \mathbb{R}\colon \text{there is } y \in \mathbb{R} \text{ s. t. } (x,y) \in B\}$. It is a well known fact that there are Borel sets $B \subseteq \mathbb{R}^2$ such that the projection $\mathsf{proj}_1(B)$ is *not* a Borel set in $\mathbb{R}$ (see [62, Theorem 4.1.5]). For such sets B, the query Q_B is not measurable.

The rest of this paper is devoted to proving that queries and views expressed in standard query languages, specifically relational algebra, possibly extended by aggregation, and Datalog queries, are measurable mappings and thus have a well-defined open-world semantics over probabilistic databases.

It will be sufficient to focus on *queries*, because views can be composed from queries and the measurability results can be lifted (as we formally show in the next subsection). *Throughout the rest of the paper, we adopt the following notational conventions: queries are denoted by Q and map a PDB $\Delta = (\mathbb{D}, \mathfrak{D}, P)$ to a PDB $\Delta' = (\mathbb{D}', \mathfrak{D}', P')$ such that Δ is of schema $\mathcal{S}$ and Δ' is of schema $\mathcal{S}'$.*

▶ Observation 9. The task of establishing measurability of queries in our framework is simplified by the following.

1. If we want to demonstrate the measurability of Q, it suffices to show that $Q^{-1}(\mathcal{D}') \in \mathfrak{D}$ for all *counting events* $\mathcal{D}' = \mathcal{C}'(F, n)$ of $(\mathbb{D}', \mathfrak{D}')$. This is due to Fact 2 because they generate $\mathfrak{D}'$.
2. Since compositions of measurable mappings are measurable (again from Fact 2), composite queries are immediately measurable if all their components are measurable queries to begin with. In particular, we can demonstrate the measurability of general queries of some query language by structural induction.

▶ Remark 10. Let us again mention something related to the well-established knowledge on point processes. The mappings (queries) we investigate map between point processes that are defined on different measure spaces that are themselves a conglomerate of simpler measure spaces of different shape. It is well-known that measurable transformations of the state space of a point process define a new point process on the transformed state space (a strengthening of this result is commonly referred to as the "mapping theorem" [48]). Our queries however are in general already defined on point configurations and not on the state space of facts. Thus, their measurability can in general not be obtained by the idea just sketched.

3.3 Assembling Views from Queries

We think of views as finite sets of queries, including one for every relation of the output schema. Suppose $V = \{Q_1, \dots, Q_k\}$ is a view consisting of measurable queries $Q_1, \dots, Q_k$ where the names of the target relations of the Q_i are mutually distinct. The target schema $\mathcal{S}'$ of V is given by the union of the target schemas of Vs individual queries. Now every fact $f \in \mathsf{facts}_{\mathcal{S}'}(\mathcal{R}')$ of the new schema originates from the target schema of exactly one of the queries $Q_1, \dots, Q_k$. We refer to that query as Q_f. Then for all $D \in \mathbb{D}$ and $f \in \mathsf{facts}_{\mathcal{S}'}(\mathcal{R}')$, we define $\#_{V(D)}(f) := \#_{Q_f(D)}(f)$. Now if $F \subseteq \mathsf{facts}_{\mathcal{S}'}(\mathcal{R}')$, let $F_i := F \cap \mathsf{facts}_{\mathcal{S}_i'}(\mathcal{R}_i')$ where $\mathcal{S}_i' = (\mathcal{A}_i', \mathcal{R}_i')$ is the target schema of Q_i. Then

$$\#_{V(D)}(F) = n \quad \Leftrightarrow \quad \text{there are } n_1, \dots, n_k \text{ with } \textstyle\sum_{i=1}^{k} n_i = n \text{ such that } \#_{Q_i(D)}(F_i) = n_i.$$

Since the F_i are measurable if and only if F is measurable, the above describes a countable union of measurable sets. Thus, V is measurable.

4 Relational Algebra

As motivated in Section 3.2, we now investigate the measurability of relational algebra queries in our model. The concrete relational algebra for bags that we use here is basically the (unnested version of the) algebra that was introduced in [21] and investigated respectively extended and surveyed in [5, 40, 39]. It is called BALG^1 (with superscript 1) in [40]. We do not introduce nesting as it would yield yet another layer of abstraction and complexity to the spaces we investigate, although by the properties that such spaces exhibit, we have strong reason to believe that there is no technical obstruction in allowing spaces of finite bags as attribute domains.

The operations we consider are shown in the Table 1 below. As seen in [5, 40, 39], there is some redundancy within this set of operations that will be addressed later. A particular motivation for choosing this particular algebra is that possible worlds semantics are usually built on top of set semantics and these operations naturally extend the common behavior of relation algebra queries to bags. This is quite similar to the original motivation of [21] and [5] regarding their choice of operations.

Table 1 BALG^1-operators considered in this paper.

Base Queries	Constructors	$Q = \{\!\!\{\}\!\!\}$ and $Q = \{\!\!\{R(a)\}\!\!\}$
	Extractors	$Q = R$
	Renaming	$Q = \varrho_{A\to B}(R)$
Basic Bag Operations	Additive Union	$Q = R_1 \uplus R_2$
	Difference	$Q = R_1 - R_2$
	Max-Union	$Q = R_1 \cup R_2$
	(Min-)Intersection	$Q = R_1 \cap R_2$
	Deduplication	$Q = \delta(R)$
SPJ-Operations	Selection	$Q = \sigma_{(A_1,\dots,A_k)\in B}(R)$
	Projection	$Q = \pi_{(A_1,\dots,A_k)}(R)$
	Cross Product	$Q = R_1 \times R_2$

The main result we establish in this section is the following theorem:

▶ **Theorem 11.** *All queries expressible in the bag algebra* BALG^1 *are measurable.*

Since compositions of measurable mappings are measurable, the measurability of the operators from Table 1 directly entails the measurability of compound queries by structural induction.

First note that the measurability of the base queries is easy to prove.

▶ **Lemma 12.** *The queries $\{\!\!\{\}\!\!\}$, $\{\!\!\{R(a)\}\!\!\}$ and R are measurable.*

Proof. First consider $Q = \{\!\!\{\}\!\!\}$ and fix some $\mathcal{D}' \in \mathfrak{D}'$. If $\{\!\!\{\}\!\!\} \in \mathcal{D}'$, then $Q^{-1}(\mathcal{D}') = \mathbb{D} \in \mathfrak{D}$. Otherwise, $Q^{-1}(\mathcal{D}') = \emptyset \in \mathfrak{D}$. Thus, Q is measurable. The same argument applies to $Q = \{\!\!\{R(a)\}\!\!\}$.

Now consider the query $Q = R$ and let $\mathcal{C}'(F, n)$ be a counting event in the output measurable space. Then for every instance $D \in \mathbb{D}$, $\#_{Q(D)}(F) = n$ if and only if $\#_D(F) = n$ Thus, $Q^{-1}(\mathcal{C}'(F, n))$ is the counting event $\mathcal{C}(F, n)$ in $(\mathbb{D}, \mathfrak{D})$. Hence, Q is measurable. ◀

4.1 Basic Bag Operations

We will obtain the measurability of the basic bag operations $\uplus$, $-$, $\cap$, $\cup$, δ as a consequence of the following, more general result that gives some additional insight into properties that make queries measurable.

Consider a query Q of input schema $\mathcal{S}$ and output schema $\mathcal{S}'$ operating on relations R_1 and R_2 of $\mathcal{S}$. Let R' be the single (output) relation of $\mathcal{S}'$.

▶ **Lemma 13.** *Suppose that given Q there exist functions $q_1 \colon \mathsf{facts}_{\mathcal{S}'}(R') \to \mathsf{facts}_{\mathcal{S}}(R_1)$ and $q_2 \colon \mathsf{facts}_{\mathcal{S}'}(R') \to \mathsf{facts}_{\mathcal{S}}(R_2)$ with the following properties:*

1. *for all $n \in \mathbb{N}$ there exists a set $M(n) \subseteq \mathbb{N}^2$ with $(0,0) \notin M(n)$ for $n > 0$ such that for all $D \in \mathbb{D}$ and all $f \in \mathsf{facts}_{\mathcal{S}'}(R')$ it holds that*

$$\#_{Q(D)}(f) = n \qquad \textit{if and only if} \qquad \big(\#_D(q_1(f)), \#_D(q_2(f))\big) \in M(n);$$

2. *both q_1 and q_2 are injective and continuous;*
3. *the images of F under q_1 and q_2 are measurable: $q_1(F) \in \mathfrak{F}_{\mathcal{S}}(R_1)$ and $q_2(F) \in \mathfrak{F}_{\mathcal{S}}(R_2)$.*

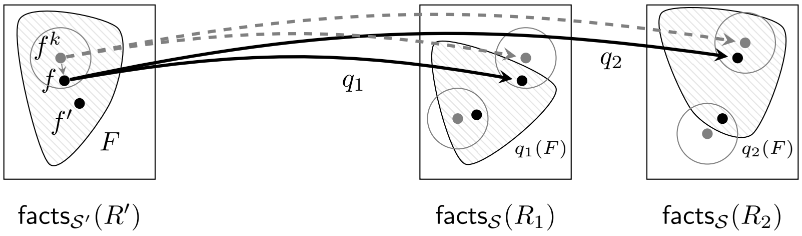
Then Q is measurable.

Let us briefly mention the impact of the various preconditions of the lemma before turning to its proof. The existence of the functions q_1 and q_2 ensures that preimages of counting events $\mathcal{C}'(F, n)$ under the query Q can be approximated by using the fact that our state spaces are Polish. They "decompose" the set F of facts into disjoint (and measurable!) sets of facts for the preimage in a continuous, invertible way that exactly captures how tuples in the preimage relate to tuples in the image.

Proof (Lemma 13). Assume that q_1 and q_2 exist with properties 1 to 3. We fix $F \in \mathfrak{F}_{\mathcal{S}'}(R')$ and $n \in \mathbb{N}_+$ and show that $Q^{-1}(\mathcal{C}'(F, n))$ is in $\mathfrak{D}$. Let F_0 be a countable, dense set in $\mathsf{facts}_{\mathcal{S}'}(R')$. We claim that $\#_{Q(D)}(F) = n$ if and only if

$$
\begin{aligned}
&\text{there exist } \ell \in \mathbb{N}_+ \text{ and } n_1, \dots, n_\ell \in \mathbb{N} \text{ with } \textstyle\sum_{i=1}^{\ell} n_i = n \text{ and}\\
&\quad\text{there exist } (n_{i,1}, n_{i,2}) \in M(n_i) \text{ and } k_0 \in \mathbb{N}_+ \text{ and}\\
&\quad\quad\text{there exist Cauchy sequences } (f_1^k)_{k\in\mathbb{N}}, \dots, (f_\ell^k)_{k\in\mathbb{N}} \text{ in } F_0 \text{ with}\\
&\quad\quad B_{1/k_0}(f_i^k) \cap B_{1/k_0}(f_{i'}^{k'}) = \emptyset \text{ for all } k, k' \text{ and } i \neq i' \text{ such that for all } k > k_0 \qquad (*)\\
&\quad\quad\quad \#_D(q_1(F) \cap B_{1/k}(q_1(f_i^k))) = n_{i,1} \text{ and } \#_D(q_2(F) \cap B_{1/k}(q_2(f_i^k))) = n_{i,2}\\
&\quad\quad\quad\quad \text{for } 1 \leq i \leq \ell \text{ and}\\
&\quad\quad\quad \#_D(q_1(F) \setminus \textstyle\bigcup_{i=1}^{\ell} B_{1/k}(q_1(f_i^k))) = 0 \text{ and } \#_D(q_2(F) \setminus \bigcup_{i=1}^{\ell} B_{1/k}(q_2(f_i^k))) = 0.
\end{aligned}
$$

Note that $(*)$ is a countable combination of counting events in $(\mathbb{D}, \mathfrak{D})$ (using condition 3, in particular). Thus, to show the measurability of Q it suffices to show the equivalence of $\#_{Q(D)}(F) = n$ and $(*)$.



f^k
f
f'
F
q_1
q_2
$q_1(F)$
$q_2(F)$
$\mathsf{facts}_{\mathcal{S}'}(R')$
$\mathsf{facts}_{\mathcal{S}}(R_1)$
$\mathsf{facts}_{\mathcal{S}}(R_2)$

Figure 1 Example illustration of $(*)$ for two facts f and f'. Both these facts are approximated by Cauchy sequences that under q_1 and q_2 also approximate their images.

Assume $\#_{Q(D)}(F) = n$. Let $f_1, \dots, f_\ell$ be the facts from F with the property that $\#_D(q_1(f)) > 0$ or $\#_D(q_2(f)) > 0$.

Let $n_i := \#_{Q(D)}(f_i)$. From condition 1 we know that $(\#_D(q_1(f_i)), \#_D(q_2(f_i))) \in M(n_i)$ as well as $\sum_{i=1}^{\ell} n_i = n$. Let $(f_1^k), \dots, (f_\ell^k)$ be Cauchy sequences from F_0 that converge to $f_1, \dots, f_\ell$. Since ℓ is finite, the balls around f_i^k and $f_{i'}^k$ do not intersect for sufficiently large k as well as the balls around their images under q_1 respectively q_2 (since both of them are injective and continuous). Thus, $\#_D(q_1(F) \cap B_{1/k}(q_1(f_i^k))) = \#_D(q_1(f_i))$ and $\#_D(q_2(F) \cap B_{1/k}(q_2(f_i^k))) = \#_D(q_2(f_i))$ for sufficiently large k. Therefore, D satisfies $(*)$.

Now for the other direction, suppose D satisfies $(*)$. As the f_i^k are Cauchy sequences, the spaces $\mathsf{facts}_{\mathcal{S}'}(R_j)$ are Polish and hence complete, and the q_j are continuous there exists (for every $1 \leq i \leq \ell$) some $f_i \in F$ such that $f_i^k \to f_i$, $q_1(f_i^k) \to q_1(f_i)$ and $q_2(f_i^k) \to q_2(f_i)$ as

$k \to \infty$ and $(\#_D(q_1(f_i)), \#_D(q_2(f_i))) = (n_{i,1}, n_{i,2}) \in M(n_i)$. By condition 1, $Q(D)$ contains f_i with multiplicity n_i and as $\sum_{i=1}^{\ell} n_i = n$ (and since D had no other facts with positive multiplicity than the above), it follows that $\#_{Q(D)}(F) = n$. ◀

Note that the result above easily generalizes to queries that depend on an arbitrary number of relations of the input probabilistic database. Lemma 13 provides a criterion to establish the measurability of queries. Checking its precondition for bag operations we consider turns out to be quite easy and yields the following lemma.

▶ **Lemma 14.** *The following queries are measurable:*

1. (Additive Union) $Q = R_1 \uplus R_2$ *with* $R_1, R_2 \in \mathcal{R}$ *of equal type.*
2. (Difference) $Q = R_1 - R_2$ *with* $R_1, R_2 \in \mathcal{R}$ *of equal type.*
3. ((Min-)Intersection) $Q = R_1 \cap R_2$ *with* $R_1, R_2 \in \mathcal{R}$ *of equal type.*
4. (Max-Union) $Q = R_1 \cup R_2$ *with* $R_1, R_2 \in \mathcal{R}$ *of equal type.*
5. (Deduplication) $Q = \delta(R)$ *with* $R \in \mathcal{R}$.

Proof. As $\cup$ and $\cap$ are expressible via $\uplus$ and $-$ (cf. [5]), we only show Statements 1, 2 and 5.

1. Define q_1 and q_2 by $q_i(R(x)) = R_i(x)$. Then q_i, $i \in \{1, 2\}$ is injective and continuous and $q_i(F) = F_i \in \mathfrak{F}_{\mathcal{S}}(R_i)$. Now let $k \in \mathbb{N}$ and let $M(k) \subseteq \mathbb{N}^2$ be the set of pairs (k_1, k_2) with the property that $k_1 + k_2 = k$. Then $\#_{Q(D)}(f) = k$ if and only if $\big(\#_D\big(q_1(f)\big), \#_D\big(q_2(f)\big)\big) \in M(k)$. Together, by Lemma 13, Q is measurable.
2. This works exactly like in the case of $\uplus$ with $M(k)$ being the set of pairs (k_1, k_2) with $\mathsf{max}(k_1 - k_2, 0) = k$.
5. In this case, we only use a single function q that maps $R'(x)$ to $R(x)$. Again, q is obviously both continuous and injective and $q(F) \in \mathfrak{F}_{\mathcal{S}}(R)$ for every measurable F. If $k = 1$, we let $M(k) = \mathbb{N} \setminus \{0\}$ and $M(k) = \{0\}$ otherwise. Then clearly $\#_{Q(D)}(R(x)) = k$ if and only if $\#_D(q(R(x)) \in M(k)$ and again, Q is measurable by Lemma 13. ◀

4.2 Selection, Projection and Join

In this section, we investigate selection and projection as well as the cross product of two relations. We start with the following helpful lemma that allows us to restructure our relations into a more convenient shape to work with. Semantically, it might be seen as a special case of a projection query.

▶ **Lemma 15.** *Reordering attributes within the type of a relation yields a measurable query.*

Proof. Recall that any permutation can be expressed as a composition of transpositions. Thus, we only consider the case where two attributes, say A and B, switch places within the type of some relation $R \in \mathcal{R}$. Let q be the function that maps $\mathsf{facts}_{\mathcal{S}}(R)$ to $\mathsf{facts}_{\mathcal{S}'}(R')$ by swapping the entries for attribute A and B. Obviously, under q, the preimage of a measurable rectangle in $\mathfrak{F}_{\mathcal{S}'}(R)$ is a measurable rectangle itself. As $\#_{Q(D)}(F) = n$ if and only if $\#_D(q^{-1}(F)) = n$, Q is measurable. ◀

▶ **Lemma 16.** *The query* $Q = \sigma_{(A_1,\dots,A_k)\in\mathcal{B}}(R)$ *is measurable for all* $R \in \mathcal{R}$*, all pairwise distinct attributes* $A_1, \dots, A_k \in \mathsf{type}_{\mathcal{S}}(R)$ *and all Borel subsets* $\mathcal{B}$ *of* $\prod_{i=1}^{k} \mathsf{dom}_{\mathcal{S}}(A_i)$.

Proof. Fix some $F \in \mathfrak{F}_{\mathcal{S}'}(R')$ and $n \in \mathbb{N}$. By Lemma 15, we may assume that $\mathsf{type}_{\mathcal{S}}(R) = (A_1, \dots, A_m)$ where $m \geq k$. Let $F_{\mathcal{B}} \coloneqq \{R\} \times \mathcal{B} \times \mathsf{dom}_{\mathcal{S}}(A_{k+1}) \times \dots \times \mathsf{dom}_{\mathcal{S}}(A_m)$. Note that $F_{\mathcal{B}} \in \mathfrak{F}_{\mathcal{S}}(R)$. (This is a consequence of Fact 4.) As $\#_{Q(D)}(F) = n$ if and only if $n = \#_{Q(D)}(F \cap F_{\mathcal{B}}) = \#_D(F \cap F_{\mathcal{B}})$, Q is measurable. ◀

▶ **Example 17.** Assume that $\mathsf{dom}_{\mathcal{S}}(A) = \mathsf{dom}_{\mathcal{S}}(B) = \mathbb{R}$ and both A and B appear in the type of $R \in \mathcal{R}$. It is well-known (and can be shown by standard arguments) that the sets $\mathcal{B}_{=} := \{(x,y) \in \mathbb{R}^2 \colon x = y\}$ and $\mathcal{B}_{<} := \{(x,y) \in \mathbb{R}^2 \colon x < y\}$ are Borel in $\mathbb{R}^2$. Thus $\sigma_{A=B}(R) := \sigma_{(A,B)\in\mathcal{B}_=}(R)$ and $\sigma_{A<B}(R) := \sigma_{(A,B)\in\mathcal{B}_<}(R)$ are measurable by Lemma 16.

▶ **Lemma 18.** . *The query $Q = \pi_{A_1,\dots,A_k}(R)$ is measurable for all $R \in \mathcal{R}$ and all mutual distinct $A_1,\dots,A_k \in \mathsf{type}_{\mathcal{S}}(R)$.*

Proof. Again, fix some $F \in \mathfrak{F}_{\mathcal{S}'}(R')$ and $n \in \mathbb{N}$. Note that F is of the shape $\{R'\} \times \mathcal{B}$ where $\mathcal{B}$ is Borel in $\mathsf{dom}_{\mathcal{S}'}(R') = \prod_{i=1}^{k} \mathsf{dom}_{\mathcal{S}}(A_k)$. By Lemma 15, we may again assume that $\mathsf{type}_{\mathcal{S}}(R) = (A_1,\dots,A_m)$ with $m \geq k$. Define $F_{\mathcal{B}}$ exactly like in the proof of Lemma 16: $F_{\mathcal{B}} := \{R\} \times \mathcal{B} \times \mathsf{dom}_{\mathcal{S}}(A_{k+1}) \times \cdots \times \mathsf{dom}_{\mathcal{S}}(A_m)$. Again, $F_{\mathcal{B}} \in \mathfrak{F}_{\mathcal{S}}(R)$. Now, we have $\#_{Q(D)}(F) = n$ if and only if $\#_D(F_{\mathcal{B}}) = n$ and hence, Q is measurable. ◀

▶ **Lemma 19.** *The query $Q = R_1 \times R_2$ is measurable for all $R_1, R_2 \in \mathcal{R}$.*

First we note that this turns out to be more involved than it seems on first sight. The straight-forward approach would be to take a counting event $\mathcal{C}(F,n)$ in the output measurable space and to decompose F into its "left and right parts" $F_1 \subseteq \mathsf{facts}_{\mathcal{S}}(R_1)$ and $F_2 \subseteq \mathsf{facts}_{\mathcal{S}}(R_2)$ such that the instances from the preimage of the query are exactly those with $\#_D(F_1) = n_1$ and $\#_D(F_2) = n_2$ such that $n_1 \cdot n_2 = n$, similar to the setting of Lemma 13. This approach does not settle the case since the sets F_1 and F_2 need not be measurable in general (see [62, Theorem 4.1.5]; we used the same argument in Example 8) which in particular violates the second precondition of Lemma 13.

Proof Sketch. Using renaming, we may assume that the types of R_1 and R_2 are disjoint in terms of attribute names. Consider $F \in \mathfrak{F}_{\mathcal{S}'}(R')$ and $n \in \mathbb{N}$. If F is a measurable rectangle $F = F_1 \times F_2$, it is easy to see that the naïve approach sketched above works via $\#_{Q(D)}(F) = n$ if and only if $\#_D(F_1) \cdot \#_D(F_2) = n$.

In the general case of F being an arbitrary Borel set, we consider the *k-coarse* preimage of $\mathcal{C}'(F,n)$ first. These are the database instances from $\mathbb{D}$ whose minimal inter-tuple distance is at least $\frac{1}{k}$ for some fixed Polish metrics. One can show that these k-coarse preimages of the query are measurable for all F, n and k. As the union of these preimages over all positive integers k is exactly the preimage of $\mathcal{C}'(F,n)$, Q is measurable. The details of this proof can be found in the full version of the paper [37]. ◀

Altogether, within the last three sections, we have established the measurability of all the (bag) relational algebra operators from Table 1 and thus have proven Theorem 11. Of course any additional operator that is expressible by a combination of operations from Table 1 is immediately measurable as well, including for example *natural joins* $Q = R_1 \bowtie R_2$ or selections where the selection predicate is a Boolean combinations of predicates of the shape $(A_1,\dots,A_k) \in \mathcal{B}$.

5 Aggregate Queries

In this section, we study various kinds of aggregate operators. Let U and V be standard Borel spaces. An *aggregate operator* (or *aggregator*) from U to V is a mapping Φ that sends bags of elements of U to elements of V: $\Phi\colon \left(\!\!\binom{U}{<\omega}\!\!\right) \to V$. Every such aggregator Φ gives rise to a query $Q = \varpi_{\Phi}(R)$ defined by $Q(D) := \{\!\{R'(v)\}\!\}$ for $v := \Phi(\{\!\{u \colon R(u) \in D\}\!\})$. (The notation we use for aggregation queries is loosely based on that of [28].) Observe that for every instance D, $\#_{Q(D)}\big(R'(v)\big) = 1$ if and only if $\Phi(\{\!\{u \colon R(u) \in D\}\!\}) = v$ (and 0 otherwise). It is easy to see that $Q = \varpi_\Phi(R)$ is a measurable query whenever Φ is measurable w. r. t. the counting σ-algebra on $\left(\!\!\left({U \atop <\omega}\right)\!\!\right)$: we have $\#_{Q(D)}(F) = 1$ if and only if $D \in \{R\} \times \Phi^{-1}(\{v\colon R'(v) \in F\})$ (and $\#_{Q(D)}(F) = 0$ otherwise).

▶ **Example 20.** The following are the most common aggregate operators:

- (Count) $\textsf{CNT}(\{\!\!\{a_1, \dots, a_n\}\!\!\}) = n$ and $\textsf{CNTd}(\{\!\!\{a_1, \dots, a_n\}\!\!\}) = |\{a_1, \dots, a_n\}|$.
- (Sum) $\textsf{SUM}(\{\!\!\{a_1, \dots, a_n\}\!\!\}) = a_1 + \cdots + a_n$ where a_i are (for instance) real numbers.
- (Minimum / Maximum) $\textsf{MIN}(\{\!\!\{a_1, \dots, a_n\}\!\!\}) = \mathsf{min}\{a_1, \dots, a_n\}$ and $\textsf{MAX}(\{\!\!\{a_1, \dots, a_n\}\!\!\}) = \mathsf{max}\{a_1, \dots, a_n\}$ for ordered domains.
- (Average) $\textsf{AVG}(\{\!\!\{a_1, \dots, a_n\}\!\!\}) = \frac{1}{n}\big(a_1 + \cdots + a_n\big)$ where the a_i might again be real numbers.

Note that $\varpi_{\textsf{CNT}}$ and $\varpi_{\textsf{CNTd}}$ are trivially measurable within our framework by the usage of the counting σ-algebra (and the measurability of deduplication for **CNTd**).

▶ **Lemma 21.** *For all $m \in \mathbb{N}$, let $\varphi_m\colon U^m \to V$ be a* symmetric *function, i. e., $\varphi_m(u) = \varphi_m(u')$ for all $u \in U^m$ and all permutations u' of u. If φ_m is measurable for all m, then $\Phi\colon \left(\!\!\left({U \atop <\omega}\right)\!\!\right) \to V$ defined via $\Phi(\{\!\!\{u_1, \dots, u_m\}\!\!\}) \coloneqq \varphi_m(u_1, \dots, u_m)$ is measurable w. r. t. the counting σ-algebra on $\left(\!\!\left({U \atop <\omega}\right)\!\!\right)$.*

Proof. It suffices to show that the restriction Φ_m of Φ to $\left(\!\!\left({U \atop m}\right)\!\!\right)$ is measurable for all $m \in \mathbb{N}$. If $\mathcal{V}$ is Borel in V, then $\varphi_m^{-1}(\mathcal{V})$ is Borel in U^m as φ_m is measurable. Moreover, since φ_m is symmetric, $\varphi_m^{-1}(\mathcal{V})$ is a symmetric set (i. e. if $\bar{u} \in \varphi_m^{-1}(\mathcal{V})$, then every permutation of u is in $\varphi_m^{-1}(\mathcal{V})$ as well). But then $\Phi_m^{-1}(\mathcal{V})$ is measurable since there is a one-to-one correspondence between the measurable sets of $\left(\!\!\left({U \atop m}\right)\!\!\right)$ and the symmetric Borel sets of U^m [49, Theorem 1]. ◀

As an example application of this lemma we note that all the mappings Φ that were introduced in Example 20 are measurable – the related mappings φ_m of Lemma 21 are all continuous and thus measurable in all of the cases.

A concept closely tied to aggregation is *grouping*. Suppose we want to group a relation R by its attributes $A_1, \dots, A_k$ and perform the aggregation only over the values of attribute A, and separately for every distinct $(A_1, \dots, A_k)$-entry in R. Without loss of generality, we assume that the type of R is $A_1 \times \cdots \times A_k \times A$. We define a query $Q = \varpi_{A_1, \dots, A_k, \Phi(A)}(R)$ by

$$Q(D) = \{\!\!\{R'(\bar{u}, v)\colon R(\bar{u}) \in \pi_{A_1, \dots, A_k}(R(D)) \text{ and } v = \Phi(\{\!\!\{u\colon R(\bar{u}, u) \in D\}\!\!\})\}\!\!\}.$$

▶ **Lemma 22.** *Let $\mathsf{type}_{\mathcal{S}}(R) = A_1 \times \cdots \times A_k \times A$ and $U = \mathsf{dom}_{\mathcal{S}}(A)$. If $\Phi\colon \left(\!\!\left({U \atop <\omega}\right)\!\!\right) \to V$ is measurable (with U and V standard Borel), then $\varpi_{A_1, \dots, A_k, \Phi(A)}(R)$ is a measurable query.*

Proof. Let $Q = \varpi_{A_1, \dots, A_k, \Phi(A)}(R)$ and $\bar{A} = (A_1, \dots, A_k)$. Observe that for every tuple $x_1, \dots, x_n, \varepsilon$ with $x_i \in \prod_{j=1}^k \mathsf{dom}_{\mathcal{S}}(A_j)$ and $\varepsilon > 0$, the following query is a composition of measurable queries and thus measurable itself:

$$\tilde{Q}_{(x_1, \dots, x_n, \varepsilon)} = \bigcup_{i=1}^n \pi_{\bar{A}}\big(\sigma_{\bar{A} \in B_\varepsilon(x_i)}(R)\big) \times \varpi_\Phi\big(\pi_A\big(\sigma_{\bar{A} \in B_\varepsilon(x_i)}(R)\big)\big).$$

We have $\#_{Q(D)}(F) = n$ if and only if there exist pairwise distinct $f_1, \dots, f_n \in F$ such that $Q(D)$ has 1 hit in each of the f_i and nowhere else in F. Having D fixed, every f_i determines the value of the $(A_1, \dots, A_k)$-part of an R-fact in D. Call this tuple y_i. We can fix a countable sequence of $(n + 1)$-tuples $(x_1, \dots, x_n, \varepsilon)$ such that (1) all x_i are from a countable dense set in $\prod_{j=1}^k \mathsf{dom}_{\mathcal{S}}(A_j)$, (2) $d(x_i, y_i) < \varepsilon$ for some fixed Polish metric, and, (3) $\varepsilon \to 0$. Then Q is the (pointwise) limit of the $\tilde{Q}_{(x_1, \dots, x_n, \varepsilon)}$ and, as such, Q is measurable. ◀

As noted before, the aggregates of Example 20 easily satisfy the precondition of Lemma 22.

▶ **Corollary 23.** *The query $\varpi_{A_1, \dots, A_k, \Phi}(R)$ with $A_1, \dots, A_k \in \mathsf{type}_{\mathcal{S}}(R)$ is measurable for all aggregates $\Phi \in \{\textsf{CNT}, \textsf{CNTd}, \textsf{SUM}, \textsf{MIN}, \textsf{MAX}, \textsf{AVG}\}$.*

6 Datalog Queries

In this section, we want to show that our measurability results extend to Datalog queries and in fact all types of queries with operators based on countable iterative (or inductive, inflationary, fixed-point) processes. We will not introduce Datalog or any of the related query languages. The details in the definitions do not matter when it comes to measurability of the queries. Here, we only consider set PDBs and queries with a set (rather than bag) semantics. The key observation is the following lemma.

▶ **Lemma 24.** *Let Q_i, for $i \in \mathbb{N}_+$, be a countable family of measurable queries of the same schema such that $Q = \bigcup_{i\geq 1} Q_i$, defined by $Q(D) := \bigcup_{i\geq 0} Q_i(D)$ for every instance D, is a well-defined query (that is, $Q(D)$ is finite for every D). Then Q is measurable.*

Proof. For every $n \in \mathbb{N}_+$, let $Q^{(n)} := \bigcup_{i=1}^{n} Q_i$. As a finite union of measurable queries, $Q^{(n)}$ is measurable. Since $Q = \mathsf{lim}_{n\to\infty} Q^{(n)}$, the measurability of Q follows. ◀

As every Datalog query can be written as a countable union of conjunctive queries, we obtain the following corollary.

▶ **Corollary 25.** *Every Datalog query is measurable.*

The same is true for queries in languages like inflationary Datalog or least fixed-point logic. For partial Datalog / fixed-point logic, we cannot directly use Lemma 24, but a slightly more complicated argument still based on countable limits works there as well.

7 Beyond Possible Worlds Semantics

In the literature on probabilistic databases, and motivated by real world application scenarios, also other kinds of queries have been investigated that have no intuitive description in the possible worlds semantics framework. A range of such queries is surveyed in [3, 67]. The reason for the poor integration into possible worlds semantics is because such queries lack a sensible interpretation on single instances that could be lifted to PDB events. Instead, they directly refer to the probability space of all instances.

Notable examples of such queries (cf. [47, 3, 67]) are:

- *probabilistic threshold queries* that intuitively return a deterministic table containing only those facts which have a marginal probability over some specified threshold;
- *probabilistic top-k-queries* that intuitively return a deterministic table containing the k most probable facts;
- *probabilistic skyline queries* [55] that consider how different instances compare to each other with respect to some notion of *dominance*; and
- *conditioning* [47] the probabilistic database to some event.

Note that the way we informally explained the first two queries above is only sensible if the space of facts is discrete. In a continuous setting, we interpret these queries with respect to a suitable countable partition of the fact space into measurable sets.

Let $\Delta_{\mathcal{S}}$ denote the class of probabilistic databases of schema $\mathcal{S}$. Note that all PDBs in $\Delta_{\mathcal{S}}$ have the same instance measurable space $(\mathbb{D}, \mathfrak{D})$. Queries and, more generally, views of input schema $\mathcal{S}$ and output schema $\mathcal{S}'$ are now mappings $V \colon \Delta_{\mathcal{S}} \to \Delta_{\mathcal{S}'}$.

We classify views in the following way:

▶ **Definition 26.** *Let $V\colon \Delta_{\mathcal{S}} \to \Delta_{\mathcal{S}'}$ with $V\colon \Delta = (\mathbb{D}, \mathfrak{D}, P) \mapsto (\mathbb{D}', \mathfrak{D}', P') = \Delta'$.*

1. *Every view V is of* type I*.*
2. *The view V is of* type II *(or,* pointwise local*) if for every $\Delta \in \Delta_{\mathcal{S}}$ there exists a measurable mapping $q_\Delta\colon \mathbb{D} \to \mathbb{D}$ such that $P'(\mathcal{D}') = P(q_\Delta^{-1}(\mathcal{D}'))$ for every $\mathcal{D}' \in \mathfrak{D}$.*
3. *The view V is of* type III *(or,* uniformly local*) if there exists a measurable mapping $q\colon \mathbb{D} \to \mathbb{D}$ such that $P'(\mathcal{D}') = P(q^{-1}(\mathcal{D}'))$ for every $\mathcal{D}' \in \mathfrak{D}'$.*

Letting $\mathcal{V}^{\mathrm{I}}$, $\mathcal{V}^{\mathrm{II}}$ and $\mathcal{V}^{\mathrm{III}}$ denote the classes of type I, type II and type III views (from $\Delta_{\mathcal{S}}$ to $\Delta_{\mathcal{S}'}$). Then $\mathcal{V}^{\mathrm{III}}$ captures the possible worlds semantics of views. Obviously, $\mathcal{V}^{\mathrm{III}} \subseteq \mathcal{V}^{\mathrm{II}} \subseteq \mathcal{V}^{\mathrm{I}}$. The following examples show that these inclusions are strict.

▶ **Example 27.** Consider the query $Q = Q_\alpha(D) = \{f \in \mathsf{facts}_{\mathcal{S}}(R) \colon P(\mathcal{C}(f, > 0)) \geq \alpha\} = q_\Delta$ for some $\alpha > 0$. Note that the set of facts of marginal probability at least α is finite in every PDB [38], hence the query is well-defined. This query is of type II. However, considering the simple PDBs Δ_1 and Δ_2 and two distinct facts f and f' such that

- the only possible world of positive probability in Δ_1 is $\{\!\{f\}\!\}$ with $P_{\Delta_1}(\{\!\{f\}\!\}) = 1$;
- similarly, Δ_2 has the worlds $\{\!\{f\}\!\}$ and $\{\!\{f'\}\!\}$ with $P_{\Delta_2}(\{\!\{f\}\!\}) = P_{\Delta_2}(\{\!\{f'\}\!\}) = \frac{1}{2}$.

Suppose q exists like in the Definition 26, part 3 and consider the event $\mathcal{D}'$ that f' occurs (this is a set of instances in the target measurable space of Q_α). Then $P_{\Delta_1}(q^{-1}(\mathcal{D}')) = 0$ entails $\{\!\{f\}\!\} \notin q^{-1}(\mathcal{D}')$. On the other hand $P_{\Delta_2}(q^{-1}(\mathcal{D}')) = 1$ and thus $\{\!\{f\}\!\}, \{\!\{f'\}\!\} \in q^{-1}(\mathcal{D}')$, a contradiction. Thus, Q is type II, but not type III.

▶ **Example 28.** Fix some PDB Δ with three possible worlds D_1, D_2 and D_3 with probabilities $p_1 = \frac{1}{6}$, $p_2 = \frac{1}{3}$ and $p_3 = \frac{1}{2}$. Now consider the query Q that conditions Δ on the event $\{D_1, D_2\}$ and pick the database instance $D = D_1$. Then $P(D \cap \{D_1, D_2\}) = P(\{D_1\}) = \frac{1}{6}$ and $P(\{D_1, D_2\}) = \frac{1}{6} + \frac{1}{2} = \frac{4}{6}$. Thus, $P(Q^{-1}(D)) = \frac{1}{6} / \frac{4}{6} = \frac{1}{4}$, but there is no event $\mathcal{D}$ in Δ with the property that $P(\mathcal{D}) = 1/4$. Thus, Q is type I, but not type II.

8 Conclusions

In this work, we described how to construct suitable probability spaces for infinite probabilistic databases, completing the picture of [38]. The viability of this model as a general framework for finite *and infinite* databases is supported by its compositionality with respect to typical database queries. Our main technical results establish that standard query languages have a well-defined open-world semantics.

It might be interesting to explore, whether more in-depth results on point processes have a natural interpretation when it comes to probabilistic databases. We believe for example that there is a strong connection between the infinite independence assumptions that were introduced in [38] and the class of Poisson point processes (cf. [48, p. 52]).

In the last section of the paper, we briefly discussed queries for PDBs that go beyond the possible worlds semantics. Such queries are very relevant for PDBs and deserve a systematic treatment in their own right in an infinite setting.

References

1 Serge Abiteboul, T.-H. Hubert Chan, Evgeny Kharlamov, Werner Nutt, and Pierre Senellart. Capturing Continuous Data and Answering Aggregate Queries in Probabilistic XML. *ACM Transactions on Database Systems (TODS)*, 36(4):25:1–25:45, 2011. doi:10.1145/1804669.1804679.

2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, Boston, MA, USA, 1st edition, 1995.

3 Charu C. Aggarwal and Philip S. Yu. A Survey of Uncertain Data Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 21(5):609–623, 2009. doi:10.1109/TKDE.2008.190.

4 Parag Agrawal and Jennifer Widom. Continuous Uncertainty in Trio. In *Proceedings of the 3rd VLDB Workshop on Management of Uncertain Data (MUD '09)*, pages 17–32, Enschede, The Netherlands, 2009. Centre for Telematics and Information Technology (CTIT).

5 Joseph Albert. Algebraic Properties of Bag Data Types. In *Proceedings of the 17th International Conference on Very Large Databases (VLDB 1991)*, pages 211–219, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

6 Adrian Baddeley. Spatial Point Processes and Their Applications. In Wolfgang Weil, editor, *Stochastic Geometry*, Lecture Notes in Mathematics, chapter 1, pages 1–75. Springer, Berlin, Heidelberg, Germany, 1st edition, 2007.

7 Vince Bárány, Balder Ten Cate, Benny Kimelfeld, Dan Olteanu, and Zografoula Vagena. Declarative Probabilistic Programming with Datalog. *ACM Transactions on Database Systems (TODS)*, 42(4), 2017.

8 Daniel Barbará, Héctor García-Molina, and Daryl Porter. The Management of Probabilistic Data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, 1992. doi:10.1109/69.166990.

9 Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Ontology-Mediated Queries for Probabilistic Databases. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI '17)*, pages 1063–1069, Palo Alto, CA, USA, 2017. AAAI Press.

10 Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Recent Advances in Querying Probabilistic Knowledge Bases. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 5420–5426. International Joint Conferences on Artificial Intelligence, 2018. doi:10.24963/ijcai.2018/765.

11 Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Ontology-Mediated Query Answering over Log-Linear Probabilistic Data. In *Proceedings fo the Thirty-Third AAAI Conference on Artificial Intelligence*, volume 33, Palo Alto, CA, USA, 2019. AAAI Press. doi:10.1609/aaai.v33i01.33012711.

12 Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Ré, and Dan Suciu. MYSTIQ: A System for Finding more Answers by Using Probabilities. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05)*, pages 891–893, New York, NY, USA, 2005. ACM. doi:10.1145/1066157.1066277.

13 Nicolas Bourbaki. *General Topology. Chapters 5–10.* Springer, Berlin and Heidelberg, Germany, 1st edition, 1989. Original French edition published by MASSON, Paris, 1974.

14 Nicolas Bourbaki. *General Topology. Chapters 1–4.* Springer, Berlin and Heidelberg, Germany, 1st edition, 1995. Original French edition published by MASSON, Paris, 1971. doi:10.1007/978-3-642-61701-0.

15 Roger Cavallo and Michael Pittarelli. The Theory of Probabilistic Databases. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '87)*, pages 71–81, San Francisco, CA, USA, 1987. Morgan Kaufmann.

16 İsmail İlkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-World Probabilistic Databases. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR '16)*, pages 339–348, Palo Alto, CA, USA, 2016. AAAI Press.

17 Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating Probabilistic Queries over Imprecise Data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pages 551–562, New York, NY, USA, 2003. ACM. doi:10.1145/872757.872823.

18 Daryl John Daley and David Vere-Jones. *An Introduction to the Theory of Point Processes, Volume I: Elementary Theory and Models.* Probability and its Applications. Springer, New York, NY, USA, 2nd edition, 2003. doi:10.1007/b97277.

19 Daryl John Daley and David Vere-Jones. *An Introduction to the Theory of Point Processes, Volume II: General Theory and Structure*. Probability and its Applications. Springer, New York, NY, USA, 2nd edition, 2008. doi:10.1007/978-0-387-49835-5.

20 Nilesh Dalvi, Christopher Ré, and Dan Suciu. Probabilistic Databases: Diamonds in the Dirt. *Communications of the ACM*, 52(7):86–94, 2009. doi:10.1145/1538788.1538810.

21 Umeshwar Dayal, Nathan Goodman, and Randy Howard Katz. An Extended Relational Algebra with Control over Duplicate Elimination. In *Proceedings of the 1st ACM SIGACT-SIGMOD Composium on Principles of Database Systems (PODS '82)*, pages 117–123, New York, NY, USA, 1982. ACM.

22 Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, San Rafael, CA, USA, 2016.

23 Christoph Degen. *Finite Point Processes and Their Application to Target Tracking*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2015.

24 Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. Model-Driven Data Acquisition in Sensor Networks. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB '04)*, pages 588–599, St. Louis, 2004. Morgan Kaufmann. doi:10.1016/B978-012088469-8.50053-X.

25 Daniel Deutch, Christoph Koch, and Tova Milo. On Probabilistic Fixpoint and Markov Chain Query Languages. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '10)*, pages 215–226, New York, NY, USA, 2010. ACM.

26 Debabrata Dey and Sumit Sarkar. A Probabilisitic Relational Model and Algebra. *ACM Transactions on Database Systems (TODS)*, 21(3), 1996. doi:10.1145/232753.232796.

27 Anton Faradjian, Johannes Gehrke, and Philippe Bonnett. GADT: A Probability Space ADT for Representing and Querying the Physical World. In *Proceedings of the 18th International Conference on Data Engineering (ICDE '02)*, pages 201—211. IEEE Computing Society, 2002. doi:10.1109/ICDE.2002.994710.

28 Robert Fink, Larisa Han, and Dan Olteanu. Aggregation in Probabilistic Databases via Knowledge Compilation. In *Proceedings of the 38th International Conference on Very Large Data Bases (VLDB '12)*, volume 5, pages 490–501. VLDB Endowment, 2012. doi:10.14778/2140436.2140445.

29 David H. Fremlin. *Measure Theory, Volume 4: Topological Measure Spaces*. Torres Fremlin, Colchester, UK, 2nd edition, 2013.

30 David H. Fremlin. *Measure Theory, Volume 2: Broad Foundations*. Torres Fremlin, Colchester, UK, 2nd printing edition, 2016.

31 Tal Friedman and Guy Van den Broeck. On Constrained Open-World Probabilistic Databases. In *The 1st Conference on Automated Knowledge Base Construction (AKBC)*, 2019.

32 Bert E. Fristedt and Lawrence F. Gray. *A Modern Approach to Probabilitiy Theory*. Probability and its Applications. Birkhäuser, Cambridge, MA, USA, 1st edition, 1997.

33 Norbert Fuhr. Probabilistic Datalog—A Logic for Powerful Retrieval Methods. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '95)*, pages 282–290, New York, NY, USA, 1995. ACM.

34 Norbert Fuhr and Thomas Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transactions on Information Systems (TOIS)*, 15(1):32–66, 1997. doi:10.1145/239041.239045.

35 Erol Gelenbe and George Hebrail. A Probability Model of Uncertainty in Data Bases. In *1986 IEEE Second International Conference on Data Engineering*, pages 328–333. IEEE, 1986 . doi:10.1109/ICDE.1986.7266237.

36 Todd J. Green. Models for Incomplete and Probabilistic Information. In Charu C. Aggarwal, editor, *Managing and Mining Uncertain Data*, volume 35 of *Advances in Database Systems*, chapter 2, pages 9–43. Springer, Boston, MA, USA, 2009. doi:10.1007/978-0-387-09690-2_2.

37 Martin Grohe and Peter Lindner. Infinite Probabilistic Databases, 2019. arXiv e-prints, arXiv:1904.06766 [cs.DB].

38 Martin Grohe and Peter Lindner. Probabilistic Databases with an Infinite Open-World Assumption. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '19)*, pages 17–31, New York, NY, USA, 2019. ACM. Extended version available at arXiv: arXiv:1807.00607 [cs.DB]. doi:10.1145/3294052.3319681.

39 Stéphane Grumbach, Leonid Libkin, Tova Milo, and Limsoon Wong. Query Languages for Bags: Expressive Power and Complexity. *ACM SIGACT News*, 1996(2):30–44, 1996. doi:10.1145/235767.235770.

40 Stéphane Grumbach and Tova Milo. Towards Tractable Algebras for Bags. *Journal of Computer and System Sciences*, 52(3):570–588, 1996. doi:10.1006/jcss.1996.0042.

41 Ravi Jampani, Fei Xu, Mingxi Wu, Luis Perez, Chris Jermaine, and Peter J. Haas. MCDB: A Monte Carlo Approach to Managing Uncertain Data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, pages 687–700, New York, NY, USA, 2008. ACM Press. doi:10.1145/1376616.1376686.

42 Ravi Jampani, Fei Xu, Mingxi Wu, Luis Perez, Chris Jermaine, and Peter J. Haas. The Monte Carlo Database System: Stochastic Analysis Close to the Data. *ACM Transactions on Database Systems (TODS)*, 36(3):18:1–18:41, 2011. doi:10.1145/2000824.2000828.

43 Olav Kallenberg. *Foundations of Modern Probability*. Probability and its Applications. Springer, New York, NY, USA, 1st edition, 1997.

44 Oliver Kennedy and Christoph Koch. PIP: A Database System for Great and Small Expectations. In *Proceedings of the 26th International Conference on Data Engineering (ICDE '10)*, pages 157–168, Washington, DC, USA, 2010. IEEE.

45 Christoph Koch. On Query Algebras for Probabilistic Databases. *ACM SIGMOD Record*, 37(4):78–85, 2008.

46 Christoph Koch. MayBMS: A System for Managing Large Probabilistic Databases. In Charu C. Aggarwal, editor, *Managing and Mining Uncertain Data*, volume 35 of *Advances in Database Systems*, chapter 6, pages 149–184. Springer, Boston, MA, USA, 2009. doi:10.1007/978-0-387-09690-2_6.

47 Christoph Koch and Dan Olteanu. Conditioning Probabilistic Databases. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB '08)*, volume 1, pages 313–325. VLDB Endowment, 2008. doi:10.14778/1453856.1453894.

48 Günter Last and Matthew Penrose. *Lectures on the Poisson Process*. Institute of Mathematical Statistics Textbook. Cambridge University Press, Cambridge, UK, 2017. doi:10.1017/9781316104477.

49 Odile Macchi. The Coincidence Approach to Stochastic Point Processes. *Advances in Applied Probability*, 7(1):83–122, 1975. doi:10.2307/1425855.

50 Ronald P. S. Mahler. *Statistical Multisource-Multitarget Information Fusion*. Artech House, Inc., Norwood, MA, USA, 2007.

51 Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, David L. Ong, and Andrey Kolobov. BLOG: Probabilistic Models with Unknown Objects. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*, St. Louis, MO, USA, 2005. Morgan Kaufmann.

52 Brian Christopher Milch. *Probabilistic Models with Unknown Objects*. PhD thesis, University of California, Berkeley, 2006.

53 José Enrique Moyal. The General Theory of Stochastic Population Processes. *Acta Mathematica*, 108:1–31, 1962. doi:10.1007/BF02545761.

54 Raghotham Murthy, Robert Ikeda, and Jennifer Widom. Making Aggregation Work in Uncertain and Probabilistic Databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(8):1261–1273, 2011. doi:10.1109/TKDE.2010.166.

55 Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic Skylines on Uncertain Data. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07)*, pages 15–26. VLDB Endowment, 2007.

56 Michael Pittarelli. An Algebra for Probabilistic Databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 6(2):293–303, 1994.

57 Raymond Reiter. On Closed World Data Bases. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, NY, USA, 1st edition, 1978.

58 Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Machine Learning*, 62(1–2):107—136, 2006. doi:10.1007/s10994-006-5833-1.

59 Robert Ross, V. S. Subrahmanian, and John Grant. Aggregate Operators in Probabilistic Databases. *Journal of the ACM (JACM)*, 52(1):54–101, 2005. doi:10.1145/1044731.1044734.

60 Sarvjeet Singh, Chris Mayfield, Rahul Shah, Sunil Prabhakar, Susanne Hambrusch, Jennifer Neville, and Reynold Cheng. Database Support for Probabilistic Attributes and Tuples. In *2008 IEEE 24th International Conference on Data Engineering (ICDE '08)*, pages 1053–1061, Washington, DC, USA, 2008. IEEE Computer Society. doi:10.1109/ICDE.2008.4497514.

61 Parag Singla and Pedro Domingos. Markov Logic in Infinite Domains. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI '07)*, pages 368–375, Arlington, VA, USA, 2007. AUAI Press.

62 Shashi Mohan Srivastava. *A Course on Borel Sets*. Graduate Texts in Mathematics. Springer, New York, NY, USA, 1st edition, 1998. doi:10.1007/b98956.

63 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool, San Rafael, CA, USA, 1st edition, 2011. doi:10.2200/S00362ED1V01Y201105DTM016.

64 Thanh T. L. Tran, Liping Peng, Yanlei Diao, Andrew McGregor, and Anna Liu. CLARO: Modeling and Processing Uncertain Data Streams. *The VLDB Journal*, 21(5):651–676, 2012. doi:10.1007/s00778-011-0261-7.

65 Guy Van den Broeck and Dan Suciu. Query Processing on Probabilistic Data: A Survey. *Foundations and Trends® in Databases*, 7(3–4):197–341, 2017. doi:10.1561/1900000052.

66 Brend Wanders and Maurice van Keulen. Revisiting the Formal Foundation of Probabilistic Databases. In *Proceedings of the 2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology (IFSA-EUSFLAT '15)*, Advances in Intelligent Systems Research, pages 289–296, Paris, France, 2015. Atlantis Press.

67 Yijie Wang, Xiaoyong Li, Xiaoling Li, and Yuan Wang. A Survey of Queries over Uncertain Data. *Knowledge and Information Systems*, 37(3):485–530, 2013. doi:10.1007/s10115-013-0638-6.

68 Jennifer Widom. Trio: A System for Data, Uncertainty, and Lineage. In Charu C. Aggarwal, editor, *Managing and Mining Uncertain Data*, volume 35 of *Advances in Database Systems*, chapter 5, pages 113–148. Springer, Boston, MA, USA, 2009. doi:10.1007/978-0-387-09690-2_5.

69 Eugene Wong. A Statistical Approach to Incomplete Information in Database Systems. *ACM Transactions on Database Systems (TODS)*, 7(3):470–488, 1982. doi:10.1145/319732.319747.

70 Yi Wu, Siddharth Srivastava, Nicholas Hay, Simon Du, and Stuart Russell. Discrete-Continuous Mixtures in Probabilistic Programming: Generalized Semantics and Inference Algorithms. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, volume 80 of *Proceedings of Machine Learning Research*, pages 5343–5352. PMLR, 2018.

71 Esteban Zimányi. Query Evaluation in Probabilistic Relational Databases. *Theoretical Computer Science*, 171(1):179–219, 1997. doi:10.1016/S0304-3975(96)00129-6.

Coordination-Free Byzantine Replication with Minimal Communication Costs

Jelle Hellings

Exploratory Systems Lab, Department of Computer Science, University of California, Davis,
Davis, CA 95616-8562, USA
jhellings@ucdavis.edu

Mohammad Sadoghi

Exploratory Systems Lab, Department of Computer Science, University of California, Davis,
Davis, CA 95616-8562, USA
msadoghi@ucdavis.edu

Abstract

State-of-the-art fault-tolerant and federated data management systems rely on fully-replicated designs in which all participants have equivalent roles. Consequently, these systems have only limited scalability and are ill-suited for high-performance data management. As an alternative, we propose a hierarchical design in which a *Byzantine cluster* manages data, while an arbitrary number of *learners* can reliable learn these updates and use the corresponding data.

To realize our design, we propose the *delayed-replication algorithm*, an efficient solution to the *Byzantine learner problem* that is central to our design. The delayed-replication algorithm is coordination-free, scalable, and has minimal communication cost for all participants involved. In doing so, the delayed-broadcast algorithm opens the door to new high-performance fault-tolerant and federated data management systems. To illustrate this, we show that the delayed-replication algorithm is not only useful to support specialized learners, but can also be used to reduce the overall communication cost of permissioned blockchains and to improve their storage scalability.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Information systems → Distributed database transactions

Keywords and phrases Byzantine learner, coordination-free checkpoint protocol, delayed-replication, information dispersal, consensus

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.17

1 Introduction

Recently saw the introduction of several blockchain-inspired database systems and blockchain fabrics [5, 6, 24, 25, 54, 55]. At the same time, there is also a huge interest from public and private sectors in blockchain technology (e.g., [7, 11, 13, 16, 18, 27, 29, 34, 35, 40, 41, 52, 55, 61, 62, 65, 73]). In each of these systems and use cases, blockchain technology is used to provide *fault-tolerant and federated data management*: systems in which independent participants (e.g., different companies) together manage a single common database and that continuously provide reliable services even when some of the participants are compromised. The interest in fault-tolerant and federated data management is easy explained by the huge societal and economic impact of recent cyberattacks on data-based services [31, 56, 57, 58, 68], and on the huge negative economic impact of bad data [23, 39, 64].

Blockchain techniques build upon traditional distributed consensus [38, 53]: both traditional techniques and their blockchain counterparts provide fault-tolerant and federated data management via a *fully-replicated* design in which all participants (replicas) maintain a full copy of all data and participate in modifying this data. To do so, traditional consensus – which are also used in *permissioned blockchains* in which the identities of all participants are known – requires vast amounts of communication between all participants

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 17; pp. 17:1–17:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(e.g., [8, 9, 14, 15, 45, 46, 59, 69, 70]). Consequently, systems using traditional consensus have difficulty scaling up to hundreds of participants. Techniques used in anonymous permissionless blockchains such as Bitcoin can effectively support thousands of participants, however. Unfortunately, these blockchain techniques incur massive *computational costs* on all participants, which has raised questions about the sustainability of the *energy consumption* of these systems [21, 72]. Even with these computational costs, the performance of Bitcoin is abysmal, as Bitcoin can only process 7 transactions per second [61].

We see the necessity of fault-tolerant and federated data management but – as outlined above – we also believe that the current state-of-the-art techniques are too limited and lack scalability. To combat these limitations, we believe there is a strong need for the development of more refined scalable designs that provide fault-tolerant and federated data management.

1.1 Our Vision: Specializing for Read-only Workloads

In many practical distributed database and data processing systems, a distinction is made between read-only workloads and update-workloads [1, 19, 20, 22, 32, 60, 70]. Typically, read-only workloads are isolated to a single replica, whereas update-workloads are executed by all replicas (e.g., via a commit protocol [30, 36, 67]). In most cases this improves scalability significantly, as the majority of workloads are read-only and can be processed in parallel by individual replicas. Unfortunately, such read-only single-replica optimizations cannot be applied to state-of-the-art fault-tolerant and federated data management: fault-tolerant systems need to assure validity of the result of every read-only query in the presence of malicious replicas. These systems do so by executing every query at all replicas, after which the issuer of the query can compare the query outcomes and determine which outcome is valid (supported by a majority).

In many practical situations, workloads need access to the full history of all the data managed or to large portions thereof. Examples of such workloads are analytics, data provenance, machine learning, and data visualization. For data-hungry workloads, it makes little sense to retrieve all data in an inefficient way via read-only queries. Furthermore, these workloads are typically computational complex, ruling out their integration within a fault-tolerant system. To enable these practical workloads, we propose an alternative *hierarchical design*. This hierarchical design is sketched in Figure 1.

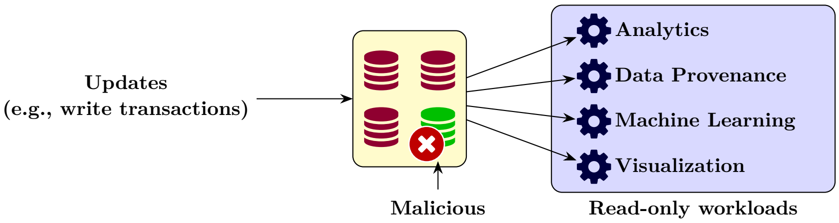


Figure 1 Schematic overview of *hierarchical* fault-tolerant and federated data management. At the core is a *Byzantine cluster* that manages and stores all data in a fault-tolerant manner. Some of the replicas in this core can crash or be malicious. The managed data is used by many *independent read-only participants*, e.g., for analytics, data provenance, machine learning, and visualization. To do so, these participants do not need to partake in managing and storing the data, they only need to *reliably learn the data*.

In our design, we propose that a *Byzantine cluster* of replicas (e.g., a permissioned blockchain system) manages the data by coordinating data updates. As the cluster is Byzantine fault-tolerant, it can be used to provide fault-tolerant and federated data management.

Dedicated *learners*, independent of the Byzantine cluster, can register themselves at the Byzantine cluster to receive all data updates. These learners will receive the stream of data updates made in the cluster and will receive these updates in an efficient and reliable manner. On these learners, data-hungry and compute intensive read-only workloads (e.g., analytics, data provenance, machine learning, and data visualization) can be performed efficiently without affecting the Byzantine cluster. Learners can also be deployed in trusted environments close to end-users and act as read-only proxies. In this read-only proxy capacity, learners provide end-users with high performance, low latency, read-only access to the data. In this hierarchical design, learners cannot directly modify the data, but can still forward data update requests to the Byzantine cluster. The Byzantine cluster can, in turn, assure reliable processing of such updates.

1.2 The Need for the Byzantine Learner Problem

To enable the hierarchical design proposed in the previous section, we need to develop techniques to reliably sent the data updates made by a *Byzantine cluster* to independent *learners* – which we refer to as the *Byzantine learner problem.* In specific, our contributions are as follows:

1. We formalize the Byzantine learner problem.
2. We demonstrate that the straightforward *pull-based* solution to this learning problem is highly inefficient and enables several attacks.
3. To address the Byzantine learner problem, we propose the *delayed-replication algorithm*, a coordination-free, push-based, scalable algorithm with minimal communication cost for both the sending cluster and the receiving learner.
4. We provide three specialized variants of the delayed-replication algorithm, whose characteristics are summarized in Figure 2. The basic variant does not use checksums and can deal with clusters in which replicas can *crash.* We also provide a variant that uses *simple checksums* to deal with *Byzantine replicas* that sent corrupted or otherwise invalid messages. The final variant uses *tree checksums* to aid learners in discarding corrupted or otherwise invalid messages with low computational costs. These tree checksums only add minimal communication costs for all participants involved.
5. To further underline the strengths of the delayed-replication algorithm, we show that the delayed-replication algorithm can be used to improve the design of permissioned blockchain systems and other types of Byzantine clusters. First, we show how delayed-replication techniques enables scalable shared storage designs for permissioned blockchain systems, allowing them to turn away from wasteful state-of-the-art fully-replicated designs. Then, we show how the delayed-replication algorithm can be used within permissioned blockchain systems to reduce the communication complexity of coordinating data updates.

2 Formalizing the Byzantine Learner Problem

We model a *system* as a tuple $(\mathfrak{R}, \mathfrak{L})$, in which $\mathfrak{R}$ is a *Byzantine cluster* of replicas that make *update decisions* and $\mathfrak{L}$ is a set of *learners* that want to learn these *update decisions.* We assign each replica $\textsc{r} \in \mathfrak{R}$ a unique identifier $\mathsf{id}(\textsc{r})$ with $0 \leq \mathsf{id}(\textsc{r}) < |\mathfrak{R}|$. We write $\mathcal{B} \subseteq \mathfrak{R}$ to denote the set of *Byzantine replicas* that can behave in arbitrary, possibly coordinated and malicious, manners; we write $\mathcal{C} \subseteq \mathfrak{R}$ to denote the set of *crashed replicas* that behave correctly up till some point after which they stop participating; and we write $\mathcal{G} = \mathfrak{R} \setminus (\mathcal{B} \cup \mathcal{C})$ to denote the set of *non-faulty (good) replicas* in $\mathfrak{R}$. We assume that non-Byzantine replicas

System	Checksum	Complexity for the learner		
		Data sent per replica	*Data received*	*Decode steps*
$\mathbf{b} = 0$	None	$\mathcal{O}(s/\mathbf{g})$	$\mathcal{O}(s(\mathbf{n}/\mathbf{g}))$	$u/\mathbf{n}$
$\mathbf{b} < \mathbf{g}$	Simple	$\mathcal{O}(s/\mathbf{g})$	$\mathcal{O}(s(\mathbf{n}/\mathbf{g}))$	$\binom{\mathbf{g}+\mathbf{b}}{\mathbf{g}}(u/\mathbf{n})$
$\mathbf{b} < \mathbf{g}$	Tree	$\mathcal{O}(s/\mathbf{g} + (u/\mathbf{n})\log(\mathbf{n}))$	$\mathcal{O}(s(\mathbf{n}/\mathbf{g}) + u\log(\mathbf{n}))$	$u/\mathbf{n}$

Figure 2 Overview of *delayed-replication algorithms* running on a cluster of $\mathbf{n}$ replicas, of which $\mathbf{b}$ are Byzantine and $\mathbf{g}$ are non-faulty. The first two columns describe the system conditions and the checksums used. The last three columns provide the complexity to sent a journal with u updates and storage size s to a learner in terms of the data sent per replica, data received by the learner, and the worst-case number of decode steps the learner needs to perform.

behave in accordance to the algorithms and are deterministic: on identical inputs, non-faulty replicas must produce identical outputs. Notice that we do not make any assumptions on the learners, each learner can be malicious without affecting the operations in $\mathfrak{R}$. We write $\mathbf{n} = |\mathfrak{R}|$, $\mathbf{b} = |\mathcal{B}|$, $\mathbf{c} = |\mathcal{C}|$, and $\mathbf{g} = |\mathcal{G}|$ to denote the number of replicas, Byzantine replicas, crashed replicas, and non-faulty replicas, respectively. Finally, we assume that $\mathbf{g} > \mathbf{b}$, a minimal condition to distinguish Byzantine and non-Byzantine behavior.

▶ **Definition 2.1.** *Let* $(\mathfrak{R}, \mathfrak{L})$ *be a system. The* Byzantine learner problem *states that each learner in* $\mathfrak{L}$ *will eventually learn of the update decisions made by* $\mathfrak{R}$*.*

We will formalize the Byzantine learner problem in terms of learning *journal updates*. Let $(\mathfrak{R}, \mathfrak{L})$ be a system. We assume that each replica $\mathrm{R} \in \mathfrak{R}$ maintains an append-only *update journal* $\mathbb{J}_{\mathrm{R}}$ that consists of a sequence of *data updates* (e.g., write transactions in a database system). To work with sequences, we introduce the following notations. Let $S = [s_0, \dots, s_{m-1}]$ be a sequence. We write $S[i]$ to denote s_i, $S[i : j]$ to denote $[s_i, \dots, s_{j-1}]$, and $|S|$ to denote the length m of S. Finally, if T is also a sequence, then S is a *prefix* of T, denoted $S \preceq T$, if $|S| \leq |T|$ and $S = T[0 : |S|]$. We refer to any subsequence $S[i : i + \mathbf{n}]$, $i \bmod \mathbf{n} = 0$, as a *block*.

We assume that the non-Byzantine replicas all make *the same update decisions* in the same order (e.g., by utilizing a consensus protocol such as `Paxos` or `Pbft` [14, 15, 46, 47]). These updates are not necessarily registered at each replica at exactly the same time. Consequently, we can only assume that, for each $\mathrm{R}, \mathrm{Q} \in (\mathcal{G} \cup \mathcal{C})$, either $\mathbb{J}_{\mathrm{R}} \preceq \mathbb{J}_{\mathrm{Q}}$ or $\mathbb{J}_{\mathrm{Q}} \preceq \mathbb{J}_{\mathrm{R}}$. We write $\mathbb{J}_{\mathfrak{R}}$ to denote the unique journal $\mathbb{J}_{\mathrm{Q}}$, $\mathrm{Q} \in \mathcal{G}$, that contains the maximum-length sequence of update decisions all non-faulty replicas agree on. Hence, $\mathbb{J}_{\mathfrak{R}} \preceq \mathbb{J}_{\mathrm{R}}$ for all $\mathrm{R} \in \mathcal{G}$.

▶ **Example 2.2.** Consider a Byzantine cluster $\mathfrak{R} = \{\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2, \mathrm{B}\}$ with

$$\mathbb{J}_{\mathrm{R}_0} = [u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7]; \qquad \mathbb{J}_{\mathrm{R}_1} = [u_0, u_1, u_2, u_3, u_4, u_5, u_6];$$
$$\mathbb{J}_{\mathrm{R}_2} = [u_0, u_1, u_2, u_3, u_4, u_5, u_6]; \qquad \mathbb{J}_{\mathrm{B}} = [u_0, u_1, u_2, u_3', u_4'].$$

The update journal of replica B diverges from the other replicas and, hence, B must be Byzantine. The three non-faulty replicas share the update journal $\mathbb{J}_{\mathfrak{R}} = [u_1, u_2, u_3, u_4, u_5, u_6]$. Currently, the cluster is deciding on the eight update u_7. This update is already fully processed by R_0, whereas replicas R_1 and R_2 are still processing this update.

▶ **Definition 2.3.** *Let* $(\mathfrak{R}, \mathfrak{L})$ *be a system and* $\mathrm{L} \in \mathfrak{L}$ *a learner. For every* i*,* $0 \leq i < |\mathbb{J}_{\mathfrak{R}}|$*, the* Byzantine learner problem *states that* L *will eventually learn of the* i*-th update decision* $\mathbb{J}_{\mathfrak{R}}[i]$*. At the same time, no Byzantine replica* $\mathrm{B} \in \mathcal{B}$ *can convince* L *that any other update was the* i*-th update decision made.*

Notice that we only specified the data model of replicas. We did not specify how the learners store data or process data, we only specified that the learners will receive *all* update decisions made by the replicas. Indeed, the specifics of what a learner does with the updates received depend on the workload for which the learner is designed.

▶ **Example 2.4.** Consider the Byzantine cluster $\mathfrak{R}$ from Example 2.2. A learner $\textsc{l}$ will be able to learn the updates u_0, u_1, u_2, u_3, u_4, u_5, and u_6. The learner will not yet be able to learn u_7, as this update is still being processed by some non-faulty replicas in $\mathfrak{R}$. Replica $\textsc{b}$ will never be able to convince the learner that the updates u'_3 or u'_4 happened, as $\textsc{b}$ is Byzantine. The learner $\textsc{l}$ can store the updates it learned in a temporal database view that provides access to historical data, e.g., for in-depth analysis.

As a simple solution to the *Byzantine learner problem*, consider a system in which each replica can be queried for their journal content. Any learner $\textsc{l}$ can now determine the i-th journal update by simply querying different replicas in $\mathfrak{R}$. As soon as $\textsc{l}$ receives $\mathbf{b}+1$ identical responses, it is ensured that at least one of these responses came from a non-Byzantine replica and, hence, must be the valid i-th journal update. Unfortunately, this simple and naive solution has several major weaknesses that can be exploited by malicious participants:

▶ **Example 2.5.** Firstly, there is the issue of *load balancing* due to a lack of coordination. As all learners have to query for journal updates independently, they can all end up querying the same non-faulty replica $\textsc{r} \in \mathcal{G}$. Due to the amount of queries, $\textsc{r}$ has to dedicate most of its resources to answering these queries. Consequently, $\textsc{r}$ will have fewer resources available for its other tasks, e.g., for deciding on new data updates. In the worst case, this can reduce the data update throughput of $\mathfrak{R}$. Secondly, the issue of load balancing can be *exploited* by the ability of *malicious learners* to coordinately target some non-faulty replicas, which could overload these replicas in an attempt to impede the services of $\mathfrak{R}$.

Moreover, as learners do not have a reliable way to distinguish between non-faulty replicas that are slow, crashed replicas, and Byzantine replicas, they have to always query at least $\mathbf{b}+\mathbf{c}+1$ distinct replicas in $\mathfrak{R}$ to have a guarantee on an outcome (as $\mathbf{b}+\mathbf{c}$ replicas could be Byzantine or have crashed and consequently not respond). Furthermore, an additional $\mathbf{b}$ distinct replicas in $\mathfrak{R}$ need to be queried to assure that the majority of all received outcomes come from non-Byzantine replicas (as $\mathbf{b}$ replicas could be Byzantine and respond with invalid or corrupted outcomes). This makes learning an update *unnecessary expensive*.

In Section 3, we propose the delayed-replication algorithm to provide reliable high-performance Byzantine learning that does not suffer from the shortcomings of the above naive simple solution.

In the following, we assume *asynchronous reliable communication*: all messages send by non-faulty replicas will eventually arrive at their destination. We also assume *authenticated communication*: on receipt of a message m from replica $\textsc{r} \in \mathfrak{R}$, one can determine that $\textsc{r}$ did sent m if $\textsc{r} \notin \mathcal{B}$; and one can only determine that m was sent by a replica in $\mathcal{C}\cup\mathcal{G}$ if $\textsc{r} \in (\mathcal{C}\cup\mathcal{G})$. Hence, Byzantine replicas are able to impersonate each other, but are not able to impersonate non-Byzantine replicas. Authenticated communication is a minimum requirement to deal with Byzantine behavior and can be implemented using message authentication codes [43, 50].

3 The Delayed-Replication Algorithm

Next, we propose the *delayed-replication algorithm*, which provides an efficient solution to the Byzantine learner problem. Our delayed-replication algorithm uses *information dispersal* [63] to balance the load among all non-faulty replicas and to minimize overall communication costs. The delayed-replication algorithm itself consists of two parts: the *information dispersal step*, which is executed by the replicas in $\mathfrak{R}$, and the *information learning step*, which is executed by the learners in $\mathfrak{L}$.

3.1 Information Dispersal

We use an *information dispersal algorithm* that is able to *encode* any value v with storage size $\|v\|$ into $\mathbf{n}$ pieces v_i, $0 \leq i < \mathbf{n}$, such that v can be *decoded* from every set of $\mathbf{g}$ distinct pieces. We assume that the information dispersal algorithm is *optimal* in the sense that each piece v_i has size $\|v_i\| \leq \lceil \|v\|/\mathbf{g} \rceil$. Hence, the minimal number of pieces necessary for recovering v by decoding, $\mathbf{g}$ pieces, have a combined storage size of $\mathbf{g} \lceil \|v\|/\mathbf{g} \rceil \approx \|v\|$. The information dispersal algorithm (IDA) of Rabin provides these properties [63].

We assume that each non-Byzantine replica $\textsc{r} \in (\mathcal{C} \cup \mathcal{G})$ is equipped with IDA. We write $\texttt{slice}_{\textsc{r}}(v)$, for any value v, to denote the $\mathsf{id}(\textsc{r})$-th piece $v_{\mathsf{id}(\textsc{r})}$ obtained by encoding v. With these assumptions and notations, we have $\|\texttt{slice}_{\textsc{r}}(v)\| \leq \lceil \|v\|/\mathbf{g} \rceil$.

► **Example 3.1.** Consider a cluster $\mathfrak{R} = \{\textsc{r}_0, \textsc{r}_1, \textsc{r}_2, \textsc{b}\}$ with $\mathcal{B} = \{\textsc{b}\}$ and $\mathcal{C} = \emptyset$. Hence, $\mathbf{g} = 3$ and $\mathbf{b} = 1$. Let v be a piece of data. When using the encode step of IDA, we obtain pieces v_0, v_1, v_2, v_3 with $\|v_0\| = \|v_1\| = \|v_2\| = \|v_3\| = \lceil \|v\|/3 \rceil$. Consequently, $\texttt{slice}_{\textsc{r}_0}(v) = v_0$, $\texttt{slice}_{\textsc{r}_1}(v) = v_1$, $\texttt{slice}_{\textsc{r}_2}(v) = v_2$, and, finally, $\texttt{slice}_{\textsc{b}}(v) = v_3$.

Now consider any learner $\textsc{l} \in \mathfrak{L}$. Upon obtaining any three valid and distinct pieces, $\textsc{l}$ can use the decode step of IDA to reconstruct v. As the replicas $\textsc{r}_0$, $\textsc{r}_1$, and $\textsc{r}_2$ are non-faulty, $\textsc{l}$ will always be able to obtain v_0, v_1, and v_2. Hence, $\textsc{l}$ can reconstruct v. We notice that $\|v_0\| + \|v_1\| + \|v_2\| = 3 \lceil \|v\|/3 \rceil \approx \|v\|$. Hence, the communication required for $\textsc{l}$ to reconstruct v is minimal (due to IDA being optimal).

3.2 The Information Dispersal Step

In the *information dispersal step*, every replica $\textsc{r} \in \mathfrak{R}$ is instructed to broadcast the update decisions appended to their journal after every block $\mathbb{B}$ of $\mathbf{n}$ appends. The pseudo-code for the information dispersal step can be found in Figure 3. To minimize communication costs, replicas will encode the block $\mathbb{B}$ using an optimal information dispersal algorithm (Line 4). To allow learners to validate the correctness of the encoded block, replicas will include a *checksum* of $\mathbb{B}$. The exact type of checksum used depends on the type of attacks the delayed-replication algorithm needs to be able to deal with, and we refer to the information learning step for details on the types of checksums supported (Section 3.3 and Section 3.4). After encoding, each replica broadcasts the encoded block and the checksum to all learners (Line 5). In doing so, the information dispersal step provides reliable replication of sufficient information among *all* learners such that each learner can reconstruct any segment of $\mathbf{n}$ update decisions. We refer to Figure 4 for a schematic representation of the interactions between replicas and a learner due to the information dispersal step.

We notice that the information dispersal step is a *push-based* algorithm that pushes the update journal to all learners without any coordination. Additionally, the total communication cost of the information dispersal step is shared equally among all participating replicas, independent of the behavior of any faulty replicas. This is in sharp contrast with the simple and naive *pull-based* approach of Section 2, which is at the basis of many *practical checkpoint algorithms* (see, e.g., Section 4.2). Next, we show that the communication complexity of the information dispersal step is low.

```
event R appends a new decision to $\mathbb{J}_{R}$ do
    if $\mathbb{J}_{R} \neq [\,]$ and $|\mathbb{J}_{R}| \bmod \mathbf{n} = 0$ then
        $\mathbb{B} := \mathbb{J}_{R}[|\mathbb{J}_{R}| - \mathbf{n} : |\mathbb{J}_{R}|]$.
        $s, c := \texttt{slice}_{R}(\mathbb{B}), \texttt{checksum}(\mathbb{B})$.
        Broadcast $(|\mathbb{J}_{R}|, s, c)$ to all learners $L \in \mathfrak{L}$.
```

Figure 3 The *information dispersal step* of the delayed-replication algorithm running at every non-faulty replica $R \in \mathcal{G}$.

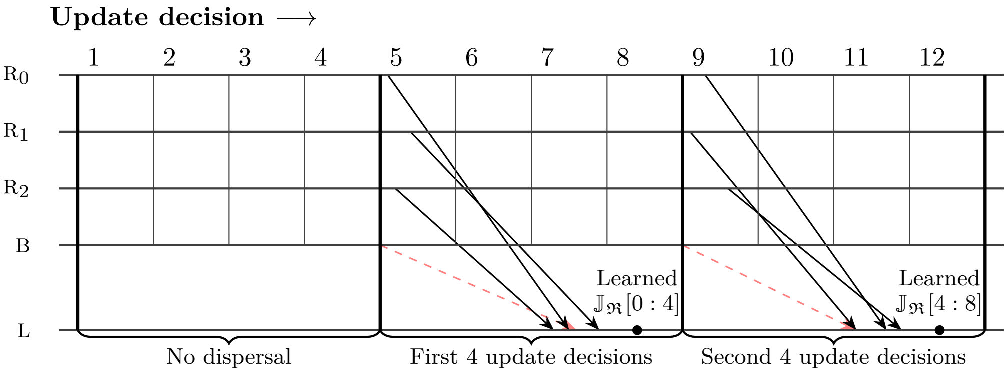

Update decision ⟶
1
2
3
4
5
6
7
8
9
10
11
12
R_0
R_1
R_2
B
L
Learned $\mathbb{J}_{\mathfrak{R}}[0:4]$
Learned $\mathbb{J}_{\mathfrak{R}}[4:8]$
No dispersal
First 4 update decisions
Second 4 update decisions

Figure 4 A schematic representation of the interactions between a cluster $\mathfrak{R} = \{R_0, R_1, R_2, B\}$ and a learner L participating in the information dispersal step. The replica B is Byzantine and sends invalid messages. The other replicas repeatedly send a valid message encoding 4 decisions from their update journal. After receiving these messages, L is able to reconstruct (learn) the update decisions made by $\mathfrak{R}$. In specific, after the $\mathbf{n} = 4$-th update decision, L will start receiving messages from which it can reconstruct the first four update decisions.

▶ **Theorem 3.2.** *Consider the information dispersal step of Figure 3 running at replica $R \in \mathcal{G}$ after R appends the ρ-th decision, $\rho \geq 1$, to $\mathbb{J}_{R}$. After this step, R has sent $\lfloor \rho/\mathbf{n} \rfloor$ messages to each learner with a total size of $\mathcal{O}(c(\rho/\mathbf{n}) + \|\mathbb{J}_{R}\|/\mathbf{g})$, in which c is the size of a checksum.*

Proof. Notice that R only broadcasts messages after every i-th decision, $i \geq 1$ and $i \bmod \mathbf{n} = 0$. Hence, after the ρ-th decision, R will have broadcasted $m = \lfloor \rho/\mathbf{n} \rfloor$ messages. Consider the messages sent by R to any learner $L \in \mathfrak{L}$. In these messages, the pieces $\texttt{slice}_{R}(\mathbb{J}_{R}[(i-1)\mathbf{n} : i\mathbf{n}])$, $1 \leq i \leq m$, have a non-constant size and we assume that the remainder of each message has size $\gamma = \Theta(c)$. Hence, in total, the m messages send to L have size σ at most

$$\begin{aligned}
\sigma &\leq \sum_{1 \leq i \leq m} (\gamma + \|\texttt{slice}_{R}(\mathbb{J}_{R}[(i-1)\mathbf{n} : i\mathbf{n}])\|) \\
&\leq \gamma m + \sum_{1 \leq i \leq m} \left\lceil \frac{\|\mathbb{J}_{R}[(i-1)\mathbf{n} : i\mathbf{n}]\|}{\mathbf{g}} \right\rceil \leq \gamma m + \sum_{1 \leq i \leq m} \left(1 + \frac{\|\mathbb{J}_{R}[(i-1)\mathbf{n} : i\mathbf{n}]\|}{\mathbf{g}}\right) \\
&\leq \gamma m + m + \frac{\|\mathbb{J}_{R}[0 : \mathbf{n}m]\|}{\mathbf{g}} \leq m(\gamma + 1) + \frac{\|\mathbb{J}_{R}\|}{\mathbf{g}} = \mathcal{O}(c(\rho/\mathbf{n}) + \|\mathbb{J}_{R}\|/\mathbf{g}).
\end{aligned}$$ ◀

Based on Theorem 3.2, it is straightforward to determine the number and size of messages received by each learner.

▶ **Corollary 3.3.** *Consider the learner* $\textsc{l} \in \mathfrak{L}$ *after it has received all messages sent by the information dispersal steps following the* ρ*-th decision in* $\mathbb{J}_{\mathfrak{R}}$*,* $\rho \geq 1$*. The learner* $\textsc{l}$ *has received at most* ρ *messages with a total size of* $\mathcal{O}(c\rho + \|\mathbb{J}_{\mathfrak{R}}\|(\mathbf{n}/\mathbf{g}))$*, in which* c *is the size of a checksum.*

To conclude, we notice that the information dispersal step we provide assumes a steady flow of update decisions. If update decisions are infrequent, then information dispersal can be delayed arbitrary. To deal with periods of inactivity, the system can always resort to filling the current block of $\mathbf{n}$ updates with `null`-values (although this will reduce communication efficiency).

3.3 The Information Learning Step with Simple Checksums

In the previous section, we presented the information dispersal step that will broadcast an encoded block of journal updates from $\mathbb{J}_{\mathfrak{R}}$ to each learner $\textsc{l} \in \mathfrak{L}$. In this section, we show how $\textsc{l}$ can reliable reconstruct these journal updates from the encoded information. To provide resilience against Byzantine replicas, we will use *simple checksums* $\texttt{checksum}(\mathbb{B}) = \texttt{hash}(\mathbb{B})$, in which $\texttt{hash}(\cdot)$ is a *collision-resistant hash function* that maps an arbitrary value v to a numeric value $\texttt{hash}(v)$ in a bounded range [43, 50]. We assume that it is practically impossible to find another value v', $v \neq v'$, such that $\texttt{hash}(v) = \texttt{hash}(v')$. These simple checksums have a constant size independent of $\mathbf{n}$ or $\mathbf{g}$.

▶ **Theorem 3.4.** *Consider the learner* $\textsc{l} \in \mathfrak{L}$ *after it has received all messages sent by the information dispersal steps following the* ρ*-th decision in* $\mathbb{J}_{\mathfrak{R}}$*,* $\rho \geq 1$*. If* $\mathbf{g} > \mathbf{b}$*, then, after receiving these messages,* $\textsc{l}$ *can reconstruct the first* $\mathbf{n}\lfloor\rho/\mathbf{n}\rfloor$ *update decisions made by* $\mathfrak{R}$ *using at most* $\binom{\mathbf{g}+\mathbf{b}}{\mathbf{g}}\lfloor\rho/\mathbf{n}\rfloor$ *information dispersal decode steps.*

Proof. Let $i = \mathbf{n}\lfloor\rho/\mathbf{n}\rfloor$ be the last round after which $\textsc{l}$ received update decisions. We assume that $\textsc{l}$ has already reconstructed the first $(i-1)\mathbf{n}$ update decisions, and we show how $\textsc{l}$ can reconstruct the block $\mathbb{B}$ containing update decisions $(i-1)\mathbf{n}, \dots, i\mathbf{n}-1$, this independent of the behavior of the Byzantine replicas. To initiate reconstruction of $\mathbb{B}$, $\textsc{l}$ will collect messages of the form (i, s_j, c_j), with s_j an encoded piece of $\mathbb{B}$ and c_j a checksum, of distinct replicas $\textsc{r}_j$ with $\mathsf{id}(\textsc{r}_j) = j$. Eventually, $\textsc{l}$ will receive $\mathbf{g}$ messages from replicas in $\mathcal{C} \cup \mathcal{G}$, as all $\mathbf{g}$ replicas in $\mathcal{G}$ will send messages to $\textsc{l}$. Using these messages, $\textsc{l}$ can reconstruct $\mathbb{B}$ by decoding the pieces contained in the received messages.

Byzantine replicas are able to send corrupted messages, however, which complicates construction of $\mathbb{B}$ from the messages received. Learners do not a-priori know which replicas are Byzantine. Hence, learners need to verify whether any block reconstructed from $\mathbf{g}$ collected messages is equivalent to $\mathbb{B}$. The first step in this verification process is to determine the checksum $\texttt{hash}(\mathbb{B})$. Consider the first $z > \mathbf{b}$ messages received. We distinguish two cases:

1. At least $\mathbf{b}+1$ messages have identical checksum c. In this case, at least one such message must be sent by a non-faulty replica. Hence, we have $c = \texttt{hash}(\mathbb{B})$.
2. At most $\mathbf{b}$ messages have identical checksums. In this case, some of the messages received have been sent by Byzantine replicas. As $\textsc{l}$ will eventually receive $\mathbf{g} > \mathbf{b}$ messages from non-faulty replicas, and all these messages will contain the same checksum $\texttt{hash}(\mathbb{B})$, we can wait until more messages are received to determine the checksum $\texttt{hash}(\mathbb{B})$.

After determining $\texttt{hash}(\mathbb{B})$, $\textsc{l}$ can simply reconstruct $\mathbb{B}$ by trying to decode every combination of $\mathbf{g}$ received pieces, this until eventually a block b is constructed with $\texttt{hash}(b) = \texttt{hash}(\mathbb{B})$. In the worst case, $\textsc{l}$ will have to wait until it receives $\mathbf{g}+\mathbf{b}$ messages before it receives $\mathbf{g}$ uncorrupted messages and it will have to try to decode $\binom{\mathbf{g}+\mathbf{b}}{\mathbf{g}}$ combinations of $\mathbf{g}$ pieces before it finds $\mathbf{g}$ pieces sent by non-Byzantine replicas.

The only way for Byzantine replicas to subvert the learning step is by finding a value w, $w \neq \mathbb{B}$, with $\texttt{hash}(w) = \texttt{hash}(\mathbb{B})$. As we assumed that $\texttt{hash}(\cdot)$ is a *collision-resistant hash function*, it is exceedingly hard for the Byzantine replicas to find such a value w. Furthermore, as $\mathbf{g}$ pieces are used during decoding and $\mathbf{g} > \mathbf{b}$, the Byzantine replicas not only need to find w, but must also find a way to encode w such that it can be reconstructed using pieces provided by one or more non-faulty replicas. Hence, assuming reasonable limits on the computational resources, the Byzantine replicas are unable to subvert the learning step.[1] ◀

Notice that if the system has no Byzantine replicas ($\mathbf{b} = 0$), then the checksums can be omitted entirely, as every set of $\mathbf{g}$ pieces will decode to the searched-for block of update decisions. This strongly reduces the computational costs for the learner. By combining Theorem 3.2 and Theorem 3.4, we obtain:

▶ **Corollary 3.5.** *Consider the learner* $\textsc{l} \in \mathfrak{L}$ *and replica* $\textsc{r} \in \mathcal{G}$*. If* $\mathbf{g} > \mathbf{b}$*, then the delayed-replication algorithm with simple checksums guarantees*

1. $\textsc{l}$ *will learn the update journal* $\mathbb{J}_{\mathfrak{R}}$*;*
2. $\textsc{l}$ *will receive at most* $|\mathbb{J}_{\mathfrak{R}}|$ *messages with a total size of* $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|(\mathbf{n}/\mathbf{g}))$*;*
3. $\textsc{l}$ *will only need worst-case* $\binom{\mathbf{g}+\mathbf{b}}{\mathbf{g}}(|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n})$ *information dispersal decode steps; and*
4. $\textsc{r}$ *will sent at most* $|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n}$ *messages to* $\textsc{l}$ *with a total size of* $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|/\mathbf{g})$*.*

To conclude, we notice that Byzantine replicas that send corrupted messages are easily detectable by the learners. After a learner $\textsc{l} \in \mathfrak{L}$ has decoded $\mathbf{g}$ pieces into a valid block $\mathbb{B}$, it can simply encode this block and determine the exact value of each encoded piece a replica should have sent to $\textsc{l}$. Hence, after trying to subvert a learning step of $\textsc{l}$, Byzantine replicas can be recognized and be eliminated from future considerations. When the set of Byzantine replicas is relatively stable over time, we can use this approach towards detecting Byzantine behavior at $\textsc{l}$ to prevent the worst-case upper bound on the number of information dispersal decode steps, $\binom{\mathbf{g}+\mathbf{b}}{\mathbf{g}}$, from happening repeatedly.

3.4 The Information Learning Step with Tree Checksums

In the previous section, we have shown how learners can reliably reconstruct $\mathbb{J}_{\mathfrak{R}}$ in the presence of Byzantine replicas. In theory, this provided approach has a high computational overhead for the learners due to the worst-case combinatorics involved. In this section, we explorer a different checksum scheme that allows the learners to discard any invalid messages with minimal effort, this with only a low communication overhead for the replicas and learners involved. Consequently, the learners can directly select the appropriate messages and perform only a single information dispersal decode step. Inspired by the *fingerprints* of Alon et al. [2, 3], we base our checksum scheme on *Merkle trees* [51].

▶ **Definition 3.6.** *Consider a block of* $\mathbf{n}$ *update decisions* $\mathbb{B} = \mathbb{J}_{\mathfrak{R}}[(j-1)\mathbf{n} : j\mathbf{n}]$*. The replica* $\textsc{r}$ *with* $\mathsf{id}(\textsc{r}) = i$, $0 \leq i < \mathbf{n}$*, should produce the* i*-th encoded piece* $\mathbb{B}_i = \texttt{slice}_{\textsc{r}}(\mathbb{B})$*. To simplify presentation, we assume that the total number of such pieces is a power-of-two (otherwise, we simply add* $\texttt{null}$*-pieces until we have a power-of-two number of pieces). A* Merkle tree *build over these pieces is a balanced binary tree constructed as follows:*

1. *The* i*-th leaf of the tree has the value* $\texttt{hash}(\mathbb{B}_i)$*.*
2. *The value of an internal node of which the left-child has value* w_1 *and the right-child has value* w_2 *is* $\texttt{hash}([w_1, w_2])$*.*

[1] A theoretical attack of this type can always be detected by the learner: this attack will yield at least two sets of $\mathbf{g}$ pieces that decode to values w and $\mathbb{B}$ with $w \neq \mathbb{B}$ and $\texttt{hash}(w) = \texttt{hash}(\mathbb{B})$.

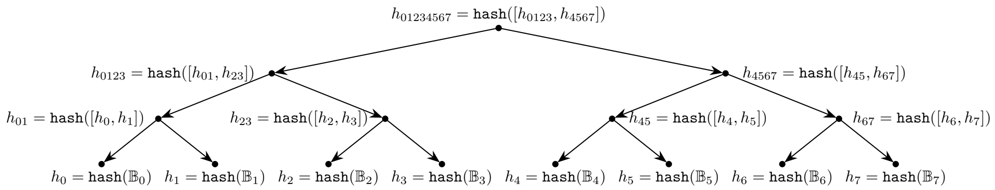

$h_{01234567} = \texttt{hash}([h_{0123}, h_{4567}])$
$h_{0123} = \texttt{hash}([h_{01}, h_{23}])$
$h_{4567} = \texttt{hash}([h_{45}, h_{67}])$
$h_{01} = \texttt{hash}([h_0, h_1])$
$h_{23} = \texttt{hash}([h_2, h_3])$
$h_{45} = \texttt{hash}([h_4, h_5])$
$h_{67} = \texttt{hash}([h_6, h_7])$
$h_0 = \texttt{hash}(\mathbb{B}_0)$
$h_1 = \texttt{hash}(\mathbb{B}_1)$
$h_2 = \texttt{hash}(\mathbb{B}_2)$
$h_3 = \texttt{hash}(\mathbb{B}_3)$
$h_4 = \texttt{hash}(\mathbb{B}_4)$
$h_5 = \texttt{hash}(\mathbb{B}_5)$
$h_6 = \texttt{hash}(\mathbb{B}_6)$
$h_7 = \texttt{hash}(\mathbb{B}_7)$

Figure 5 A *Merkle tree* over eight data pieces $\mathbb{B}_0, \dots, \mathbb{B}_7$. The leaf nodes are each labeled with the hash of a data piece, while every internal node is labeled with the hash of the value of its two children. The *tree checksum* for $\mathbb{B}_5$ is $\texttt{checksum}(\mathbb{B}_5) = [h_{01234567}, h_{0123}, h_{67}, h_4]$.

Notice that this construction is deterministic. Hence, every non-faulty replica will construct exactly the same Merkle tree for $\mathbb{B}$*. The* tree checksum *we propose for the i-th piece* $\mathbb{B}_i$*,* $\texttt{checksum}(\mathbb{B}_i)$*, consists of the value of the root of the Merkle tree and the values of the sibling of each node on the path from the root to the i-th leaf.*

We illustrate this further in the following example.

▶ **Example 3.7.** Assume $\mathbf{n} = 8$ and consider a block $\mathbb{B}$ that encodes into pieces $\mathbb{B}_0, \dots, \mathbb{B}_7$. The Merkle tree for $\mathbb{B}$ can be found in Figure 5. The tree checksum $\texttt{checksum}(\mathbb{B}_5)$ is obtained as follows. First, the path from the root of the tree to the 5-th leaf visits the nodes with values h_{4567}, h_{45}, and h_5. The node with value h_{4567} has the sibling with value h_{0123}; the node with value h_{45} has the sibling with value h_{67}; and, finally, the node with value h_5 has the sibling with value h_4. The root of the tree has value $h_{01234567}$. Hence, $\texttt{checksum}(\mathbb{B}_5) = [h_{01234567}, h_{0123}, h_{67}, h_4]$.

Next, we show that these tree checksums are sufficient to recognize messages corrupted by Byzantine replicas.

▶ **Theorem 3.8.** *Consider the learner* $\textsc{l} \in \mathfrak{L}$ *after it has received all messages sent by the information dispersal steps following the* ρ*-th decision in* $\mathbb{J}_{\mathfrak{R}}$*,* $\rho \geq 1$*. If* $\mathbf{g} > \mathbf{b}$*, then, after receiving these messages,* $\textsc{l}$ *can reconstruct the first* $\mathbf{n}\lfloor \rho/\mathbf{n} \rfloor$ *update decisions made by* $\mathfrak{R}$ *using only* $\lfloor \rho/\mathbf{n} \rfloor$ *information dispersal decode steps.*

Proof. Let $i = \mathbf{n}\lfloor \rho/\mathbf{n} \rfloor$ be the last round after which $\textsc{l}$ received update decisions. As in the proof of Theorem 3.4, we only focus on how $\textsc{l}$ can reconstruct the block $\mathbb{B}$ containing update decisions $(i-1)\mathbf{n}, \dots, i\mathbf{n} - 1$, this independent of the behavior of the Byzantine replicas. Every message sent by non-faulty replicas will include a valid tree checksums. Each of these checksums is constructed over the same Merkle tree. Consequently, each of these checksums share the same value for the root of the Merkle tree. Hence, using the reasoning of Theorem 3.4, $\textsc{l}$ can reliably learn the root value r of the Merkle tree after receiving at least $\mathbf{b} + 1$ messages with identical root values in their checksum.

Now consider the message (i, s_j, c_j) received from the replica $\textsc{r}$ with $\mathsf{id}(\textsc{r}) = j$. To determine whether this message is valid and uncorrupted, we first check whether the root value in c_j matches r. If this check fails, we can already discard the message. Next, we compute the hash $\texttt{hash}(s_j)$ to obtain the value of the j-th leaf in the Merkle tree. We observe that c_j contains the value of the sibling of the j-th leaf. Hence, we can construct the value of the parent p of the j-th leaf. This can be repeated: for any ancestor of the j-th leaf, c_j also contains the value of the sibling of this ancestor. Hence, one can recompute the value of every ancestor of the j-th leaf based on the value s_j. When done, one will obtain the root value r when the message is valid and uncorrupted. If any other value is obtained, then the message must be corrupted and one can discard the message.

As with the simple checksums, the only way in which Byzantine replicas can subvert the learning step is by finding hash collisions. Hence, assuming reasonable limits on the computational resources, the Byzantine replicas are unable to subvert the learning step. ◀

To further clarify the verification of messages, we illustrate how the verification process of the proof of Theorem 3.8 works:

▶ **Example 3.9.** Consider the situation of Example 3.7. Let $\textsc{l}$ be a learner that already determined that the root value is $h_{01234567}$. At some point, $\textsc{l}$ receives a message containing $\mathbb{B}'_5$ and the checksum $\texttt{checksum}(\mathbb{B}'_5) = [w_1, w_2, w_3, w_4]$ from replica $\textsc{r}$ with $\mathsf{id}(\textsc{r}) = 5$. The learner checks whether $w_1 = h_{01234567}$, as otherwise the message is discarded. We assume $w_1 = h_{01234567}$. Next, $\textsc{l}$ computes

$$\begin{aligned} h'_5 &= \texttt{hash}(\mathbb{B}'_5); \\ h'_{45} &= \texttt{hash}([w_4, h'_5]); \\ h'_{4567} &= \texttt{hash}([h'_{45}, w_3]); \\ h'_{01234567} &= \texttt{hash}([w_2, h_{4567}]). \end{aligned}$$

If $\mathbb{B}'_5 = \mathbb{B}_5$, $w_4 = h_4$, $w_3 = h_{67}$, and $w_2 = h_{0123}$, then $h'_{01234567} = h_{01234567}$ and the message received from $\textsc{r}$ is valid and uncorrupted. In any other case, the resulting value $h'_{01234567}$ will not match $h_{01234567}$ and the message is discarded.

In Theorem 3.8, we analyzed the computational complexity of the information learning step with tree checksums in terms of the number of information dispersal decode steps. As we show in Example 3.9, one also needs to validate the correctness of each message via its tree checksum, for which $\log(\mathbf{n})$ hashes need to be computed. In practice, the information dispersal decode steps are much more costly than these validation steps (this is especially true when using modern processors that provide hardware acceleration for hashing). Hence, in our analysis, we only focus on the number of information dispersal decode steps.

Notice that, for any block $\mathbb{B}$, we obtain $\|\texttt{checksum}(\mathbb{B})\| = \Theta(\log(\mathbf{n}))$, this independent of $\|\mathbb{B}\|$. By combining Theorem 3.2 and Theorem 3.8, we obtain:

▶ **Corollary 3.10.** *Consider the learner $\textsc{l} \in \mathfrak{L}$ and replica $\textsc{r} \in \mathcal{G}$. If $\mathbf{g} > \mathbf{b}$, then the delayed-replication algorithm with tree checksums guarantees*

1. *$\textsc{l}$ will learn the update journal $\mathbb{J}_{\mathfrak{R}}$;*
2. *$\textsc{l}$ will receive at most $|\mathbb{J}_{\mathfrak{R}}|$ messages with a total size of $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|(\mathbf{n}/\mathbf{g}) + |\mathbb{J}_{\mathfrak{R}}| \log(\mathbf{n}))$;*
3. *$\textsc{l}$ will only need at most $|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n}$ information dispersal decode steps; and*
4. *$\textsc{r}$ will sent at most $|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n}$ messages to $\textsc{l}$ with a total size of $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|/\mathbf{g} + (|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n}) \log(\mathbf{n}))$.*

4 Use Case: Improving Permissioned Blockchains

In this paper, we introduced the *Byzantine learner problem* and the *delayed-replication algorithm*, this to support the *hierarchical architecture* for fault-tolerant and federated data management systems that we envisioned in Section 1.1. Our hierarchical architecture relies on a Byzantine cluster to manage the data. Typically, such Byzantine clusters are implemented by permissioned fully-replicated blockchains that use traditional consensus techniques. Next, we illustrate how the delayed-replication algorithm can be generalized to improve on such permissioned blockchains by introducing *scalable shared storage* instead of full replication and by reducing the cost of *update decision making*. Consequently, our techniques also improves the applicability of permissioned blockchains to extend database systems towards fault-tolerant and federated data management.

4.1 Towards Scalable Shared Storage

As noted in the Introduction, state-of-the-art systems use fully replicated designs in which every replica in a cluster $\mathfrak{R}$ maintains the full update journal $\mathbb{J}_{\mathfrak{R}}$. Replicas $\textsc{r} \in \mathfrak{R}$ typically only need the current view $\mathbb{V}$ of the data to make update decisions, however, and do not need access to the full history of all updates stored in $\mathbb{J}_{\mathfrak{R}}$. Hence, a fully replicated design is unnecessary costly and limits scalability. Fortunately, the delayed-replication algorithm already showed that full replication of $\mathbb{J}_{\mathfrak{R}}$ is unnecessary to guarantee the ability to recover and learn $\mathbb{J}_{\mathfrak{R}}$. Instead of storing all of $\mathbb{J}_{\mathfrak{R}}$ at each $\textsc{r}$, each replica $\textsc{r}$ can simply processes each block $\mathbb{B}$ of $\mathbf{n}$ journal updates, compute $\texttt{slice}_{\textsc{r}}(\mathbb{B})$, and only keep this encoded piece around. This lowers the storage cost for $\mathbb{J}_{\mathfrak{R}}$ from $\|\mathbb{J}_{\mathfrak{R}}\|$ per replica to $\|\mathbb{J}_{\mathfrak{R}}\|/\mathbf{g}$ per replica, which makes the storage capacity of $\mathfrak{R}$ scalable with the number of non-faulty replicas without hampering the availability of $\mathbb{J}_{\mathfrak{R}}$ for replica recovery and for external learners.

▶ **Example 4.1.** Consider a federated inventory management system $(\mathfrak{R}, \mathfrak{L})$ used by several companies to keep track of their inventories and of transactions between them. To decide upon the updates on the data, replicas in $\mathfrak{R}$ only need to be able to validate updates: e.g., a transfer of ownership from company A to company B of a product is only a valid update if A originally owned the product. Hence, for validation, it is not necessary that replicas in $\mathfrak{R}$ maintain full copies of the journal $\mathbb{J}_{\mathfrak{R}}$, they only need the status of the current inventory, a much smaller dataset. Other tasks such as periodic analytics and data provenance will need read-only access to the full history of the data, which they can obtain as learners via the delayed-replication algorithm.

To further illustrate the necessity of storage scalability in blockchains, we only have to look at the permissionless Bitcoin blockchain. The size of the Bitcoin *ledger*, which represents a fully-replicated journal of financial transactions, is currently exceeding 256 GB and has grown with 59 GB over the last year. As noted in the introduction, Bitcoin is only able to process 7 transactions per second whereas Visa already processes 2000 transactions per second on average [61]. The permissioned blockchains our work focusses on can easily process hundreds to thousands transactions per second, as already exemplified by the BFS system in 2002 [14, 15]. Hence, the size of the journal maintained by permissioned blockchains can grow even more rapidly. We conclude:

▶ **Proposition 4.2.** *Let $\mathfrak{R}$ be a Byzantine cluster with update journal $\mathbb{J}_{\mathfrak{R}}$, current data view $\mathbb{V}$, and in which only $\mathbb{V}$ is necessary to make update decisions. If $\mathbf{g} > \mathbf{b}$, then the delayed-replication algorithm can provide storage scalability with these guarantees:*

1. *If simple checksums are used, then the storage cost per replica $\textsc{r} \in \mathfrak{R}$ is reduced from $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\| + \|\mathbb{V}\|)$ to $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|/\mathbf{g} + \|\mathbb{V}\|)$.*
2. *If tree checksums are used, then the storage cost per replica $\textsc{r} \in \mathfrak{R}$ is reduced from $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\| + \|\mathbb{V}\|)$ to $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|/\mathbf{g} + (|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n}) \log(\mathbf{n}) + \|\mathbb{V}\|)$.*

Proof. These results follow directly from Corollaries 3.5 and 3.10. ◀

4.2 Improved Checkpoints in Byzantine Consensus

Next, we will show how the delayed-replication algorithm can be used internally in Byzantine clusters to reduce the cost of decision making. Typical permissioned blockchains use *consensus protocols* to coordinate making update decisions [4, 8, 9, 10, 14, 15, 17, 28, 34, 42, 44, 45, 48, 69, 71]. Most practical consensus protocols can be traced back to the influential design of the *Practical Byzantine Fault Tolerance protocol* (`Pbft`) of Castro et al. [14, 15].

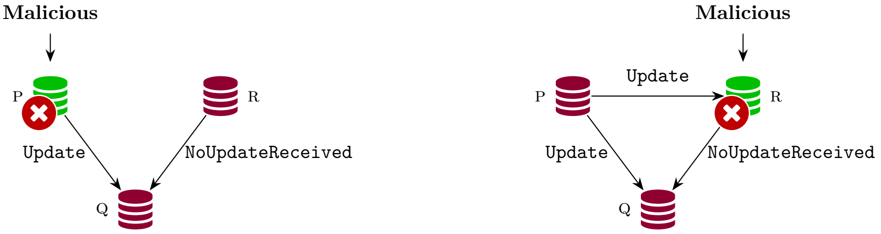


Figure 6 Two cases of possible malicious behavior. On the left, the primary P is malicious. On the right, the replica R is malicious. Unfortunately, the non-faulty replica Q receives the same set of messages in both cases and, hence, cannot determine which replica is malicious in which case.

In `Pbft`, a replica is elected *primary* and is in charge of coordinating update decision making among all replicas. To allow other replicas to determine whether the primary, which could be Byzantine, is coordinating correctly, `Pbft` employs two phases of broadcast-based communication among all replicas before any update decision is committed. This communication ensures that update decisions are only committed if a majority of all non-faulty replicas are aware of these decisions. Unfortunately, a malicious primary can keep some non-faulty replicas *in the dark* by not sending them any update decisions. The remaining Byzantine replicas can cover for this malicious behavior by acting as non-faulty replicas. Such an attack cannot be reliably detected by the other replicas in the system, as the following example illustrates.

▶ **Example 4.3.** Let $\mathfrak{R}$ be a Byzantine cluster with $\mathbf{n} = 3\mathbf{b} + 1$. Let P $\in \mathfrak{R}$ be the elected primary, let R $\in \mathfrak{R}$ be another replica, and let Q $\in \mathcal{G}$ be a non-faulty replica. Assume that a correct primary will send the same updates to all replicas. Consider the following two cases:

1. We have R $\in \mathcal{G}$ and P $\in \mathcal{B}$. The primary sends updates via `Update` messages, except that it excludes R. The replica R detects this, as it does not receive any messages, and notifies Q that the primary is malicious via a `NoUpdateReceived` message.
2. We have R $\in \mathcal{B}$ and P $\in \mathcal{G}$. Independent of the actions of the primary, R notifies Q that the primary is malicious via a `NoUpdateReceived` message.

We have sketched these two cases in Figure 6. In both cases, Q receives exactly the same set of messages from P and R. Consequently, Q cannot determine which of the replicas is malicious and which of the replicas is non-faulty.

The issue of replicas being left in the dark is faced by not only `Pbft`, but also by many other practical primary-backup protocols. The typical way to assure that all replicas eventually learn the update decisions made, even when the primary is malicious, is by using periodical checkpoints. Unfortunately, the usual checkpoint protocols employed use broadcast-based primitives with very high communication complexity. Moreover, checkpoint protocols typically have a pull-based component, which makes them vulnerable to the attacks illustrated in Example 2.5. Fortunately, we can employ the push-based delayed-replication algorithm to provide checkpoints with low costs for all replicas involved. To do so, we model any replica left in the dark by a malicious primary as *crashed*. In the worst case, `Pbft` allows for a situation in which $\mathbf{g} = \mathbf{b}+1$ and $\mathbf{c} = \mathbf{b}$ (the maximum number of non-Byzantine replicas a malicious primary can keep in the dark without being detected), making $\mathbf{n} \geq 3\mathbf{b}+1$. Hence, in all situations our delayed-replication algorithm can be employed as a checkpoint protocol by making all replicas in $\mathfrak{R}$ listeners. We conclude:

▶ **Proposition 4.4.** *Let $\mathfrak{R}$ be a Byzantine cluster running* `Pbft`*. If $\mathbf{g} > \mathbf{b}$, then the delayed-replication algorithm can provide* `Pbft`*-style checkpoints with these guarantees:*

1. *If simple checksums are used, then every replica $\textsc{r} \in \mathfrak{R}$ will be able to learn $\mathbb{J}_{\mathfrak{R}}$ and will send and receive at most $|\mathbb{J}_{\mathfrak{R}}|$ messages with a combined size of $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|)$.*
2. *If tree checksums are used, then every replica $\textsc{r} \in \mathfrak{R}$ will be able to learn $\mathbb{J}_{\mathfrak{R}}$ and will send and receive at most $|\mathbb{J}_{\mathfrak{R}}|$ messages with a combined size of $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\| + |\mathbb{J}_{\mathfrak{R}}|\log(\mathbf{n}))$.*

Proof. We choose $\mathfrak{L} = \mathfrak{R}$ and we use the delayed-replication algorithm with $\mathbf{c} = \mathbf{b}$ and, as $\mathfrak{R}$ is running `Pbft`, $\mathbf{n} > 3\mathbf{b}$. Hence, $\mathbf{g} = \mathbf{n} - \mathbf{c} - \mathbf{b} = \mathbf{n} - 2\mathbf{b} \geq \mathbf{b} + 1$ non-faulty replicas will be senders in the delayed-replication algorithm, guaranteeing success.

Next, we consider the number of messages sent and received. First, consider the case using simple checksums. Let $\textsc{r} \in \mathcal{G}$ be a non-faulty replica that is not left in the dark. Applying the results of Corollaries 3.5, we learn that replica $\textsc{r}$ sends at most $|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n}$ messages to each replica with a total size of $\mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|/\mathbf{g})$. As $\mathbf{n} = |\mathfrak{R}|$, $\textsc{r}$ will send $\mathbf{n}(|\mathbb{J}_{\mathfrak{R}}|/\mathbf{n}) = |\mathbb{J}_{\mathfrak{R}}|$ messages with a total size of at most $d\mathbf{n}(\|\mathbb{J}_{\mathfrak{R}}\|/\mathbf{g})$, for some constant d. We have $\mathbf{n} > 3\mathbf{b}$ and $\mathbf{g} = \mathbf{n} - 2\mathbf{b}$. Hence, $\mathbf{n} = 3\mathbf{b} + j$ and $\mathbf{g} = \mathbf{n} - 2\mathbf{b} = \mathbf{b} + j$ for some j, $j \geq 1$. We have $\mathbf{n}/\mathbf{g} = (3\mathbf{b} + j)/(\mathbf{b} + j) = 1 + 2\mathbf{b}/(\mathbf{b} + j)$. As $j \geq 1$, we have $0 \leq 2\mathbf{b}/(\mathbf{b} + j) < 2$ and $\mathbf{n}/\mathbf{g} \leq 3$. We conclude that the total size of all messages send by $\textsc{r}$ is upper bounded by $3d\|\mathbb{J}_{\mathfrak{R}}\| = \mathcal{O}(\|\mathbb{J}_{\mathfrak{R}}\|)$. In a similar manner, we can derive the same upper bounds on the number and size of the messages received by $\textsc{r}$. For the case using tree checksums, we apply the results of Corollary 3.10 to the above reasoning, which leads to only adding a cost of $\mathcal{O}(\log(\mathbf{n}))$ to each message sent and received. ◀

Since the introduction of `Pbft`, many improvements on its design have been proposed. Recently, there has been a surge in protocols that aim at bringing down the communication cost of the normal-case operations of `Pbft` from a *quadratic amount* per update decision to a *linear amount*, which vastly improves the scalability of consensus. Examples include `HotStuff` [75], `LinBFT` [74], and `SBFT` [33]. These examples all use *threshold signatures* [66] to summarize confirmation of any decision by the majority of all replicas in a constant-sized signature – which eliminates the need for broadcast-based quadratic communication among all replicas. None of the current approaches satisfactory deal with recovery of replicas that are left in the dark, however. Hence, we believe that our highly-efficient delayed-replication algorithm can fill in the checkpoint gap in such linear designs.

5 Related Work

There is an abundant literature on distributed systems, distributed scalable storage (e.g., via information dispersal), and on fault-tolerant fully-replicated systems (e.g. [12, 60, 69, 70]). In this paper, we primarily focused on bridging the gap between, on the one hand, fault-tolerant systems and, on the other hand, scalable distributed systems.

Learners in fault-tolerant systems. `Paxos`, a consensus protocol that can be used to make reliable update decisions in a cluster with only crash failures, and several `Paxos`-like protocols have a concept of *learners* [46, 47, 49]. As the name suggests, these `Paxos`-learners will learn all update decisions made by the cluster, not unlike the learners we propose. In `Paxos`, these learners are also crucial to determine whether consensus is reached, are deeply involved in the consensus protocol, cannot be arbitrarily selected, and perform significant amounts of communication, however. This makes the architecture of `Paxos` incomparable with the hierarchical architecture we propose.

Most other consensus protocols, especially those based on the *Practical Byzantine Fault Tolerance protocol* (`Pbft`) of Castro et al. [14, 15], do not distinguish between the roles of replicas in a Byzantine cluster. In `Pbft`, some *read-only* optimizations are considered, but even these optimizations require participation of all replicas involved. Hence, the approach of `Pbft` towards read-only data processing is non-scalable, whereas our hierarchical architecture benefits from the addition of non-faulty replicas to the Byzantine cluster.

The HyperLedger permissioned blockchain fabric does utilize a hierarchical design similar to what we propose [6]. HyperLedger distinguishes between, on the one hand, endorsers and orderers that coordinate data updates, and, on the other hand, peers that only learn of data updates. Currently, HyperLedger relies on Apache Kafka [26] to provide only crash-tolerant ordering and to broadcast updates to peers. The approach we propose – by using the delayed-replication algorithm – is not only able to tolerate an arbitrary number of crashes, but also addresses Byzantine behavior. Furthermore, our approach is highly scalable and requires only a minimum of communication. Finally, our approach enables a way towards permissioned blockchains with scalable storage, which is not provided by HyperLedger.

Finally, the Byzantine learner problem we study in this paper differs from the *cluster-sending problem* of Hellings et al. [37]. On the one hand, we provide in this work techniques to stream sequences of data updates from a single Byzantine cluster to learners, this with minimal communication costs in terms of the data exchanged. On the other hand, the cluster-sending problem of Hellings et al. [37] focusses on the problem of sending a single value between two Byzantine clusters with minimal communication costs in terms of the number of messages exchanged.

Information dispersal and scalable storage. `IDA`, the information dispersal algorithm we utilize in the delayed-replication algorithm, was proposed by Rabin [63] to provide reliable load-balanced storage and communication in a distributed setting. Alon et al. [2, 3] expanded `IDA` towards recovery of failures by adapting the scheme used in `IDA` towards recognizing faulty encoded pieces. Unfortunately, the methods employed by Alon et al. always introduce a space overhead per participant. We build upon these information dispersal techniques by using them to solve the *Byzantine learner problem* and we showed how these techniques can be used to resolve current issues in state-of-the-art permissioned blockchains that provide fault-tolerant and federated data management. Moreover, our results structurally improve on the results of Rabin and Alon et al. via the delayed-replication algorithm with simple checksums, which enables Byzantine fault-tolerant communication and storage without any space overhead.

6 Conclusions and Future Work

In this paper, we studied the *Byzantine learner problem* – the problem of efficiently distributing a sequence of data updates made by replicas in a Byzantine cluster to an arbitrary number of learners. As our main result, we proposed the *delayed-replication algorithm* to address this Byzantine learner problem. Our algorithm is coordination-free, equally distributes communication costs among all replicas, and leverages information dispersal to achieve Byzantine learning with minimal communication costs. Our delayed-replication algorithm opens the door to hierarchical fault-tolerant and federated database systems that can effectively deal with big read-only workloads, e.g., by running complex data processing tasks on individual specialized learners and by providing trusted read-only proxies close to end-users for fast query answering. Moreover, we showed that the delayed-replication algorithm and its underlying techniques open the door for the development of new high-performance fault-tolerant database systems by improving the design of existing permissioned blockchain-based database systems.

Our techniques are only a first step toward developing fault-tolerant, reliable, scalable, and high-performance database systems and permissioned blockchains. To further these developments, we need not only support big read-only workloads and storage scalability, but also improve on other areas. To do so, we see several avenues of future research and development:

1. Our techniques can be used to improve fault-tolerant and federated data management by reducing the cost of read-only workloads and by introducing scalable shared storage. We did not yet address the scalability of decision making (e.g., write workloads), however. To improve decision making, we are interested in the development of efficient decision making techniques in Byzantine settings that – for performance reasons – provides less strict guarantees than traditional consensus-based techniques. To further aid scalability of decision making, we are also interested in developing further non-fully-replicated designs, e.g., by incorporating fault-tolerant sharding.
2. Our design is primarily intended to reduce the cost of read-only workloads that require access to the full history of changes. Examples of such workloads include analytics, data provenance, machine learning, visualization, and read-only proxies. Besides these workloads, there are also many smaller read-only workloads, e.g., one-off relational querying. Current fault-tolerant approaches toward such workloads remain non-scalable, as they require the independent execution of such queries by all replicas in the Byzantine cluster. We are interested in the development of techniques that can lift this burden on scalability.

References

1 2ndQuadrant. Postgres-XL: Open source scalable SQL database cluster. URL: https://www.postgres-xl.org/.

2 Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, and Julien Stern. Scalable Secure Storage When Half the System Is Faulty. *Information and Computation*, 174(2):203–213, 2002. doi:10.1006/inco.2002.3148.

3 Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, and Julien Stern. Addendum to "Scalable secure storage when half the system is faulty". *Information and Computation*, 205(7):1114–1116, 2007. doi:10.1016/j.ic.2006.02.007.

4 Yair Amir, Claudiu Danilov, Danny Dolev, Jonathan Kirsch, John Lane, Cristina Nita-Rotaru, Josh Olsen, and David Zage. Steward: Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks. *IEEE Transactions on Dependable and Secure Computing*, 7(1):80–93, 2010. doi:10.1109/TDSC.2008.53.

5 Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. CAPER: A cross-application permissioned blockchain. *Proceedings of the VLDB Endowment*, 12(11):1385–1398, 2019. doi:10.14778/3342263.3342275.

6 Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, pages 30:1–30:15. ACM, 2018. doi:10.1145/3190508.3190538.

7 GSM Association. Blockchain for Development: Emerging Opportunities for Mobile, Identity and Aid, 2017. URL: https://www.gsma.com/mobilefordevelopment/wp-content/uploads/2017/12/Blockchain-for-Development.pdf.

8 Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The Next 700 BFT Protocols. *ACM Transactions on Computer Systems*, 32(4):12:1–12:45, 2015. doi:10.1145/2658994.

9 Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: Redundant byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE, 2013. doi:10.1109/ICDCS.2013.53.

10 Christian Berger and Hans P. Reiser. Scaling Byzantine Consensus: A Broad Analysis. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, SERIAL'18, pages 13–18. ACM, 2018. doi:10.1145/3284764.3284767.

11 Burkhard Blechschmidt. Blockchain in Europe: Closing the Strategy Gap. Technical report, Cognizant Consulting, 2018. URL: https://www.cognizant.com/whitepapers/blockchain-in-europe-closing-the-strategy-gap-codex3320.pdf.

12 Christian Cachin and Marko Vukolic. Blockchain Consensus Protocols in the Wild (Keynote Talk). In *31st International Symposium on Distributed Computing*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.1.

13 Michael Casey, Jonah Crane, Gary Gensler, Simon Johnson, and Neha Narula. The Impact of Blockchain Technology on Finance: A Catalyst for Change. Technical report, International Center for Monetary and Banking Studies, 2018. URL: https://www.cimb.ch/uploads/1/1/5/4/115414161/geneva21_1.pdf.

14 Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186. USENIX Association, 1999.

15 Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002. doi:10.1145/571637.571640.

16 Christie's. Major Collection of the Fall Auction Season to be Recorded with Blockchain Technology, 2018. URL: https://www.christies.com/presscenter/pdf/9160/RELEASE_ChristiesxArtoryxEbsworth_9160_1.pdf.

17 Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 153–168. USENIX Association, 2009.

18 Cindy Compert, Maurizio Luinetti, and Bertrand Portier. Blockchain and GDPR: How blockchain could address five areas associated with GDPR compliance. Technical report, IBM Security, 2018. URL: https://public.dhe.ibm.com/common/ssi/ecm/61/en/61014461usen/security-ibm-security-solutions-wg-white-paper-external-61014461usen-20180319.pdf.

19 Oracle Corporation. Maximize Availability with Oracle Database 18c. URL: https://www.oracle.com/technetwork/database/availability/maximum-availability-wp-18c-4403435.pdf.

20 Oracle Corporation. MySQL - MySQL 8.0 reference manual: 17 replication. URL: https://dev.mysql.com/doc/refman/8.0/en/replication.html.

21 Alex de Vries. Bitcoin's Growing Energy Problem. *Joule*, 2(5):801–805, 2018. doi:10.1016/j.joule.2018.04.016.

22 Microsoft Docs. SQL Server replication. URL: https://docs.microsoft.com/en-us/sql/relational-databases/replication/sql-server-replication.

23 Wayne W. Eckerson. Data quality and the bottom line: Achieving Business Success through a Commitment to High Quality Data. Technical report, The Data Warehousing Institute, 101communications LLC., 2002.

24 Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. BlockchainDB: A shared database on blockchains. *Proceedings of the VLDB Endowment*, 12(11):1597–1609, 2019. `doi:10.14778/3342263.3342636`.

25 Muhammad El-Hindi, Martin Heyden, Carsten Binnig, Ravi Ramamurthy, Arvind Arasu, and Donald Kossmann. BlockchainDB – towards a shared database on blockchains. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1905–1908. ACM, 2019. `doi:10.1145/3299869.3320237`.

26 The Apache Software Foundation. Apache Kafka: A distributed streaming platform. URL: `https://kafka.apache.org/`.

27 Lan Ge, Christopher Brewster, Jacco Spek, Anton Smeenk, and Jan Top. Blockchain for agriculture and food: Findings from the pilot study. Technical report, Wageningen University, 2017. URL: `https://www.wur.nl/nl/Publicatie-details.htm?publicationId=publication-way-353330323634`.

28 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017. `doi:10.1145/3132747.3132757`.

29 William J. Gordon and Christian Catalini. Blockchain Technology for Healthcare: Facilitating the Transition to Patient-Driven Interoperability. *Computational and Structural Biotechnology Journal*, 16:224–230, 2018. `doi:10.1016/j.csbj.2018.06.003`.

30 Jim Gray. Notes on Data Base Operating Systems. In *Operating Systems, An Advanced Course*, pages 393–481. Springer-Verlag, 1978. `doi:10.1007/3-540-08755-9_9`.

31 Andy Greenberg. The Untold Story of NotPetya, the Most Devastating Cyberattack in History, 2018. URL: `https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/`.

32 The PostgreSQL Global Development Group. PostgreSQL documentation: Chapter 26. high availability, load balancing, and replication. URL: `https://www.postgresql.org/docs/current/static/high-availability.html`.

33 Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: a scalable and decentralized trust infrastructure, 2019. URL: `https://arxiv.org/abs/1804.01626`.

34 Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. Brief Announcement: Revisiting Consensus Protocols through Wait-Free Parallelization. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:3. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.DISC.2019.44`.

35 Suyash Gupta and Mohammad Sadoghi. *Blockchain Transaction Processing*, pages 1–11. Springer International Publishing, 2018. `doi:10.1007/978-3-319-63962-8_333-1`.

36 Suyash Gupta and Mohammad Sadoghi. EasyCommit: A non-blocking two-phase commit protocol. In *Proceedings of the 21st International Conference on Extending Database Technology*. Open Proceedings, 2018. `doi:10.5441/002/edbt.2018.15`.

37 Jelle Hellings and Mohammad Sadoghi. Brief Announcement: The Fault-Tolerant Cluster-Sending Problem. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:3. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.DISC.2019.45`.

38 Maurice Herlihy. Blockchains from a Distributed Computing Perspective. *Communications of the ACM*, 62(2):78–85, 2019. `doi:10.1145/3209623`.

39 Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data Quality and Record Linkage Techniques*. Springer New York, 2007. `doi:10.1007/0-387-69505-2`.

40 Matt Higginson, Johannes-Tobias Lorenz, Björn Münstermann, and Peter Braad Olesen. The promise of blockchain. Technical report, McKinsey&Company,

2017. URL: https://www.mckinsey.com/industries/financial-services/our-insights/the-promise-of-blockchain.

41 Maged N. Kamel Boulos, James T. Wilson, and Kevin A. Clauson. Geospatial blockchain: promises, challenges, and scenarios in health and healthcare. *International Journal of Health Geographics*, 17(1):1211–1220, 2018. doi:10.1186/s12942-018-0144-x.

42 Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. CheapBFT: Resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 295–308. ACM, 2012. doi:10.1145/2168836.2168866.

43 Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2nd edition, 2014.

44 Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, pages 45–58. ACM, 2007. doi:10.1145/1294261.1294267.

45 Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Transactions on Computer Systems*, 27(4):7:1–7:39, 2009. doi:10.1145/1658357.1658358.

46 Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks (1976)*, 2(2):95–114, 1978. doi:10.1016/0376-5075(78)90045-4.

47 Leslie Lamport. Paxos Made Simple. *ACM SIGACT News, Distributed Computing Column 5*, 32(4):51–58, 2001. doi:10.1145/568425.568433.

48 Jian Liu, Wenting Li, Ghassan O. Karame, and N. Asokan. Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing. *IEEE Transactions on Computers*, 68(1):139–151, 2019. doi:10.1109/TC.2018.2860009.

49 Jean-Philippe Martin and Lorenzo Alvisi. Fast Byzantine Consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006. doi:10.1109/TDSC.2006.35.

50 Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1st edition, 1996.

51 Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, pages 369–378. Springer Berlin Heidelberg, 1988. doi:10.1007/3-540-48184-2_32.

52 Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. URL: https://bitcoin.org/en/bitcoin-paper.

53 Arvind Narayanan and Jeremy Clark. Bitcoin's Academic Pedigree. *Communications of the ACM*, 60(12):36–45, 2017. doi:10.1145/3132259.

54 Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. Blockchain Meets Database: Design and Implementation of a Blockchain Relational Database. *Proceedings of the VLDB Endowment*, 12(11):1539–1552, 2019. doi:10.14778/3342263.3342632.

55 Faisal Nawab and Mohammad Sadoghi. Blockplane: A Global-Scale Byzantizing Middleware. In *35th International Conference on Data Engineering (ICDE)*, pages 124–135. IEEE, 2019. doi:10.1109/ICDE.2019.00020.

56 Dick O'Brie. Internet Security Threat Report: Ransomware 2017, An ISTR Special Report. Technical report, Symantec, 2017. URL: https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-ransomware-2017-en.pdf.

57 The Council of Economic Advisers. The Cost of Malicious Cyber Activity to the U.S. Economy. Technical report, Executive Office of the President of the United States, 2018. URL: https://www.whitehouse.gov/wp-content/uploads/2018/03/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf.

58 National Audit Office. Investigation: WannaCry cyber attack and the NHS, 2018. URL: https://www.nao.org.uk/report/investigation-wannacry-cyber-attack-and-the-nhs/.

59 Michael Okun. *Byzantine Agreement*, pages 255–259. Springer New York, 2016. doi:10.1007/978-1-4939-2864-4_60.

60 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer New York, 3th edition, 2011.

61 Michael Pisa and Matt Juden. Blockchain and Economic Development: Hype vs. Reality. Technical report, Center for Global Development, 2017. URL: https://www.cgdev.org/publication/blockchain-and-economic-development-hype-vs-reality.

62 PwC. Blockchain – an opportunity for energy producers and consumers?, 2016. URL: https://www.pwc.com/gx/en/industries/energy-utilities-resources/publications/opportunity-for-energy-producers.html.

63 Michael O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the ACM*, 36(2):335–348, 1989. doi:10.1145/62044.62050.

64 Thomas C. Redman. The Impact of Poor Data Quality on the Typical Enterprise. *Communications of the ACM*, 41(2):79–82, 1998. doi:10.1145/269012.269025.

65 Scott Ruoti, Ben Kaiser, Arkady Yerukhimovich, Jeremy Clark, and Robert Cunningham. Blockchain Technology: What is It Good For? *Communications of the ACM*, 63(1):46—-53, 2019. doi:10.1145/3369752.

66 Victor Shoup. Practical Threshold Signatures. In *Advances in Cryptology — EUROCRYPT 2000*, pages 207–220. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-45539-6_15.

67 Dale Skeen. A Quorum-Based Commit Protocol. Technical report, Cornell University, 1982.

68 Symantec. Internet Security Threat Report, Volume 32, 2018. URL: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf.

69 Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2001.

70 Maarten van Steen and Andrew S. Tanenbaum. *Distributed Systems*. Maarten van Steen, 3th edition, 2017. URL: https://www.distributed-systems.net/.

71 Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. Efficient Byzantine Fault-Tolerance. *IEEE Transactions on Computers*, 62(1):16–30, 2013. doi:10.1109/TC.2011.221.

72 Harald Vranken. Sustainability of bitcoin and blockchains. *Current Opinion in Environmental Sustainability*, 28:1–9, 2017. doi:10.1016/j.cosust.2017.04.011.

73 Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. EIP-150 revision. URL: https://gavwood.com/paper.pdf.

74 Yin Yang. LinBFT: Linear-communication byzantine fault tolerance for public blockchains, 2018. URL: https://arxiv.org/abs/1807.01829.

75 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019. doi:10.1145/3293611.3331591.

Integrity Constraints Revisited: From Exact to Approximate Implication

Batya Kenig
University of Washington, Seattle, WA, USA
batyak@cs.washington.edu

Dan Suciu
University of Washington, Seattle, WA, USA
https://homes.cs.washington.edu/~suciu/
suciu@cs.washington.edu

Abstract

Integrity constraints such as functional dependencies (FD), and multi-valued dependencies (MVD) are fundamental in database schema design. Likewise, probabilistic conditional independences (CI) are crucial for reasoning about multivariate probability distributions. The implication problem studies whether a set of constraints (antecedents) implies another constraint (consequent), and has been investigated in both the database and the AI literature, under the assumption that all constraints hold *exactly*. However, many applications today consider constraints that hold only *approximately*. In this paper we define an approximate implication as a linear inequality between the degree of satisfaction of the antecedents and consequent, and we study the *relaxation problem*: when does an exact implication relax to an approximate implication? We use information theory to define the degree of satisfaction, and prove several results. First, we show that any implication from a set of data dependencies (MVDs+FDs) can be relaxed to a simple linear inequality with a factor at most quadratic in the number of variables; when the consequent is an FD, the factor can be reduced to 1. Second, we prove that there exists an implication between CIs that does not admit any relaxation; however, we prove that every implication between CIs relaxes "in the limit". Finally, we show that the implication problem for differential constraints in market basket analysis also admits a relaxation with a factor equal to 1. Our results recover, and sometimes extend, several previously known results about the implication problem: implication of MVDs can be checked by considering only 2-tuple relations, and the implication of differential constraints for frequent item sets can be checked by considering only databases containing a single transaction.

2012 ACM Subject Classification Theory of computation → Database theory; Theory of computation → Database constraints theory

Keywords and phrases Integrity constraints, The implication problem

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.18

Related Version A full version of the paper is available at https://arxiv.org/abs/1812.09987.

Funding This work was supported by NSF under grants III-1614738 and IIS 1907997.

1 Introduction

Traditionally, integrity constraints are assertions about a database that are stated by the database administrator and enforced by the system during updates. However, in several applications of Big Data, integrity constraints are discovered, or mined in a database instance, as opposed to being asserted by the administrator [13, 34, 7, 3, 20]. For example, data cleaning can be done by first learning conditional functional dependencies in some reference data, then using them to identify inconsistencies in the test data [16, 7]. Causal reasoning [35, 28, 31] and learning sum-of-product networks [29, 11, 26] repeatedly discover conditional independencies in the data. Constraints also arise in many other domains, for example in the *frequent itemset*

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 18; pp. 18:1–18:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Summary of results: relaxation bounds for the implication $\Sigma \Rightarrow \tau$ for the sub-cones of Γ_n under various restrictions. (1) *General*; no restrictions to either Σ or τ (2) Σ is a set of saturated CIs and conditional entropies (i.e., MVDs+FDs in databases), and τ is a conditional entropy. (3) Σ is a set of saturated CIs and conditional entropies, τ is *any* CI (4) *Disjoint* integrity constraints. The terms in Σ are both saturated and *disjoint* (see definition 10 in Sec. 4), and τ is saturated.

Cone	Relaxation Bounds			
	General	MVDs+FDs $\Rightarrow$ FD	MVDs+FDs $\Rightarrow$ any	Disjoint MVDs+FDs $\Rightarrow$ MVD/FD
Γ_n	$(2^n)!$ (Thm. 21)	1 (Thm. 6)	$\frac{n^2}{4}$ (Thm. 6)	1 (Thm. 11)
Γ_n^*	∞ (Thm. 16)	1 (Thm. 6)	$\frac{n^2}{4}$ (Thm. 6)	1 (Thm. 11)
$\mathcal{P}_n$	1 (Thm. 23)	1 (Thm. 23)	1 (Thm. 23)	1 (Thm. 23)

problem (FIS) [22, 5], or as *measure based constraints* [32] in applications like Dempster-Shafer theory, possibilistic theory, and game theory (see discussion in [32]). In all these applications, quite often the constraints are learned from the data, and are not required to hold exactly, but it suffices if they hold only to a certain degree.

The classical *implication problem* asks whether a set of constraints, called the *antecedents*, logically imply another constraint called the *consequent*. In this setting, both antecedents and consequent are required to hold exactly, hence we refer to it as an *exact implication* (EI). The database literature has extensively studied the EI problem for integrity constraints and shown that the implication problem is decidable and axiomatizable for Functional Dependencies (FDs) and Multivalued Dependencies (MVDs) [23, 19, 1, 2], and undecidable for Embedded Multivalued Dependencies (EMVDs) [15]. The AI community has studied extensively the EI problem for Conditional Independencies (CI), which are assertions of the form $X \perp Y \mid Z$, stating that X is independent of Y conditioned on Z, and has shown that the implication problem is decidable and axiomatizable for saturated CIs [12] (where $XYZ =$ all variables), but not finitely axiomatizable in general [36]. In the FIS problem, a constraint like $X \rightarrow Y \vee Z \vee U$ means that every basket that contains X also contains at least one of Y, Z, U, and the implication problem here is also decidable and axiomatizable [33].

The Relaxation Problem. In this paper we consider a new problem, called the *relaxation problem*: if an exact implication holds, does an approximate implication hold too? For example, suppose we prove that a given set of FDs implies another FD, but the input data satisfies the antecedent FDs only to some degree: to what degree does the consequent FD hold on the database? An *approximate implication* (AI) is an inequality that (numerically) bounds the consequent by a linear combination of the antecedents. The relaxation problem asks whether we can convert an EI into an AI. When relaxation holds, then any inference system for proving exact implication, e.g. using a set of axioms or some algorithm, can be used to infer an approximate implication.

In order to study the relaxation problem we need to measure the degree of satisfaction of a constraint. In this paper we use Information Theory. This is the natural semantics for modeling CIs of multivariate distributions, because $X \perp Y \mid Z$ iff $I(X;Y|Z) = 0$ where I is the *conditional mutual information*. FDs and MVDs are special cases of CIs [21, 8, 38] (reviewed in Sec. 2.1), and thus they are naturally modeled using the information theoretic measure $I(X;Y|Z)$ or $H(Y|X)$; in contrast, EMVDs do not appear to have a natural interpretation using information theory, and we will not discuss them here. Several papers have argued that information theory is a suitable tool to express integrity constraints [21, 8, 38, 24, 13].

An exact implication (EI) becomes an assertion of the form $(\sigma_1 = 0 \wedge \sigma_2 = 0 \wedge \ldots) \Rightarrow (\tau = 0)$, while an approximate implication (AI) is a linear inequality $\tau \leq \lambda \cdot \left(\sum \sigma_i\right)$, where $\lambda \geq 0$, and $\tau, \sigma_1, \sigma_2, \ldots$ are information theoretic measures. We say that a class of constraints can be *relaxed* if EI implies AI; we also say that it λ-relaxes, when we want to specify the factor λ in the AI. We notice an AI always implies EI.

Results. We make several contributions, summarized in Table 1. We start by showing in Sec. 4 that MVDs+FDs admit an $n^2/4$-relaxation, where n is the number of variables. When the consequent is an FD, we show that implication admits a 1-relaxation. Thus, whenever an exact implication holds between MVD+FDs, a simple linear inequality also holds between their associated information theoretic terms. In fact, we prove a stronger result that holds for CIs in general, which implies the result for MVDs+FDs. In addition, under some mild syntactic restrictions to the antecedents, we strengthen the result from a $n^2/4$-relaxation to a 1-relaxation, even when the consequent is an MVD; we leave open the question whether 1-relaxation exists in general.

So far, we have restricted ourselves to saturated or conditional CIs (which correspond to MVDs or FDs). In Sec. 5 we remove any restrictions, and prove a negative result: there exists an EI that does not relax (Eq. (9), based on an example in [17]). Nevertheless, we show that every EI can be relaxed to an AI plus an error term, which can be made arbitrarily small, at the cost of increasing the factor λ. This result implies that *every* EI can be proven from some inequality, corresponding to the AI associated to the EI, plus an error term. In fact, the EI in Eq. (9) follows from an inequality by Matúš [25], which is precisely the associated AI plus an error term; our result shows that every EI can be proven in this style.

Next, we consider two restrictions, which are commonly used in model theory. First, in Sec. 6 we restrict the class of *axioms* used to prove implications, to Shannon's inequalities (monotonicity and submodularity, reviewed in Sec. 2.2). In general, Shannon's inequalities are sound but incomplete for proving exact and approximate implications that hold for all probability distributions [41, 42], but they are complete for deriving inequalities that hold for all polymatroids [40]. We show that they are also complete for saturated+conditional constraints (as we show in Sec 4), and for measure-based constraints [32] (Sec. 7). We prove that every exact implication that holds for all polymatroids relaxes to an approximate implication, and prove an upper bound $\lambda \leq (2^n)!$, and a lower bound $\lambda \geq 3$; the exact bound remains open. Second, in Sec. 7 we restrict the class of *models* used to check an implication, to probability distributions with exactly 2 outcomes (tuples), each with probability 1/2; we justify this shortly. We prove that, under this restriction, the implication problem has a 1-relaxation. Restricting the models leads to a complete but unsound method for checking general implication, however this method is sound for saturated+conditional (as we show in Sec 4) and is also sound for checking FIS constraints (as we show in Sec. 7).

Two Consequences. While our paper is focused on relaxation, our results have two consequences for the exact implication problem. The first is a *2-tuple model property*: an exact implication, where the antecedents are saturated or conditional CIs, can be verified on uniform probability distributions with 2 tuples. A similar result is known for MVD+FDs [30]. Geiger and Pearl [12], building on an earlier result by Fagin [10], prove that every set of CIs has an *Armstrong model*: a discrete probability distribution that satisfies only the CIs and their consequences, and no other CI. The Armstrong model is also called a *global witness*, and, in general, can be arbitrarily large. Our result concerns a *local witness*: for any EI, if it fails on some probability distribution, then it fails on a 2-tuple uniform distribution.

The second consequence concerns the equivalence between the implication problem of saturated+conditional CIs with that of MVD+FDs. It is easy to check that the former implies the latter (Sec. 2). Wong et al. [38] prove the other direction, relying on the sound and complete axiomatization of MVDs [2]. Our 2-tuple model property implies the other direction immediately.

2 Notation and Preliminaries

We denote by $[n] = \{1, 2, \dots, n\}$. If $\Omega = \{X_1, \dots, X_n\}$ denotes a set of variables and $U, V \subseteq \Omega$, then we abbreviate the union $U \cup V$ with UV.

2.1 Integrity Constraints and Conditional Independence

A *relation instance* R over signature $\Omega = \{X_1, \dots, X_n\}$ is a finite set of tuples with attributes Ω. Let $X, Y, Z \subseteq \Omega$. We say that the instance R satisfies the *functional dependency* (FD) $X \rightarrow Y$, and write $R \models X \rightarrow Y$, if forall $t_1, t_2 \in R$, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$. We say that R satisfies the *embedded multivalued dependency* (EMVD) $X \twoheadrightarrow Y \mid Z$, and write $R \models X \twoheadrightarrow Y \mid Z$, if for all $t_1, t_2 \in R$, $t_1[X] = t_2[X]$ implies $\exists t_3 \in R$ such that $t_1[XY] = t_3[XY]$ and $t_2[XZ] = t_3[XZ]$. One can check that $X \twoheadrightarrow Y \mid Y$ iff $X \rightarrow Y$. When $XYZ = \Omega$, then we call $X \twoheadrightarrow Y \mid Z$ a *multivalued dependency*, MVD; notice that X, Y, Z are not necessarily disjoint [2].

A set of constraints Σ *implies* a constraint τ, in notation $\Sigma \Rightarrow \tau$, if for every instance R, if $R \models \Sigma$ then $R \models \tau$. The implication problem has been extensively studied in the literature; Beeri et al. [2] gave a complete axiomatization of FDs and MVDs, while Herrman [15] showed that the implication problem for EMVDs is undecidable.

Recall that two discrete random variables X, Y are called *independent* if $p(X = x, Y = y) = p(X = x) \cdot p(Y = y)$ for all outcomes x, y. Fix $\Omega = \{X_1, \dots, X_n\}$ a set of n jointly distributed discrete random variables with finite domains $\mathcal{D}_1, \dots, \mathcal{D}_n$, respectively; let p be the probability mass. For $\alpha \subseteq [n]$, denote by X_α the joint random variable $(X_i : i \in \alpha)$ with domain $\mathcal{D}_\alpha \stackrel{\text{def}}{=} \prod_{i \in \alpha} D_i$. We write $p \models X_\beta \perp X_\gamma | X_\alpha$ when X_β, X_γ are conditionally independent given X_α; in the special case $\beta = \gamma$, then $p \models X_\beta \perp X_\beta | X_\alpha$ iff X_α functionally determines[1] X_β, and we write $p \models X_\alpha \rightarrow X_\beta$.

An assertion $Y \perp Z | X$ is called a *Conditional Independence* statement, or a CI; this includes $X \rightarrow Y$ as a special case. When $XYZ = \Omega$ we call it *saturated*, and when $Z = \emptyset$ we call it *marginal*. A set of CIs Σ *implies* a CI τ, in notation $\Sigma \Rightarrow \tau$, if every probability distribution that satisfies Σ also satisfies τ. This implication problem has also been extensively studied: Pearl and Paz [27] gave a sound but incomplete set of *graphoid axioms*, Studeny [36] proved that no finite axiomatization exists, while Geiger and Pearl [12] gave a complete axiomatization for saturated, and marginal CIs.

Lee [21] observed the following connection between database constraints and CIs. The *empirical distribution* of a relation R is the uniform distribution over its tuples, in other words, $\forall t \in R$, $p(t) = 1/|R|$. Then:

▶ **Lemma 1.** ([21]) *Forall $X, Y, Z \subset \Omega$ such that $XYZ = \Omega$.*

$$R \models X \rightarrow Y \quad \Leftrightarrow \quad p \models X \rightarrow Y \qquad \textit{and} \qquad R \models X \twoheadrightarrow Y|Z \quad \Leftrightarrow \quad p \models (Y \perp Z|X) \qquad (1)$$

[1] This means: $\forall u \in D_\alpha$, if $p(X_\alpha = u) \neq 0$ then $\exists v \in D_\beta$ s.t. $p(X_\beta = v | X_\alpha = u) = 1$, and v is unique.

The lemma no longer holds for EMVDs, and for that reason we no longer consider EMVDs in this paper. The lemma immediately implies that if Σ, τ are saturated and/or conditional CIs and the implication $\Sigma \Rightarrow \tau$ holds for all probability distributions, then the corresponding implication holds in databases, where the CIs are interpreted as MVDs or FDs respectively. Wong [38] gave a non-trivial proof for the other direction; we will give a much shorter proof in Corollary 8.

2.2 Background on Information Theory

We adopt required notation from the literature on information theory [40, 6]. For $n > 0$, we identify vectors in $\mathbb{R}^{2^n}$ with functions $2^{[n]} \to \mathbb{R}$.

Polymatroids. A function[2] $h \in \mathbb{R}^{2^n}$ is called a *polymatroid* if $h(\emptyset) = 0$ and satisfies the following inequalities, called *Shannon inequalities*:

1. Monotonicity: $h(A) \leq h(B)$ for $A \subseteq B$
2. Submodularity: $h(A \cup B) + h(A \cap B) \leq h(A) + h(B)$ for all $A, B \subseteq [n]$

The set of polymatroids is denoted $\Gamma_n \subseteq \mathbb{R}^{2^n}$, and forms a polyhedral cone (reviewed in Sec. 5). For any polymatroid h and subsets $A, B, C \subseteq [n]$, we define[3]

$$h(B|A) \stackrel{\text{def}}{=} h(AB) - h(A) \tag{2}$$

$$I_h(B;C|A) \stackrel{\text{def}}{=} h(AB) + h(AC) - h(ABC) - h(A) \tag{3}$$

Then, $\forall h \in \Gamma_n$, $I_h(B;C|A) \geq 0$ and $h(B|A) \geq 0$. The *chain rule* is the identity:

$$I_h(B;CD|A) = I_h(B;C|A) + I_h(B;D|AC) \tag{4}$$

We call $I_h(B;C|A)$ *saturated* if $ABC = [n]$, and *elemental* if $|B| = |C| = 1$; $h(B|A)$ is a special case of I_h, because $h(B|A) = I_h(B;B|A)$.

Entropic Functions. If X is a random variable with a finite domain $\mathcal{D}$ and probability mass p, then $H(X)$ denotes its entropy

$$H(X) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{D}} p(x) \log \frac{1}{p(x)} \tag{5}$$

For a set of jointly distributed random variables $\Omega = \{X_1, \ldots, X_n\}$ we define the function $h : 2^{[n]} \to \mathbb{R}$ as $h(\alpha) \stackrel{\text{def}}{=} H(X_\alpha)$; h is called an *entropic function*, or, with some abuse, an *entropy*. The set of entropic functions is denoted Γ_n^*. The quantities $h(B|A)$ and $I_h(B;C|A)$ are called the *conditional entropy* and *conditional mutual information* respectively. The conditional independence $p \models B \perp C \mid A$ holds iff $I_h(B;C|A) = 0$, and similarly $p \models A \to B$ iff $h(B|A) = 0$, thus, entropy provides us with an alternative characterization of CIs.

2-Tuple Relations and Step functions. 2-tuple relations play a key role for the implication problem of MVDs+FDs: if an implication fails, then there exists a witness consisting of only two tuples [30]. We define a *step function* as the entropy of the empirical distribution of a

[2] Most authors consider rather the space $\mathbb{R}^{2^n-1}$, by dropping $h(\emptyset)$ because it is always 0.
[3] Recall that AB denotes $A \cup B$.

X_1	X_2	U_1	U_2	Pr
0	0	0	0	1/2
1	1	0	0	1/2

(a)

X	Y	Z	Pr
0	0	0	1/4
0	1	1	1/4
1	0	1	1/4
1	1	0	1/4

(b)

A	B	C	D	Pr
0	0	0	0	$1/2 - \varepsilon$
0	1	0	1	$1/2 - \varepsilon$
1	0	1	0	ε
1	1	0	0	ε

(c)

Figure 1 Two relations and their empirical distribution (a),(b); a distribution from [17] (c).

2-tuple relation; $R = \{t_1, t_2\}$, $t_1 \neq t_2$, and $p(t_1) = p(t_2) = 1/2$. We denote the step function by h_U, where $U \subsetneq \Omega$ is the set of attributes where t_1, t_2 agree. One can check:

$$h_U(W) = \begin{cases} 0 & \text{if } W \subseteq U \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

If we set $U = \Omega$ in (6) then $h_\Omega \equiv 0$. Unless otherwise stated, in this paper we do not consider h_Ω to be a step function. Thus, there are $2^n - 1$ step functions and their set is denote $\mathcal{S}_n$. We will use the following fact extensively in this paper: $I_{h_U}(Y; Z|X) = 1$ if $X \subseteq U$ and $Y, Z \not\subseteq U$, and $I_{h_U}(Y; Z|X) = 0$ otherwise.

▶ **Example 2.** Consider the relational instance in Fig. 1 (a). It's entropy is the step function $h_{U_1U_2}(W)$, which is 0 for $W \subseteq U_1U_2$ and 1 otherwise. $R \models X_1 \rightarrow X_2$ because $h(X_2|X_1) = h(X_1X_2) - h(X_1) = 1 - 1 = 0$, and $R \not\models U_1 \rightarrow X_1$ because $h(X_1|U_1) = h(X_1U_1) - h(U_1) = 1 - 0 \neq 0$.

The relational instance $R = \{(x, y, z) \mid x + y + z \mod 2 = 0\}$ in Fig. 1 (b) is called the *parity function*. It's entropy is $h(X) = h(Y) = h(Z) = 1$, $h(XY) = h(XZ) = h(YZ) = h(XYZ) = 2$. We have that $R \models Y \perp Z$ because $I_h(Y; Z) = h(Y) + h(Z) - h(YZ) = 1 + 1 - 2 = 0$, but $R \not\models Y \perp Z|X$ because $I_h(Y; Z|X) = 1$ [4].

2.3 Discussion

This paper studies exact and approximate implications, expressed as equalities or inequalities of entropic functions h. For example, the augmentation axiom for MVDs [2] $A \twoheadrightarrow B|CD \Rightarrow AC \twoheadrightarrow B|D$ is expressed as $I_h(B; CD|A) = 0 \Rightarrow I_h(B; D|AC) = 0$, which holds by the chain rule (4). Thus, our golden standard is to prove that (in)equalities hold forall entropic functions, Γ_n^*. It is known that Γ_n^* is not topologically closed [40]; its topological closure, $\mathbf{cl}(\Gamma_n^*)$, is called the set of *almost entropic functions*. If an *inequality* holds for all entropic functions $h \in \Gamma_n^*$, then, by continuity, it also holds for all almost entropic functions $h \in \mathbf{cl}(\Gamma_n^*)$. However, this observation does not extend to *implications* of (in)equalities; Kaced and Romashchenko [17] gave an example of an exact implication that holds only for entropic functions but fails for almost entropic functions. Thus, when discussing an EI, it matters whether we assume that it holds for Γ_n^* or for $\mathbf{cl}(\Gamma_n^*)$. The only result in this paper where this distinction matters are the two main theorems in Sec. 5: the negative result Theorem 16 holds for both Γ_n^* and for $\mathbf{cl}(\Gamma_n^*)$, while the positive result Theorem 17 holds only for $\mathbf{cl}(\Gamma_n^*)$. The results in Sec. 4 apply to *any* set of polymatroids K that contains all step functions, i.e. $\mathcal{S}_n \subseteq K \subseteq \Gamma_n$, thus they apply to both Γ_n^* and $\mathbf{cl}(\Gamma_n^*)$, while those in Sec 6 and Sec. 7 are stated only for Γ_n and only for (the conic closure of) $\mathcal{S}_n$ respectively.

[4] $h(XY) + h(XZ) - h(X) - h(XYZ) = 2 + 2 - 1 - 2 = 1$

3 Definition of the Relaxation Problem

We now formally define the relaxation problem. We fix a set of variables $\Omega = \{X_1, \ldots, X_n\}$, and consider formulas of the form $\sigma = (Y; Z|X)$, where $X, Y, Z \subseteq \Omega$, which we call a *conditional independence*, CI; when $Y = Z$ then we write it as $X \rightarrow Y$ and call it a *conditional*. An *implication* is a formula $\Sigma \Rightarrow \tau$, where Σ is a set of CIs called *antecedents* and τ is a CI called *consequent*. For a CI $\sigma = (B; C|A)$, we define $h(\sigma) \stackrel{\text{def}}{=} I_h(B; C|A)$, for a set of CIs Σ, we define $h(\Sigma) \stackrel{\text{def}}{=} \sum_{\sigma \in \Sigma} h(\sigma)$. Fix a set K s.t. $\mathcal{S}_n \subseteq K \subseteq \Gamma_n$.

▶ **Definition 3.** *The* exact implication *(EI) $\Sigma \Rightarrow \tau$ holds in K, denoted $K \models_{EI} (\Sigma \Rightarrow \tau)$ if, forall $h \in K$, $h(\Sigma) = 0$ implies $h(\tau) = 0$. The* λ-approximate implication *(λ-AI) holds in K, in notation $K \models \lambda \cdot h(\Sigma) \geq h(\tau)$, if $\forall h \in K$, $\lambda \cdot h(\Sigma) \geq h(\tau)$. The* approximate implication *holds, in notation $K \models_{AI} (\Sigma \Rightarrow \tau)$, if there exist a $\lambda \geq 0$ such that the λ-AI holds.*

We will sometimes consider an equivalent definition for AI, as $\sum_{\sigma \in \Sigma} \lambda_\sigma h(\sigma) \geq h(\tau)$, where $\lambda_\sigma \geq 0$ are coefficients, one for each $\sigma \in \Sigma$; these two definitions are equivalent, by taking $\lambda = \max_\sigma \lambda_\sigma$. Notice that both EI and AI are preserved under subsets of K in the sense that $K_1 \subseteq K_2$ and $K_2 \models_x (\Sigma \Rightarrow \tau)$ implies $K_1 \models_x (\Sigma \Rightarrow \tau)$, for $x \in \{EI, AI\}$.

AI always implies EI. Indeed, $h(\tau) \leq \lambda \cdot h(\Sigma)$ and $h(\Sigma) = 0$, implies $h(\tau) \leq 0$, which further implies $h(\tau) = 0$, because $h(\tau) \geq 0$ for every CI τ, and every polymatroid h. In this paper we study the reverse.

▶ **Definition 4.** *Let $\mathcal{I}$ be a syntactically-defined class of implication statements $(\Sigma \Rightarrow \tau)$, and let $K \subseteq \Gamma_n$. We say that $\mathcal{I}$* admits a relaxation *in K if, every implication statement $(\Sigma \Rightarrow \tau)$ in $\mathcal{I}$ that holds exactly, also holds approximately: $K \models_{EI} \Sigma \Rightarrow \tau$ implies $K \models_{AI} \Sigma \Rightarrow \tau$. We say that $\mathcal{I}$ admits a* λ-relaxation *if every EI admits a λ-AI.*

▶ **Example 5.** Let $\Sigma = \{(A; B|\emptyset), (A; C|B)\}$ and $\tau = (A; C|\emptyset)$. Since $I_h(A; C|\emptyset) \leq I_h(A; B|\emptyset) + I_h(A; C|B)$ by the chain rule (4), then the exact implication $\Gamma_n \models_{EI} \Sigma \Rightarrow \tau$ admits a 1-AI.

4 Relaxation for FDs and MVDs: Always Possible

In this section we consider the implication problem where the antecedents are either saturated CIs, or conditionals. This is a case of special interest in databases, because the constraints correspond to MVDs, or FDs. Recall that a CI $(B; C|A)$ is *saturated* if $ABC = \Omega$ (i.e., the set of all attributes). Our main result in this section is:

▶ **Theorem 6.** *Assume that each formula in Σ is either saturated, or a conditional, and let τ be an arbitrary CI. Assume $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$. Then:*

1. *$\Gamma_n \models \frac{n^2}{4} h(\Sigma) \geq h(\tau)$.*
2. *If τ is a conditional, $Z \rightarrow X$, then $\Gamma_n \models h(\Sigma) \geq h(\tau)$.*

Before we prove the theorem, we list two important consequences.

▶ **Corollary 7.** *Let Σ consist of saturated CIs and/or conditionals, and let τ be any CI. Then $\mathcal{S}_n \models \Sigma \Rightarrow_{EI} \tau$ implies $\Gamma_n \models \Sigma \Rightarrow_{EI} \tau$*

Proof. If $\mathcal{S}_n \models \Sigma \Rightarrow_{EI} \tau$ then $\forall h \in \Gamma_n$, $h(\tau) \leq \frac{n^2}{4} h(\Sigma)$, thus $h(\Sigma) = 0$ implies $h(\tau) = 0$. ◀

The corollary has an immediate application to the inference problem in graphical models [12]. There, the problem is to check if every probability distribution that satisfies all CIs in Σ also satisfies the CI τ; we have seen that this is equivalent to $\Gamma_n^* \models_{EI} \Sigma \Rightarrow \tau$. The corollary states that it is enough that this implication holds on all of the uniform 2-tuple distributions, i.e. $\mathcal{S}_n \models \Sigma \Rightarrow_{EI} \tau$, because this implies the (even stronger!) statement $\Gamma_n \models \Sigma \Rightarrow_{EI} \tau$. Decidability was already known: Geiger and Pearl [12] proved that the set of graphoid axioms is sound and complete for the case when both Σ and τ are saturated, while Gyssens et al. [14] improve this by dropping any restrictions on τ.

The second consequence is the following:

▶ **Corollary 8.** *Let Σ, τ consist of saturated CIs and/or conditionals. Then the following two statements are equivalent:*

1. *The implication $\Sigma \Rightarrow \tau$ holds, where we interpret Σ, τ as MVDs and/or FDs.*
2. $\Gamma_n \models_{EI} \Sigma \Rightarrow \tau$.

Proof. We have shown right after Lemma 1 that (2) implies (1). For the opposite direction, by Th. 6, we need only check $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$, which holds because on every uniform probability distribution a saturated CI holds iff the corresponding MVD holds, and similarly for conditionals and FDs. Since the 2-tuple relation satisfies the implication for MVDs+FDs, it also satisfies the implication for CIs, proving the claim. ◀

Wong et al. [38] have proven that the implication for MVDs is equivalent to that of the corresponding saturated CIs (called there BMVD); they did not consider FDs. For the proof in the hard direction, they use the sound and complete axiomatization of MVDs in [2]. In contrast, our proof is independent of any axiomatic system, and is also much shorter. Finally, we notice that the corollary also implies that, in order to check an implication between MVDs and/or FDs, it suffices to check it on all 2-tuple databases: indeed, this is equivalent to checking $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$, because this implies Item (2), which in turn implies item (1). This rather surprising fact was first proven in [30].

We now turn to the proof of Theorem 6. Before proceeding, we note that we can assume w.l.o.g. that Σ consists only of saturated CIs. Indeed, if Σ contains a non-saturated term, then by assumption it is a conditional, $X \to Y$, and we will replace it with two saturated terms: $(Y; Z|X)$ and $XZ \to Y$, where $Z = \Omega \setminus XY$. Denoting Σ' the new set of formulas, we have $h(\Sigma) = h(\Sigma')$, because $h(Y|X) = I_h(Y; Z|X) + h(Y|XZ)$. Thus, we will assume w.l.o.g. that all formulas in Σ are saturated.

Theorem 6 follows from the next result, which is also of independent interest. We say that a CI $(X; Y|Z)$ is *elemental* if $|X| = |Y| = 1$. We say that σ *covers* τ if all variables in τ are contained in σ; for example $\sigma = (abc; d|e)$ covers $\tau = (cd; be)$. Then:

▶ **Theorem 9.** *Let τ be an elemental CI, and suppose each formula in Σ covers τ. Then $\mathcal{S}_n \models_{EI} (\Sigma \Rightarrow \tau)$ implies $\Gamma_n \models h(\tau) \leq h(\Sigma)$.*

Notice that this result immediately implies Item (1) of Theorem 6, because every $\tau = (Y; Z|X)$ can be written as a sum of $|Y| \cdot |Z| \leq n^2/4$ elemental terms (by the chain rule). In what follows we prove Theorem 9, then use it to prove item (2) of Theorem 6.

Finally, we consider whether (1) of Theorem 6 can be strengthened to a 1-relaxation; we give in Th. 11 below a sufficient condition, whose proof uses the notion of I-measure [40] and is included in the full paper [18], and leave open the question whether 1-relaxation holds in general for implications where the antecedents are saturated CIs and conditionals.

▶ **Definition 10.** *We say that two CIs $(X; Y|Z)$ and $(A; B|C)$ are* disjoint *if at least one of the following four conditions holds: (1) $X \subseteq C$, (2) $Y \subseteq C$, (3) $A \subseteq Z$, or (4) $B \subseteq Z$.*

If $\tau = (X; Y|Z)$ and $\sigma = (A; B|C)$ are disjoint, then for any step function h_W, it cannot be the case that both $h_W(\tau) \neq 0$ and $h_W(\sigma) \neq 0$. Indeed, if such W exists, then $Z, C \subseteq W$ and, assuming (1) $X \subseteq C$ (the other three cases are similar), we have $ZX \subseteq W$ thus $h_W(\tau) = 0$.

▶ **Theorem 11.** *Let Σ be a set of saturated, pairwise disjoint CI terms (Def. 10), and τ be a saturated mutual information. Then, $\mathcal{S}_n \models_{EI} (\Sigma \Rightarrow \tau)$ implies $\Gamma_n \models h(\tau) \leq h(\Sigma)$.*

4.1 Proof of Theorem 9

The following holds by the chain rule (proof in the appendix), and will be used later on.

▶ **Lemma 12.** *Let $\sigma = (A; B|C)$ and $\tau = (X; Y|Z)$ be CIs such that $X \subseteq A$, $Y \subseteq B$, $C \subseteq Z$ and $Z \subseteq ABC$. Then, $\Gamma_n \models h(\tau) \leq h(\sigma)$.*

We now prove theorem 9. We use lower case for single variables, thus $\tau = (x; y|Z)$ because it is elemental. We may assume w.l.o.g. $x, y \notin Z$ (otherwise $I_h(x; y|Z) = 0$ and the lemma holds trivially). The *deficit* of an elemental CI $\tau = (x; y|Z)$ is the quantity $|\Omega - Z|$. We prove by induction on the deficit of τ that $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$ implies $\Gamma_n \models h(\tau) \leq h(\Sigma)$.

Assume $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$, and consider the step function at Z. Since $h_Z(\tau) = 1$, there exists $\sigma \in \Sigma$, $\sigma = (A; B|C)$, such that $h_Z(\sigma) = 1$; this means that $C \subseteq Z$, and $A, B \not\subseteq Z$. In particular $x, y \notin C$, therefore $x, y \in AB$, because σ covers τ. If $x \in A$ and $y \in B$ (or vice versa), then $\Gamma_n \models h(\tau) \leq h(\sigma)$ by Lemma 12, proving the theorem. Therefore we assume w.l.o.g. that $x, y \in A$ and none is in B. Furthermore, since $B \not\subseteq Z$, there exists $u \in B - Z$.

Base case: τ is saturated. Then $u \notin xyZ$, contradicting the assumption that τ is saturated; in other words, in the base case, it is the case that $x \in A$ and $y \in B$.

Step: Let $Z_A = Z \cap A$, and $Z_B = Z \cap B$. Since $C \subseteq Z$, and $\sigma = (A; B|C)$ covers τ, then $Z = Z_A Z_B C$. We also write $A = xyA'Z_A$ (since $x, y \in A$) and $B = uB'Z_B$. So, we have that $\sigma = (A; B|C) = (xyA'Z_A; uB'Z_B|C)$, and we use the chain rule to define σ_1, σ_2:

$$h(\sigma) = I_h(xyA'Z_A; uB'Z_B|C) = I_h(\underbrace{xyA'Z_A; uZ_B|C}_{\overset{\text{def}}{=}\sigma_1}) + I_h(\underbrace{xyA'Z_A; B'|uCZ_B}_{\overset{\text{def}}{=}\sigma_2})$$

We also partition Σ s.t. $h(\Sigma) = h(\sigma_1) + h(\Sigma_2)$, where $\Sigma_2 \overset{\text{def}}{=} (\Sigma \setminus \{\sigma\}) \cup \{\sigma_2\}$.
Next, define $\tau' \overset{\text{def}}{=} (x; uy|Z)$ and use the chain rule to define τ_1, τ_2:

$$h(\underbrace{x; y|Z}_{\tau}) \leq I_h(\underbrace{x; uy|Z}_{\tau'}) = I_h(\underbrace{x; u|Z}_{\overset{\text{def}}{=}\tau_1}) + I_h(\underbrace{x; y|uZ}_{\overset{\text{def}}{=}\tau_2}) \tag{7}$$

By Lemma 12, $\Gamma_n \models h(\sigma_1) \geq h(\tau_1)$. We will prove: $\mathcal{S}_n \models_{EI} \Sigma_2 \Rightarrow \tau_2$. This implies the theorem, because Σ_2 is saturated, and by the induction hypothesis $\Gamma_n \models h(\Sigma_2) \geq h(\tau_2)$ (since the deficit of τ_2 is one less than that of τ), and the theorem follows from $h(\Sigma) = h(\sigma_1) + h(\Sigma_2) \geq h(\tau_1) + h(\tau_2) = h(\tau') \geq h(\tau)$. It remains to prove $\mathcal{S}_n \models_{EI} \Sigma_2 \Rightarrow \tau_2$, and we start with a weaker claim:

▷ Claim 13. $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau_2$.

Proof. By Lemma 12 we have that $h(\sigma) = I_h(xyA'Z_A; uB'Z_B|C) \geq I_h(xy; u|Z) = I_h(y; u|Z) + I_h(x; u|yZ)$. Therefore, $\Sigma \Rightarrow (x; u|yZ)$. Since $\Sigma \Rightarrow (x; y|Z)$, then by the chain rule we have that $\Sigma \Rightarrow (x; uy|Z) = \tau'$, and the claim follows from (7). ◁

Finally, we prove $\mathcal{S}_n \models_{EI} \Sigma_2 \Rightarrow \tau_2$. Assume otherwise, and let h_W be a step function such that $h_W(\tau_2) = I_{h_W}(x; y|uZ) = 1$, and $h_W(\Sigma_2) = 0$. This means that $uZ \subseteq W$. Therefore $uZ_B \subseteq W$, implying $I_{h_W}(xyA'Z_A; uZ_B|C) = h_W(\sigma_1) = 0$ (because $uZ_BC \subseteq uZ$). Therefore, $h_W(\Sigma) = h_W(\sigma_1) + h_W(\Sigma_2) = 0$, contradicting the fact that $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau_2$.

4.2 Proof of Theorem 6 Item 2

▶ **Lemma 14.** *Suppose $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$, where $\tau = (X;Y|Z)$. Let $\sigma \in \Sigma$ such that τ, σ are disjoint (Def. 10). Then: $\mathcal{S}_n \models_{EI} \big(\Sigma\backslash\{\sigma\}\big) \Rightarrow \tau$.*

Proof. Let $\Sigma' \stackrel{\text{def}}{=} \Sigma \setminus \{\sigma\}$. Assume by contradiction that there exists a step function h_W such that $h_W(\Sigma') = 0$ and $h_W(\tau) = 1$. Since σ, τ are disjoint, $h_W(\sigma) = 0$. Then $h_W(\Sigma) = 0$, contradicting the assumption $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$. ◀

▶ **Lemma 15.** *Let Σ be a set of saturated CIs s.t. $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$. Suppose $\tau = (Z \rightarrow uX)$ (which, recall, is a shorthand for $(uX; uX|Z)$), and define $\tau_1 = (Z \rightarrow u)$, $\tau_2 = (uZ \rightarrow X)$; thus, $h(\tau) = h(\tau_1) + h(\tau_2)$. Then there exists Σ_1 and Σ_2 such that: (1) $h(\Sigma) = h(\Sigma_1) + h(\Sigma_2)$; we say that Σ_1, Σ_2 form a parition of Σ. (2) Σ_1 covers τ_1 and $\mathcal{S}_n \models_{EI} \Sigma_1 \Rightarrow \tau_1$. (3) Σ_2 is saturated and $\mathcal{S}_n \models \Sigma_2 \Rightarrow \tau_2$.*

Proof. We partition Σ into Σ_1 and Σ_2 as follows. For every $\sigma = (A;B|C) \in \Sigma$, if $u \in C$ then we place σ in Σ_2. Otherwise, assume w.l.o.g that $u \in A$, and we write $A = uA_ZA_XA'$ where $A_Z = A \cap Z$, $A_X = A \cap X$, and $A' = A\backslash\{uA_ZA_X\}$. We use the chain rule to define σ_1, σ_2:

$$I_h(A;B|C) = I_h(uA_ZA_XA';B|C) = I_h(\underbrace{uA_Z;B|C}_{\stackrel{\text{def}}{=}\sigma_1}) + I(\underbrace{A_XA';B|uA_ZC}_{\stackrel{\text{def}}{=}\sigma_2}) \tag{8}$$

We place σ_1 in Σ_1, and σ_2 in Σ_2. We observe that σ_1 covers τ_1 (because $Z = A_ZB_ZC_Z \subseteq A_ZBC$) and σ_2 is saturated. Furthermore, $h(\Sigma_1) + h(\Sigma_2) = h(\Sigma)$. We prove $\Sigma_1 \models_{EI} \tau_1$. By assumption, $\Sigma \models_{EI} \tau_1 = (Z \rightarrow u)$. Let any $\sigma_2 = (A;B|C) \in \Sigma_2$; since $u \in C$, by Lemma 14 we can remove it, obtaining $\Sigma \setminus \{\sigma_2\} \models_{EI} \tau_1$; repeating this process proves $\Sigma_1 \models_{EI} \tau_1$. Finally, we prove $\Sigma_2 \models_{EI} \tau_2$. By assumption, $\Sigma \models_{EI} \tau_2 = (uZ \rightarrow X)$. Let any $\sigma_1 = (uA_Z;B|C) \in \Sigma_1$; since $uA_Z \subseteq uZ$, by Lemma 14 we can remove it, obtaining $\Sigma \setminus \{\sigma_1\} \models_{EI} \tau_2$; repeating this process proves $\Sigma_2 \models_{EI} \tau_2$. ◀

We now complete the proof of Theorem 6 item 2. Let $\tau = (Z \rightarrow X)$, and Σ be saturated. We show, by induction on $|X|$, that if $\mathcal{S}_n \models_{EI} \Sigma \Rightarrow \tau$ then $\Gamma_n \models h(\tau) \leq h(\Sigma)$. If $|X| = 1$, then $X = \{x\}$, $h(x|Z) = I(x;x|Z)$ is elemental, and the claim follows from Th. 9. Otherwise, let u be any variable in X, write $\tau = (Z \rightarrow uX')$, and apply Lemma 15 to $\tau_1 = (Z \rightarrow u)$, $\tau_2 = (Zu \rightarrow X')$, which gives us a partition of Σ into Σ_1, Σ_2. On one hand, $\mathcal{S}_n \models_{EI} \Sigma_1 \Rightarrow \tau_1$, and from Th. 9 we derive $h(\tau_1) \leq h(\Sigma_1)$ (because τ_1 is elemental, and covered by Σ_1); on the other hand $\mathcal{S}_n \models_{EI} \Sigma_2 \Rightarrow \tau_2$ where Σ_2 is saturated, which implies, by induction, $h(\tau_2) \leq h(\Sigma_2)$. The result follows from $h(\tau) = h(\tau_1) + h(\tau_2) \leq h(\Sigma_1) + h(\Sigma_2) = h(\Sigma)$, completing the proof.

5 Relaxation for General CIs: Sometimes Impossible

We consider the relaxation problem for arbitrary Conditional Independence statements. Recall that our golden standard is to check (in)equalities forall entropic functions, $h \in \Gamma^*_n$. As we saw, for MVD+FDs, these (in)equalities coincide with those satisfied by $\mathcal{S}_n$, and with those satisfied by Γ_n. In general, however, they differ. We start with an impossibility result, then prove that relaxation with an arbitrarily small error term always exists. Both results are for the topological closure, $\mathbf{cl}(\Gamma^*_n)$. This makes the negative result stronger, but the positive result weaker; it is unlikely for the positive result to hold for Γ^*_n, see [17, Sec.V.(A)] and Appendix A.

▶ **Theorem 16.** *There exists* Σ, τ *with four variables, such that* $\mathbf{cl}\,(\Gamma_4^*) \models_{EI} (\Sigma \Rightarrow \tau)$ *and* $\mathbf{cl}\,(\Gamma_4^*) \not\models_{AI} (\Sigma \Rightarrow \tau)$.

For the proof, we adapt an example by Kaced and Romashchenko [17, Inequality ($\mathcal{I}5'$) and Claim 5], built upon an earlier example by Matúš [25]. Let Σ and τ be the following:

$$\Sigma = \{(C;D|A), (C;D|B), (A;B), (B;C|D)\} \qquad \tau = (C;D) \tag{9}$$

We first prove that, for any $\lambda \geq 0$, there exists an entropic function h such that:

$$I_h(C;D) > \lambda \cdot (I_h(C;D|A) + I_h(C;D|B) + I_h(A;B) + I_h(B;C|D)) \tag{10}$$

Indeed, consider the distribution shown in Fig. 1 (c) (from [17]). By direct calculation, $I_h(C;D) = \varepsilon + O(\varepsilon^2) = \Omega(\varepsilon)$, while $I_h(C;D|A) = I_h(C;D|B) = I_h(A;B) = 0$ and $I_h(B;C|D) = O(\varepsilon^2)$ and we obtain Eq.(10) by choosing ε small enough. Next, we prove $\mathbf{cl}\,(\Gamma_n^*) \models_{EI} (\Sigma \Rightarrow \tau)$. Matúš [25] proved the following[5] $\forall h \in \Gamma_n^*$ and $\forall k \in \mathbb{N}$:

$$I_h(C;D) \leq I_h(C;D|A) + \frac{k+3}{2} I_h(C;D|B) + I_h(A;B) + \frac{k-1}{2} I_h(B;C|D) + \frac{1}{k} I_h(B;D|C) \tag{11}$$

The inequality obviously holds for $\mathbf{cl}\,(\Gamma_n^*)$ too. The EI follows by taking $k \to \infty$. Inequality (11) is almost a relaxation of the implication (9): the only extra term is the last term, which can be made arbitrarily small by increasing k. Our second result generalizes this:

▶ **Theorem 17.** *Let* Σ, τ *be arbitrary CIs, and suppose* $\mathbf{cl}\,(\Gamma_n^*) \models \Sigma \Rightarrow \tau$. *Then, for every* $\varepsilon > 0$ *there exists* $\lambda > 0$ *such that, forall* $h \in \mathbf{cl}\,(\Gamma_n^*)$:

$$h(\tau) \leq \lambda \cdot h(\Sigma) + \varepsilon \cdot h(\Omega) \tag{12}$$

Intuitively, the theorem shows that every EI can be relaxed in $\mathbf{cl}\,(\Gamma_n^*)$, if one allows for an error term, which can be made arbitrarily small. We notice that the converse of the theorem always holds: if $h(\Sigma) = 0$, then (12) implies $h(\tau) \leq \varepsilon \cdot h(\Omega)$, $\forall \varepsilon > 0$, which implies $h(\tau) = 0$.

Proof of Theorem 17. For the proof we need a brief review of cones [37, 4]. A set $C \subseteq \mathbb{R}^N$ is *convex* if, for any two points $x_1, x_2 \in C$ and any $\theta \in [0,1]$, $\theta x_1 + (1-\theta)x_2 \in C$; and it is called a *cone*, if for every $x \in C$ and $\theta \geq 0$ we have that $\theta x \in C$. The *conic hull* of C, $\mathbf{conhull}\,(C)$, is the set of vectors of the form $\theta_1 x_1 + \cdots + \theta_k x_k$, where $x_1, \ldots, x_k \in C$ and $\theta_i \geq 0, \forall i \in [k]$. A cone K is *finitely generated* if $K = \mathbf{conhull}\,(L)$ for some finite set $L \subset \mathbb{R}^N$, and is *polyhedral* if there exists $u_1, \ldots, u_r \in \mathbb{R}^N$ s.t. $K = \{x \mid u_i \cdot x \geq 0, i \in [r]\}$; a cone is finitely generated iff it is polyhedral. For any $K \subseteq \mathbb{R}^N$, the *dual* is the set $K^* \subseteq \mathbb{R}^N$ defined as:

$$K^* \stackrel{\text{def}}{=} \{y \mid \forall x \in K, x \cdot y \geq 0\} \tag{13}$$

K^* represents the linear inequalities that hold for all $x \in K$, and is always a closed, convex cone (it is the intersection of closed half-spaces). We warn that the $*$ in Γ_n^* does *not* represent the dual; the notation Γ_n^* for entropic functions is by now well established, and we adopt it here too, despite it's clash with the standard notation for the dual cone. The following are known properties of cones (reviewed and proved in the Appendix):

(A) For any set K, $\mathbf{cl}\,\big(\mathbf{conhull}\,(K)\big) = K^{**}$.

[5] Matus [25] proved $I(C;D) \leq I(C;D|A) + I(C;D|B) + I(A;B) + I(C;E|B) + \frac{1}{k} I(B;E|C) + \frac{k-1}{2}(I(B;C|D) + I(C;D|B))$. Inequality (11) follows by setting $E = D$.

(B) If L is a finite set, then $\textbf{conhull}(L)$ is closed.

(C) If K_1 and K_2 are closed, convex cones then: $(K_1 \cap K_2)^* = \Big(\textbf{cl}\left(\textbf{conhull}(K_1^* \cup K_2^*)\right)\Big)$. ◀

Theorem 17 follows from a more general statement about cones:

▶ **Theorem 18.** *Let $K \subseteq \mathbb{R}^N$ be a closed, convex cone, and let $y_1, \ldots, y_m, y$ be $m+1$ vectors in $\mathbb{R}^N$. The following are equivalent:*

(a) *For every $x \in K$, if $x \cdot y_1 \leq 0, \ldots, x \cdot y_m \leq 0$ then $x \cdot y \leq 0$.*

(b) *For every $\varepsilon > 0$ there exists $\theta_1, \ldots, \theta_m \geq 0$ and an error vector $e \in \mathbb{R}^N$ such that $||e||_\infty < \varepsilon$ and, for every $x \in K$, $x \cdot y \leq \theta_1 x \cdot y_1 + \cdots + \theta_m x \cdot y_m + x \cdot e$.*

Proof. Let $L \stackrel{\text{def}}{=} \{-y_1, -y_2, \ldots, -y_m\}$. Statement (a) is equivalent to $-y \in (K \cap L^*)^*$. Consider statement (b). It asserts $\forall \varepsilon > 0, \exists ||e||_\infty < \varepsilon$ such that

$$\exists \theta_1 \geq 0, \ldots, \exists \theta_k \geq 0, \underbrace{\forall x \in K, x \cdot y \leq x \cdot (\sum \theta_i y_i + e)}_{-y + \sum \theta_i y_i + e \in K^*}$$

In other words, $-y + e \in \textbf{conhull}(K^* \cup L)$ and, since this must hold for arbitrarily small $||e||_\infty$, statement (b) is equivalent to $-y \in \textbf{cl}\left(\textbf{conhull}(K^* \cup L)\right)$. We prove equivalence of (a) and (b):

$$\begin{aligned}
(K \cap L^*)^* &= \textbf{cl}\left(\textbf{conhull}(K^* \cup L^{**})\right) && \text{Item (C)}\\
&= \textbf{cl}\Big(\textbf{conhull}\Big(K^* \cup \textbf{cl}\left(\textbf{conhull}(L)\right)\Big)\Big) && \text{Item (A)}\\
&= \textbf{cl}\Big(\textbf{conhull}\left(K^* \cup \textbf{conhull}(L)\right)\Big) && \text{Item (B)}\\
&= \textbf{cl}\left(\textbf{conhull}(K^* \cup L)\right) && \text{Def. of } \textbf{conhull}(-)
\end{aligned}$$ ◀

We now prove Theorem 17, using the fact that $K \stackrel{\text{def}}{=} \textbf{cl}(\Gamma_n^*)$ is a closed cone [40]. Let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$. Associate to each term $\sigma_i = (B_i; C_i | A_i)$ the vector $y_i \in \mathbb{R}^{2^n}$ such that, forall $h \in \mathbb{R}^{2^n}$, $h \cdot y_i = I_h(B_i; C_i | A_i) = h(A_i B_i) + h(A_i C_i) - h(A_i B_i C_i) - h(C_i)$ (i.e. y_i has two coordinates equal to $+1$, and two equal to -1), for $i = 1, m$. Denote by y the similar vector associated to τ. To prove Theorem 17, let $\varepsilon > 0$. By assumption, $\textbf{cl}(\Gamma_n^*) \models \Sigma \Rightarrow \tau$, thus condition (a) of Th. 18 holds, and this implies condition (b), where we choose e such that $||e||_\infty < \varepsilon/2^n$. Then, condition (b) becomes:

$$h(\tau) = h \cdot y \leq \sum_i \theta_i h \cdot y_i + h \cdot e = \sum_i \theta_i h(\sigma_i) + \sum_{W \subseteq [n]} |e_W| h(W) \leq \lambda h(\Sigma) + \varepsilon h(\Omega)$$

where $\lambda = \max_i \theta_i$. This completes the proof of Theorem 17.

6 Restricted Axioms

The characterization of the entropic cone $\textbf{cl}(\Gamma_n^*)$ is currently an open problem [40]. In other words, there is no known decision procedure capable of deciding whether an exact or approximate implication holds for all entropic functions. In this section, we consider implications that can be inferred using only the Shannon inequalities (e.g., (2), and (3)), and thus hold for all polymatroids $h \in \Gamma_n$. Several tools exists (e.g. ITIP or XITIP [39]) for checking such inequalities.

This study is important for several reasons. First, by restricting to Shannon inequalities we obtain a sound, but in general incomplete method for deciding implications. All axioms for reasoning about MVD, FD, or semi-graphoid axioms[6] [2, 27, 12] are, in fact, based on Shannon inequalities. Second, under some syntactic restrictions, they are also complete; as we saw, they are complete for MVD and/or FDs, for saturated constraints and/or conditionals, and also for marginal constraints [12]. Third, Shannon inequalities are complete for reasoning for a different class of constraints, called *measure-based constraints*, which were introduced by Sayrafi et al. [32] (where Γ_n is denoted by $\mathcal{M}_{\text{SI}}$) and shown to have a variety of applications.

We start by showing that every exact implication of CIs can be relaxed over Γ_n. This result was known, e.g. [17]; we re-state and prove it here for completeness.

▶ **Theorem 19.** *Let Σ, τ be arbitrary CIs. If $\Gamma_n \models_{EI} \Sigma \Rightarrow \tau$, then there exist $\lambda \geq 0$, s.t. $\Gamma_n \models h(\tau) \leq \lambda \cdot h(\Sigma)$. In other words, CIs admit relaxation over Γ_n.*

Proof. (Sketch) We set $K = \Gamma_n$ in Th. 18. Then K is polyhedral, hence K^* is finitely generated. Therefore, in the proof of Th. 18, the set $K^* \cup L$ is finitely generated, hence **conhull** $(K^* \cup L)$ is closed, therefore there is no need for an error vector e in Statement (b) of Th. 18, and, hence, no need for ε in AI (12) ◀

It follows that Shannon inequalities are incomplete for proving the implication $\Sigma \Rightarrow \tau$, where Σ, τ are given by Eq. (9). This is a "non-Shannon" exact implication, i.e. it holds only in $\mathbf{cl}\,(\Gamma_n^*)$, but fails in Γ_n, otherwise it would admit a relaxation. The explanation is that Matus' inequality (11) is a non-Shannon inequality. (The first example of a non-Shannon inequality is due to Yeung and Zhang [42].) Next, we turn our attention to the size of the factor λ. We prove a lower bound of 3:

▶ **Theorem 20** ([9])**.** *The following inequality holds for all polymatroids $h \in \Gamma_n$:*

$$h(Z) \leq I_h(A;B|C) + I_h(A;B|D) + I_h(C;D|E) + I_h(A;E) + 3h(Z|A) + 2h(Z|B) \qquad (14)$$

but the inequality fails if any of the coefficients 3, 2 are replaced by smaller values. In particular, denoting τ, Σ the terms on the two sides of Eq.(14), the exact implication $\Gamma_n \models_{EI} \Sigma \Rightarrow \tau$ holds, and does not have a 1-relaxation.

We have checked the two claims in the theorem using the ITIP[7] tool. For the positive result, we also provide direct (manual) proof in the full version of this paper [18]. Since some EIs relax only with $\lambda \geq 3$, the next question is, how large does λ need to be? We prove this upper bound in [18]:

▶ **Theorem 21.** *If $\Gamma_n \models \Sigma \Rightarrow \tau$ then $\Gamma_n \models \tau \leq (2^n)! \cdot h(\Sigma)$. In other words, every implication of CIs admits a $(2^n)!$-relaxation over Γ_n.*

7 Restricted Models

In this section we restrict ourselves to models of uniform 2-tuple distributions. Recall that their entropic functions are the step functions, $\mathcal{S}_n$. Denoting their conic hull by $\mathcal{P}_n \stackrel{\text{def}}{=} \textbf{conhull}\,(\mathcal{S}_n)$, we prove here that all EI's admit a 1-relaxation on $\mathcal{P}_n$. This study has two motivations. First, it leads to a complete, but unsound procedure for implication. A

[6] Semi-graphoid axioms restricted to "strictly positive" distributions, which fail Γ_n^*.

[7] `http://user-www.ie.cuhk.edu.hk/~ITIP/`

model checking system may verify an EI or AI by checking it on all 2-tuple distributions. As we saw in Sec. 4 this procedure is sound and complete for saturated or conditional CI's, but it may be unsound in general, for example the inequality $I_h(X;Y|Z) \leq I_h(X;Y)$ holds for all step functions, but fails on the "parity function" in Fig. 1 (b). Second, this model checking procedure is sound and complete in an important application, namely for checking *differential constraints* in market basket analysis [33]. Differential constraints are more general than the CIs we discussed so far, yet we prove here that they, too, admit a 1-relaxation in $\mathcal{P}_n$. Thus, our relaxation result has immediate application to market basket constraints.

Consider a set of items $\Omega = \{X_1, \ldots, X_n\}$, and a set of baskets $\mathcal{B} = \{b_1, \ldots, b_N\}$ where every basket is a subset $b_i \subseteq \Omega$. The *support function* $f : 2^\Omega \to \mathbb{N}$ assigns to every subset $W \subseteq \mathcal{B}$ the number of baskets in $\mathcal{B}$ that contain the set W: $f(W) = |\{i \mid i \in [N], W \subseteq b_i \in \mathcal{B}\}|$. A constraint $f(W) = f(WX)$ asserts that every basket that contains W also contains X. Sayrafi and Van Gucht [33] define the *density* of a function $f : 2^\Omega \to \mathbb{N}$ as $d_f(W) \overset{\text{def}}{=} \sum_{Z:W\subseteq Z}(-1)^{|Z-W|}f(Z)$; we show below this equals the number of baskets $b_i \in \mathcal{B}$ s.t. $W = b_i$. Then, they define a *differential constraint* to be a statement of the form $d_f(W) = 0$, for some $W \subseteq \Omega$, and study the implication problem of differential constraints.

We now explain the connection to step functions $\mathcal{S}_n$; for the purpose of this discussion we consider h_Ω to be a step function, which is $h_\Omega \equiv 0$ (Sec 2.2). Fix $i \in [N]$ and consider the single basket $b_i \in \mathcal{B}$. Define f_{b_i} to be the support function for the singleton set $\{b_i\}$, that is $f_{b_i}(W) = 1$ if $W \subseteq b_i$ and 0 otherwise. It follows that $h_{b_i}(W) \overset{\text{def}}{=} 1 - f_{b_i}(W)$ is precisely the step function at b_i. The support function for $\mathcal{B} = \{b_1, \ldots, b_N\}$ is $f = \sum_{i\in[N]} f_{b_i} = N - h$, where $h \overset{\text{def}}{=} \sum_{i\in[N]} h_{b_i} \in \mathcal{P}_n$. Thus, any support function f gives rise to a polymatroid $h \overset{\text{def}}{=} N - f \in \mathcal{P}_n$. By linearity, their densities are related by $d_f = d_N - d_h$, where d_N is the density of the constant function N: $d_N(W) = N \cdot \sum_{Z:W\subseteq Z\subseteq\Omega}(-1)^{|Z-W|}$, thus $d_N(\Omega) = N$ and $d_N(W) = 0$ for $W \subsetneq \Omega$; in particular, $d_f(W) = -d_h(W)$ for $W \subsetneq \Omega$. Conversely, any $h = \sum_{U\subsetneq\Omega} c_U h_U \in \mathcal{P}_n$, where $c_U \geq 0$, and any $N \geq \sum_U c_U$ gives rise to a set of baskets $\mathcal{B}$ of size N, where each set $U \subsetneq \Omega$ occurs exactly c_U times and Ω occurs exactly $N - \sum_U c_U$ times, such that the support function of $\mathcal{B}$ is $f = |\mathcal{B}| - h$. Therefore, the implication problem of differential constraints studied in [33] is equivalent to the implication problem for $\mathcal{P}_n$. We prove that the latter admits a 1-relaxation. We start with a lemma (proof in Appendix):

► **Lemma 22.** *Fix a function $h : 2^\Omega \to \mathbb{R}$ s.t. $h(\emptyset) = 0$. Then $h = \sum_{Z\subsetneq\Omega}(-d_h(Z)) \cdot h_Z$. In other words, the step functions h_Z form a basis for the vector space $\{h \in R^{2^n} \mid h(\emptyset) = 0\}$.*

Fix a step function, $h = h_W$. By the Lemma, h_W admits a unique decomposition $h_W = \sum_{Z\subsetneq\Omega}(-d_{h_W}(Z))h_Z$; it follows that $d_{h_W}(Z) = -1$ when $Z = W$ and $d_{h_W}(Z) = 0$ otherwise. In particular, $d_h \leq 0$ forall $h \in \mathcal{P}_n$. Fix a set of baskets $\mathcal{B} = \{b_1, \ldots, b_N\}$, and let f be its support function. We prove that $d_f(Z)$ is equal to the number of baskets b_i s.t. $Z = b_i$; in particular $d_f \geq 0$. Indeed, for $Z = \Omega$ this follows from the definition of the differential d_f, while for $Z \subsetneq \Omega$ we use the fact that $f = N - \sum_i h_{b_i}$ and $d_f(Z) = -\sum_i d_{h_{b_i}}(Z)$.

The quantity $I_h(y_1; y_2; \cdots; y_m|W) \overset{\text{def}}{=} -\sum_{Z:W\subseteq Z\subseteq\{y_1,\ldots,y_m\}}(-1)^{|Z-W|}h(Z)$ is called the *conditional multivariate mutual information*, thus, $-d_h(W)$ is a saturated conditional multivariate mutual information. We show in the full paper [18] that $-d_h(W)$ is precisely the I-measure of an atom in I-measure theory [40].

Once we have motivated the critical role of the negated densities $-d_h(W)$, we define an *I-measure constraint* to be an arbitrary sum $\sigma = -\sum_i d_h(W_i)$; the exact constraint is the assertion $\sigma = 0$, while an approximate constraint asserts some bound, $\sigma \leq c$. The differential constraints [33] are special cases of I-measure constraints. Any CI constraint is also a special case of an I-measure, for example $h(Y|X) = -\sum_{W:X\subseteq W, Y\not\subseteq W} d_h(W)$, and $I_h(Y;Z|X) = -\sum_{W:X\subseteq W, X\not\subseteq W, Y\not\subseteq W} d_h(W)$. Since $d_h \leq 0$ for $h \in \mathcal{P}_n$, it follows that all I-measure constraints are ≥ 0. We prove[8]:

▶ **Theorem 23.** *Exact implications of I-measure constraints admit a 1-relaxation in $\mathcal{P}_n$.*

Proof. Consider an implication $\Sigma \Rightarrow \tau$ where all constraints in Σ, τ are I-measure constraints. Let $\tau = -\sum_i d_h(W_i)$. Then, for every i, there exists some constraint $\sigma = -\sum_j d_h(W_j) \in \Sigma$ such that $W_i = W_j$ for some j, proving the theorem. If not, then for the step function $h \overset{\text{def}}{=} h_{W_i}$ we have $h(\sigma) = 0$ forall $\sigma \in \Sigma$, yet $h(\tau) = 1$, contradicting the assumption $P_n \models \Sigma \Rightarrow \sigma$. ◀

▶ **Example 24.** Consider Example 4.3 in [33]: $d_1 = f(A) + f(ABCD) - f(ABC) - f(ACD)$, $d_2 = f(C) - f(CD)$, and $d = f(AB) - f(ABD)$. Sayrafi and Van Gucht prove $d_1 = d_2 = 0$ implies $d = 0$ for all support functions f. The quantity d_1 represents the number of baskets that contain A, but do not contain BC nor CD, while d_2 is the number of baskets that contain C but not D. Our theorem converts the exact implication into an inequality as follows. Denote by $\sigma_1 \overset{\text{def}}{=} I_h(BC;CD|A), \sigma_2 \overset{\text{def}}{=} h(D|C), \tau \overset{\text{def}}{=} h(D|AB)$, $P_n \models (\sigma_1 = \sigma_2 = 0 \Rightarrow \tau = 0)$ relaxes to $P_n \models \sigma_1 + \sigma_2 \geq \tau$, which translates into $d_1 + d_2 \leq d$ forall support functions f.

8 Discussion and Future Work

Number of Repairs. A natural way to measure the degree of a constraint in a relation instance R is by the number of repairs needed to enforce the constraint on R. In the case of a key constraint, $X \to Y$, where $XY = \Omega$, our information-theoretic measure is naturally related to the number of repairs, as follows. If $h(Y|X) = c$, where h is the entropy of the empirical distribution on R, then one can check $|R|/|\Pi_X(R)| \leq 2^c$. Thus, the number of repairs $|R| - |\Pi_X(R)|$ is at most $(2^c - 1)|\Pi_X(R)|$. We leave for future work an exploration of the connections between number of repairs and information theoretic measures.

Small Model Property. We have proven in Sec. 4 that several classes of implications (including saturated CIs, FDs, and MVDs) have a "small model" property: if the implication holds for all uniform, 2-tuple distributions, then it holds in general. In other words, it suffices to check the implication on the step functions $\mathcal{S}_n$. One question is whether this small model property continues to hold for other tractable classes of implications in the literature. For example, Geiger and Pearl [12] give an axiomatization (and, hence, a decision procedure) for *marginal* CIs. However, marginal CIs do not have the same small model property. Indeed, the implication $(X \perp Y)\&(X \perp Z) \Rightarrow (X \perp YZ)$ holds for all uniform 2-tuple distributions (because $I_h(X;YZ) \leq I_h(X;Y) + I_h(X;Z)$ holds for all step functions), however it fails for the "parity distribution" in Fig.1(b). We leave for future work an investigation of the small model property for other classes of constraints.

Proof Techniques. Since we had to integrate concepts from both database theory and information theory, we had to make a choice of which proof techniques to favor. In particular, $\mathcal{P}_n$, the cone closure of the step functions, is better known in information theory as the *set of entropic functions with a non-negative I-measure*. After trying both alternatives, we have chosen to favor the step functions in most of the proofs, because of their connection to 2-tuple relations. We explain in the full paper [18] the connection to the I-measure, and include the proof of Th. 11, which is easier to express in that language.

[8] A version of this proof based on I-measure theory appears in the full version of the paper [18].

Bounds on the factor λ. In the early stages of this work we conjectured that all CIs in Γ_n admit 1-relaxation, until we discovered the counterexample in Th. 20, where $\lambda = 3$. On the other hand, the only general upper bound is $(2^n)!$. None of them is likely to be tight. We leave for future work the task of finding tighter bounds for λ.

References

1 William Ward Armstrong and Claude Delobel. Decomposition and Functional Dependencies in Relations. *ACM Trans. Database Syst.*, 5(4):404–430, 1980. doi:10.1145/320610.320620.

2 Catriel Beeri, Ronald Fagin, and John H. Howard. A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations. In *Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data, Toronto, Canada, August 3-5, 1977.*, pages 47–61, 1977. doi:10.1145/509404.509414.

3 Tobias Bleifuß, Susanne Bülow, Johannes Frohnhofen, Julian Risch, Georg Wiese, Sebastian Kruse, Thorsten Papenbrock, and Felix Naumann. Approximate Discovery of Functional Dependencies for Large Datasets. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 1803–1812, 2016. doi:10.1145/2983323.2983781.

4 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004. doi:10.1017/CBO9780511804441.

5 Toon Calders and Jan Paredaens. Axiomatization of Frequent Sets. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, pages 204–218, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

6 Terence Chan. Recent Progresses in Characterising Information Inequalities. *Entropy*, 13(2):379–401, 2011. doi:10.3390/e13020379.

7 Xu Chu, Ihab F. Ilyas, Paolo Papotti, and Yin Ye. RuleMiner: Data quality rules discovery. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1222–1225, 2014. doi:10.1109/ICDE.2014.6816746.

8 Mehmet M. Dalkilic and Edward L. Roberston. Information Dependencies. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '00, pages 245–253, New York, NY, USA, 2000. ACM. doi:10.1145/335168.336059.

9 Randall Dougherty, Christopher F. Freiling, and Kenneth Zeger. Linear rank inequalities on five or more variables. *CoRR*, abs/0910.0284, 2009. arXiv:0910.0284.

10 Ronald Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982. doi:10.1145/322344.322347.

11 Abram Friesen and Pedro Domingos. The Sum-Product Theorem: A Foundation for Learning Tractable Models. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1909–1918, New York, New York, USA, June 2016. PMLR. URL: http://proceedings.mlr.press/v48/friesen16.html.

12 Dan Geiger and Judea Pearl. Logical and Algorithmic Properties of Conditional Independence and Graphical Models. *The Annals of Statistics*, 21(4):2001–2021, 1993. URL: http://www.jstor.org/stable/2242326.

13 Chris Giannella and Edward L. Robertson. On approximation measures for functional dependencies. *Inf. Syst.*, 29(6):483–507, 2004. doi:10.1016/j.is.2003.10.006.

14 Marc Gyssens, Mathias Niepert, and Dirk Van Gucht. On the completeness of the semigraphoid axioms for deriving arbitrary from saturated conditional independence statements. *Inf. Process. Lett.*, 114(11):628–633, 2014. doi:10.1016/j.ipl.2014.05.010.

15 Christian Herrmann. Corrigendum to "On the undecidability of implications between embedded multivalued database dependencies" [Inform. and Comput. 122(1995) 221-235]. *Inf. Comput.*, 204(12):1847–1851, 2006. doi:10.1016/j.ic.2006.09.002.

16 Ihab F. Ilyas and Xu Chu. Trends in Cleaning Relational Data: Consistency and Deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015. doi:10.1561/1900000045.

17 Tarik Kaced and Andrei E. Romashchenko. Conditional Information Inequalities for Entropic and Almost Entropic Points. *IEEE Trans. Information Theory*, 59(11):7149–7167, 2013. doi:10.1109/TIT.2013.2274614.

18 Batya Kenig and Dan Suciu. Integrity Constraints Revisited: From Exact to Approximate Implication. *CoRR*, abs/1812.09987, 2018. arXiv:1812.09987.

19 Juha Kontinen, Sebastian Link, and Jouko Väänänen. Independence in Database Relations. In Leonid Libkin, Ulrich Kohlenbach, and Ruy de Queiroz, editors, *Logic, Language, Information, and Computation*, pages 179–193, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

20 Sebastian Kruse and Felix Naumann. Efficient Discovery of Approximate Dependencies. *PVLDB*, 11(7):759–772, 2018. doi:10.14778/3192965.3192968.

21 Tony T. Lee. An Information-Theoretic Analysis of Relational Databases - Part I: Data Dependencies and Information Metric. *IEEE Trans. Software Eng.*, 13(10):1049–1061, 1987. doi:10.1109/TSE.1987.232847.

22 Chen Li, editor. *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*. ACM, 2005. URL: http://dl.acm.org/citation.cfm?id=1065167.

23 David Maier. *Theory of Relational Databases.* Computer Science Pr, 1983.

24 Francesco M. Malvestuto. Statistical treatment of the information content of a database. *Inf. Syst.*, 11(3):211–223, 1986. doi:10.1016/0306-4379(86)90029-3.

25 F. Matúš. Infinitely Many Information Inequalities. In *2007 IEEE International Symposium on Information Theory*, pages 41–44, June 2007. doi:10.1109/ISIT.2007.4557201.

26 Alejandro Molina, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting. Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3828–3835, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16865.

27 Judea Pearl and Azaria Paz. Graphoids: Graph-Based Logic for Reasoning about Relevance Relations or When would x tell you more about y if you already know z? In *ECAI*, pages 357–363, 1986.

28 Jean-Philippe Pellet and André Elisseeff. Using Markov Blankets for Causal Structure Learning. *Journal of Machine Learning Research*, 9:1295–1342, 2008. doi:10.1145/1390681.1442776.

29 Hoifung Poon and Pedro M. Domingos. Sum-Product Networks: A New Deep Architecture. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 337–346, 2011. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2194&proceeding_id=27.

30 Yehoshua Sagiv, Claude Delobel, D. Scott Parker, Jr., and Ronald Fagin. An Equivalence Between Relational Database Dependencies and a Fragment of Propositional Logic. *J. ACM*, 28(3):435–453, July 1981. doi:10.1145/322261.322263.

31 Babak Salimi, Johannes Gehrke, and Dan Suciu. Bias in OLAP Queries: Detection, Explanation, and Removal. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1021–1035, 2018. doi:10.1145/3183713.3196914.

32 Bassem Sayrafi, Dirk Van Gucht, and Marc Gyssens. The implication problem for measure-based constraints. *Inf. Syst.*, 33(2):221–239, 2008. doi:10.1016/j.is.2007.07.005.

33 Bassem Sayrafi and Dirk Van Gucht. Differential Constraints. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, pages 348–357, New York, NY, USA, 2005. ACM. doi:10.1145/1065167.1065213.

34 Yannis Sismanis, Paul Brown, Peter J. Haas, and Berthold Reinwald. GORDIAN: efficient and scalable discovery of composite keys. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 691–702, 2006. URL: http://dl.acm.org/citation.cfm?id=1164187.

35 Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search.* MIT press, 2000.

36 Milan Studený. Conditional Independence Relations have no finite complete characterization. In *11th Prague Conf. Information Theory, Statistical Decision Foundation and Random Processes*, pages 377–396. Norwell, MA, 1990.

37 Milan Studený. Convex cones in finite-dimensional real vector spaces. *Kybernetika*, 29(2):180–200, 1993. URL: http://www.kybernetika.cz/content/1993/2/180.

38 S. K. Michael Wong, Cory J. Butz, and Dan Wu. On the implication problem for probabilistic conditional independency. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 30(6):785–805, 2000. doi:10.1109/3468.895901.

39 Raymond W. Yeung. A framework for linear information inequalities. *IEEE Trans. Information Theory*, 43(6):1924–1934, 1997. doi:10.1109/18.641556.

40 Raymond W. Yeung. *Information Theory and Network Coding.* Springer Publishing Company, Incorporated, 1 edition, 2008.

41 Zhen Zhang and Raymond W. Yeung. A non-Shannon-type conditional inequality of information quantities. *IEEE Trans. Information Theory*, 43(6):1982–1986, 1997. doi:10.1109/18.641561.

42 Zhen Zhang and Raymond W. Yeung. On Characterization of Entropy Function via Information Inequalities. *IEEE Trans. Information Theory*, 44(4):1440–1452, 1998. doi:10.1109/18.681320.

A Example for Section 5

Theorem 16 states that some EI does not relax to an AI. The example, based on [17], uses 4 random variables, hence it is essentially a statement about $\mathbb{R}^{15}$, making it difficult to visualize its underlying geometry. We give here a simpler counterexample, in $\mathbb{R}^3$, albeit in a vector space unrelated to information theory.

Let $K \stackrel{\text{def}}{=} \{\begin{pmatrix} x_1 & x_2 \\ x_2 & x_3 \end{pmatrix} \mid x_1 \geq 0, x_3 \geq 0, x_1x_3 \geq x_2^2\}$ be the cone of semi-positive definite 2×2 matrices. This is known to be a convex cone, also called the *positive semi-definite cone* [4]. Equivalently, we view K as a cone in $\mathbb{R}^3$, namely $K = \{(x_1, x_2, x_3) \mid x_1 \geq 0, x_3 \geq 0, x_1x_3 \geq x_2^2\}$ Then K satisfies the following Exact Implication:

$$\forall (x_1, x_2, x_3) \in K : \quad x_1 \leq 0 \Rightarrow x_2 \leq 0$$

because $x_1 \leq 0$ is equivalent to $x_1 = 0$, implying $x_2^2 \leq 0$ thus $x_2 = 0$. However, K does not satisfy the corresponding Approximate Implication, more precisely the following is false:

$$\exists \lambda > 0, \forall (x_1, x_2, x_3) \in K : \quad x_2 \leq \lambda x_1 \qquad \text{(this is false)}$$

Indeed, for every choice of $\lambda > 0$, choose $0 < x_1 < 1/\lambda$, and let $x_2 = 1$, $x_3 = 1/x_1$. Then $(x_1, x_2, x_3) \in K$, yet $x_2 > \lambda x_1$.

Instead, Theorem 18 states that, for every $\varepsilon > 0$, there exists $\lambda > 0$, and an error term $\mathbf{e} = (e_1, e_2, e_3)$, with $e_1, e_2, e_3 < \varepsilon$, such that:

$$\forall \mathbf{x} \in K : \qquad x_2 \leq \lambda x_1 + e_1x_1 + e_2x_2 + e_3x_3$$

In our simple example, this statement is easily verified. Indeed, given $\varepsilon > 0$, define $\lambda \stackrel{\text{def}}{=} 1/\varepsilon$ and $\mathbf{e} \stackrel{\text{def}}{=} (0, 0, \varepsilon/4)$. Then $\lambda x_1 + (e_1x_1 + e_2x_2 + e_3x_3) = \frac{1}{\varepsilon}x_1 + \frac{\varepsilon}{4}x_3 \geq 2\sqrt{\frac{1}{4}x_1x_3} = \sqrt{x_1x_3} \geq$

$|x_2| \geq x_2$ proving the AI. This simple example explains why the error term is necessary in Theorem 18.

Geometrically, the error term is necessary whenever the cone $\textbf{conhull}\,(K^* \cup L)$ used in the proof of Theorem 18 is not closed. In our example $K^* = K$ because the positive, semi-definite cone is self-dual [4], and $L \overset{\text{def}}{=} \{(-1,0,0)\}$, and will check that $\textbf{conhull}\,(K^* \cup L) = \textbf{conhull}\,(K \cup L)$ is not closed. For that, consider the sequence $\mathbf{y}_n \overset{\text{def}}{=} (0,1,1/n)$. On one hand, we have $(n,1,1/n) \in K$, therefore $\mathbf{y}_n = (0,1,1/n) = (n,1,1/n) + n(-1,0,0) \in \textbf{conhull}\,(K \cup L)$. On the other hand, $\lim_{n\to\infty} \mathbf{y}_n = (0,1,0) \notin \textbf{conhull}\,(K \cup L)$. To see this, it suffices to notice that every vector in $\textbf{conhull}\,(K \cup L)$ has the form $(x_1 - \lambda, x_2, x_3)$ for some $(x_1, x_2, x_3) \in K$ and $\lambda \geq 0$, and, therefore, it satisfies the EI $x_3 = 0 \Rightarrow x_2 = 0$; our limit vector $(0,1,0)$ does not satisfy this EI, hence it is not in $\textbf{conhull}\,(K \cup L)$. This proves the fact that $\textbf{conhull}\,(K^* \cup L)$ is not closed, and hence taking its closure in the proof of Theorem 18 is necessary, leading to the error term.

B Proof of Cone Properties and Identities from Section 5

We need several known properties of cones; we give the proofs of some of them, for completeness, and refer for the others to [4].

▶ **Theorem 25.** *Let $K, K_1, K_2 \subseteq \mathbb{R}^n$. The following holds.*

1. $K_1 \subseteq K_2 \Rightarrow K_2^* \subseteq K_1^*$
2. $K_1 \subseteq K_2^*$ *iff* $K_1^* \supseteq K_2$.
3. *If* $K \neq \emptyset$ *then* $\textbf{cl}\left(\textbf{conhull}\,(K)\right) = K^{**}$.
4. $K^* = \Big(\textbf{cl}\left(\textbf{conhull}\,(K)\right)\Big)^*$.
5. *If K_1 and K_2 are closed, convex cones then:* $(K_1 \cap K_2)^* = \Big(\textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)\Big)$.
6. *A cone K is finitely generated iff K^* is finitely generated.*

Proof.

Proof of (1) Let $x \in K_2^*$, and let $y \in K_1$. Since $x \cdot z \geq 0$ for every vector $z \in K_2$, and since $K_2 \supseteq K_1$ then $x \cdot y \geq 0$ as well. Therefore, $x \in K_1^*$.

Proof of (2) Both statements assert $\forall x \in K_1, \forall y \in K_2, x{\cdot}y \geq 0$.

We omit the proofs of (3) and (4) and refer to [4].

Proof of (5) By definition of union we have that: $K_i^* \subseteq \textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)$ for $i \in \{1,2\}$. By item (1) we have that $K_i^{**} \supseteq \Big(\textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)\Big)^*$ for $i \in \{1,2\}$. Since K_1 and K_2 are closed convex cones then by item (3) it holds that $K_1^{**} = K_1$ and $K_2^{**} = K_2$. Therefore, for $i \in \{1,2\}$ we have that $K_i \supseteq \Big(\textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)\Big)^*$. From the above we get that $K_1 \cap K_2 \supseteq \Big(\textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)\Big)^*$. By property (1) we get that:

$(K_1 \cap K_2)^* \subseteq \Big(\textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)\Big)^{**}$. By property (3) we have that

$$\Big(\textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)\Big)^{**} = \textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right).$$

Overall, we get that $(K_1 \cap K_2)^* \subseteq \textbf{cl}\left(\textbf{conhull}\,(K_1^* \cup K_2^*)\right)$.

Proof of (6). Suppose K is finitely generated, $K = \textbf{conhull}(\{x_1, \ldots, x_n\})$. Then $K^* = \{y \mid x_1 \cdot y \geq 0, \ldots, x_n \cdot y \geq 0\}$, hence it is polyhedral by definition. ◀

C Proof of Lemma 22

Recall that $h(\Omega|Z) = h(\Omega) - h(Z)$. Define $\delta_h(W)$ to be the Möbius inverse of $h(\Omega|W)$, in other words:

$$\forall W : \delta_h(W) = \sum_{Z:W\subseteq Z} (-1)^{|Z-W|} h(\Omega|Z) \qquad \forall W : h(\Omega|W) = \sum_{Z:W\subseteq Z} \delta_h(Z) \tag{15}$$

We claim that, for $W \subsetneq \Omega$, $\delta_h(W) = -d_h(W)$. Indeed, $\delta_h(W) = \sum_{Z:W\subseteq Z}(-1)^{|Z-W|}h(\Omega|Z) = h(\Omega)\sum_{Z:W\subseteq Z\subseteq\Omega}(-1)^{|Z-W|} - d_h(W)$ and the claim follows from $\sum_{Z:W\subseteq Z\subseteq\Omega}(-1)^{|Z-W|} = 0$ when $W \subsetneq \Omega$. We prove that $h = \sum_{Z\subsetneq\Omega} \delta_h(Z) \cdot h_Z$, by using the right part of Eq.(15):

$$h(W) =h(\Omega|\emptyset) - h(\Omega|W) = \sum_{Z} \delta_h(Z) - \sum_{Z:W\subseteq Z} \delta_h(Z) = \sum_{Z:W\not\subseteq Z} \delta_h(W) = \sum_{Z} \delta_h(Z) \cdot h_Z(W)$$

because $h_Z(W) = 1$ iff $W \not\subseteq Z$, and $h(W) = 0$ otherwise. This proves that the $2^n - 1$ step functions span the vector space $\{h \in R^{2^n} \mid h(\emptyset) = 0\}$; since the latter has dimension $2^n - 1$, it follows that the step functions form a basis.

Datalog with Negation and Monotonicity

Bas Ketsman[1]
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Christoph Koch
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
christoph.koch@epfl.ch

Abstract

Positive Datalog has several nice properties that are lost when the language is extended with negation. One example is that fixpoints of positive Datalog programs are robust w.r.t. the order in which facts are inserted, which facilitates efficient evaluation of such programs in distributed environments. A natural question to ask, given a (stratified) Datalog program with negation, is whether an equivalent positive Datalog program exists.

In this context, it is known that positive Datalog can express only a strict subset of the monotone queries, yet the exact relationship between the positive and monotone fragments of semi-positive and stratified Datalog was previously left open. In this paper, we complete the picture by showing that monotone queries expressible in semi-positive Datalog exist which are not expressible in positive Datalog. To provide additional insight into this gap, we also characterize a large class of semi-positive Datalog programs for which the dichotomy 'monotone if and only if rewritable to positive Datalog' holds. Finally, we give best-effort techniques to reduce the amount of negation that is exhibited by a program, even if the program is not monotone.

2012 ACM Subject Classification Information systems → Relational database query languages; Theory of computation → Constraint and logic programming

Keywords and phrases Datalog, Monotonicity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.19

1 Introduction

Within classical model theory, several results exist that equate syntactic with semantic restrictions of first-order logic (FOL). One example is the homomorphism preservation theorem [22], which states that the fragment of FOL that is preserved under homomorphisms has the same expressive power as the set of existential positive FOL formulas. Analogously, Lyndon's preservation theorem states that the fragment preserved under surjective homomorphisms equals the set of positive FOL formulas. (Recall that query q is preserved under homomorphisms if, for all instances I, J and mappings h, $h(I) \subseteq J$ implies $h(q(I)) \subseteq q(J)$.)

For finite structures, which are the central interest in database theory, it is well-known that most of such equalities fail. For example, for Lyndon's theorem this was shown in the 80's by Ajtai and Gurevich [4] (and by Stolboushkin [25], using a simplified counterexample). One of the only exceptions is the homomorphism theorem, which Rossman [22] proved to hold in the finite as well. A similar result exists in the context of Datalog. Here, Feder and Vardi [12] showed that the fragment of semi-positive Datalog (in which negation is allowed only over extensional atoms) preserved under homomorphisms has the same expressive power as the positive fragment of Datalog. That the latter result does not transfer to general fixpoint logics was shown by Dawar and Kreutzer [10].

[1] Currently at Vrije Universiteit Brussel, Brussels, Belgium

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 19; pp. 19:1–19:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

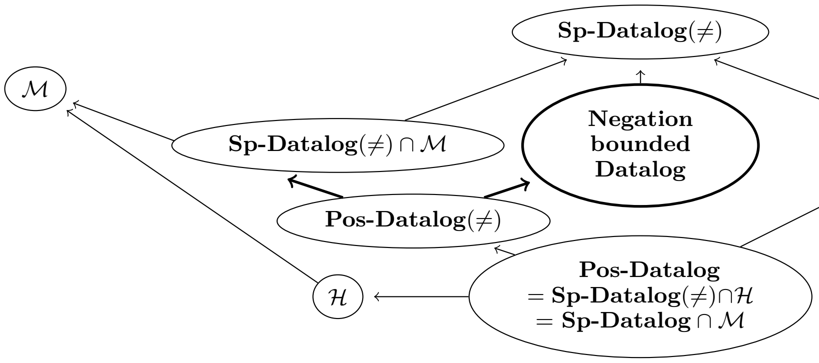

Sp-Datalog(≠)
$\mathcal{M}$
Sp-Datalog(≠) ∩ $\mathcal{M}$
Negation bounded Datalog
Sp-Datalog
Pos-Datalog(≠)
Pos-Datalog = Sp-Datalog(≠)∩$\mathcal{H}$ = Sp-Datalog ∩ $\mathcal{M}$
$\mathcal{H}$

Figure 1 Schematic overview of the different fragments of semi-positive Datalog that we consider and the relationship between their expressive power and the classes of homomorphically closed queries (denoted $\mathcal{H}$) and monotone queries (denoted $\mathcal{M}$). Here, arrows mean subsumption.

In this paper, we study the relationship between the positive and the monotone fragment of stratified Datalog (which allows negation in a stratified fashion) and semi-positive Datalog (which allows negation over existential predicates only; thus consisting of the single stratum programs). Their positive fragment is the fragment in which all forms of negation are forbidden. Their monotone fragment is the subclass of programs expressing a monotone query q; thus with $I \subseteq J$ implying $q(I) \subseteq q(J)$ for all instances I and J. It is known that positive Datalog can only express monotone queries [3, 5, 18], and that some polynomial time computable monotone queries are not expressible in positive Datalog [3, 10]. To the best of our knowledge it remained open whether such queries exist that are themselves expressible in stratified Datalog.

Our first set of contributions addresses this gap and proves the relationships that were previously left open:

(a) The monotone fragment of stratified Datalog without inequalities is strictly more expressive than positive Datalog, even when restricted to two-stratum programs. (Theorem 7.2)

(b) The monotone fragment of stratified Datalog with inequalities is strictly more expressive than positive Datalog, even when restricted to single-stratum programs. (Theorem 4.1)

(c) The monotone fragment of semi-positive Datalog without inequalities is equally expressive as positive Datalog without inequalities. (Theorem 5.1)

Motivated by contributions (a) and (b), we explore further the expressivity gap between the monotone fragment of semi-positive Datalog and positive Datalog:

(d) Based on the notion conflict-freedom, we identify a large fragment of semi-positive Datalog, called negation-bounded Datalog, for which the two restrictions coincide. (Theorem 6.2) We show that deciding conflict freedom is EXP-complete, respectively, *co*NP-complete if a bound is assumed on the arities of relations. (Theorem 3.13)

Our motivation to study the monotone fragment and the positive fragment of Datalog with negation is driven by an underlying interest in the declarative networking paradigm [1, 19], which is concerned with the design of network programs in extensions of Datalog. In this setting, it is folklore knowledge that positive Datalog programs can be computed efficiently via asynchronous pipelined joins [19, 16], because Datalog rules without negation can fire independently of each other, without a need for synchronisation. This is in stark contrast to the case where rules have negated atoms, as then a round of consensus is needed to reach agreement on the absence of facts. A model-theoretic explanation for this observation is that fixpoints of positive Datalog programs are robust w.r.t. the order in which facts are inserted. Monotonicity on the other hand is recognized as the facilitating property for this observation and as theoretical upper-bound on what can be computed efficiently [6, 16, 17, 28]. Not surprisingly, the terms positive and monotone are often mentioned in the same breath.

Our contributions (a-b) shows that the terms positive and monotone do not always interchange, even in the context of a language as simple as semi-positive Datalog.

We note that avoiding negation completely is usually not desirable, as this puts a significant limitation on the type of programs that can be formulated. Nevertheless, it is common in practice to strive for both goals: To find an optimal rewrite for a program (*i.e.*, equivalent with less exposure to negation) as well as to give up robustness in favour of computational properties (*i.e.*, less exposure to negation at the cost of some expressive power). A real-world example of the latter is the choice of the 2-phase commit (2PC) protocol to support atomicity of distributed transactions: While it is well-known that 2PC blocks under certain types of failures, it is usually favoured over more robust alternatives, because their robustness comes at the cost of additional rounds of consensus, and thus higher latency [14, 15]. When formulated in a logical language, consensus is recognisable as universal quantification, which in Datalog-like languages is encoded through negation.

In non-distributed contexts a similar trade-off exists, which is in terms of the number of strata that the program admits. Indeed, while there are several well-known optimization techniques to evaluate single-stratum programs, like semi-naive evaluation and magic-set optimization, traditional Datalog engines evaluate the strata of a stratified Datalog program one after another and thus benefit from techniques that reduce the number of strata.

Our final contribution elaborates on this by addressing the negation elimination problem:

(e) We describe how the exposure to negation in stratified Datalog programs can be reduced even if the program is not monotone. Together, these techniques form a best effort procedure to remove negated atoms from programs, which we show to run with exponential space (respectively polynomial space if a bound on the arity of relations is assumed). (Theorem 8.5) Given that almost all properties for Datalog are undecidable, this is essentially the best one can hope for.

Outline

In Section 2, we give the essential definitions that are used throughout the paper. Section 3 covers the concept of conflict-free proof trees, which is central in several of our results. In Section 4, we show that the monotone fragment of semi-positive Datalog is not expressible in positive Datalog. In Section 5 and Section 6, we give positive results on this equality for when no inequalities occur in the considered programs, or when a bound exists on the number of negated facts that can occur in proof trees of a program. Finally, in Section 7, we describe best-effort techniques that can be used to remove negation from programs independent of whether these programs are monotone.

2 Preliminaries

In this section, we give an overview of the necessary concepts and definitions that are used throughout the paper.

For positive integers n, henceforth we abbreviate the set $\{1, \dots, n\}$ by $[n]$.

2.1 Schemas and Instances

As usual, a *(database) schema* σ is a set of relation names R with associated arities $\mathit{arity}(R)$. We often write $R^{(r)} \in \sigma$, as abbreviation for $R \in \sigma$ with $\mathit{arity}(R) = r$.

Throughout the paper, we assume the existence of an infinite domain **dom** of data values. Given schema σ, a *fact* $\boldsymbol{f}$ over σ is then defined as a tuple $\boldsymbol{f} := R(\bar{t})$, with $R^{(r)} \in \sigma$ and $\bar{t} \in \mathbf{dom}^r$. By $\mathit{Facts}(\sigma)$ we denote the infinite set consisting of all facts over σ. A *(database) instance* I over σ is a finite subset of $\mathit{Facts}(\sigma)$.

2.2 Queries

A *query* q is a mapping from instances over some database schema σ_1 to instances over another database schema σ_2. We call σ_1 the *input schema* and σ_2 the *output schema* of q. As usual, we assume queries to be *generic*, which means that $\pi(q(I)) = q(\pi(I))$, for every permutation π of **dom**.

For two instances I and J over σ_1, we call a mapping h a *homomorphism from I to J*, if $h(I) \subseteq J$. We say that query q is *preserved under homomorphisms* (also called *strongly monotone* [3]) if for every homomorphism h from some instance I to instance J, $h(q(I)) \subseteq q(J)$. We say that q is *monotone* if $I \subseteq J$ implies $q(I) \subseteq q(J)$, for every pair of instances I and J, Henceforth, we denote the class of all queries that are preserved under homomorphisms by $\mathcal{H}$ and the class of all monotone queries by $\mathcal{M}$.

A query q is *contained* in a query q', denoted $q \subseteq q'$, if $q(I) \subseteq q'(I)$ for every instance I.

2.3 Datalog with Negation

We define semi-positive Datalog. For this, let **var** be an infinite domain of variables, disjoint from **dom**, and let σ be a schema. An *atom $R(\overline{x})$ over σ* consists of a relation name $R^{(r)} \in \sigma$ and a tuple $\overline{x}$ from $\mathbf{var}^r$. We do not allow constants in atoms. For a set U of atoms, we write $\mathit{Vars}(U)$ to denote the set of all variables used by the atoms in U.

A *Datalog rule* τ over σ has the following form:

$$H(\overline{y}) \leftarrow R_1(\overline{x_1}), \ldots, R_\ell(\overline{x_k}), \neg S_1(\overline{z_1}), \ldots, \neg S_m(\overline{z_m}), \beta_1, \ldots, \beta_n.$$

Here, for every $i \in [\ell]$ and $j \in [m]$, $H(\overline{y})$, $R_i(\overline{x_i})$ and $S_j(\overline{z_j})$ are atoms over σ, and, for every $k \in [n]$, β_i is an inequality of the form $x \neq y$, with $\{x, y\} \subseteq \mathbf{var}$. Henceforth, we also refer to $H(\overline{y})$ by head_τ (the *head* of τ); to $\{R_i(\overline{x_i}) \mid i \in [\ell]\}$ by Pos_τ (the *positive body atoms in τ*); to $\{S_i(\overline{z_i}) \mid i \in [m]\}$ by Neg_τ (the *negated body atoms in τ*); and finally to $\{\beta_1, \ldots, \beta_n\}$ by Ineq_τ (the *inequalities in τ*). As usual, we only consider safe rules, thus with $\mathit{Vars}(\mathit{Neg}_\tau \cup \mathit{Ineq}_\tau \cup \{\mathit{head}_\tau\}) \subseteq \mathit{Vars}(\mathit{Pos}_\tau)$.

For a schema σ, a *Semi-positive Datalog program P over σ* is a set of Datalog rules P. As usual, we call relation names from σ that occur in the head of a rule in P *intensional* and all others *extensional.* For rules τ in P, the set Neg_τ must contain only atoms with extensional relation names. By *Sp-Datalog*($\neq$) we denote the class of all semi-positive Datalog programs. We also consider the following subclasses: *Sp-Datalog* (semi-positive Datalog without inequalities) denotes the programs in which Ineq_τ is empty for every rule τ; *Pos-Datalog*($\neq$) (positive Datalog) denotes the programs in which Neg_τ is empty for every rule τ; and *Pos-Datalog* (Positive Datalog without inequalities) denotes the intersection of the latter two, thus in which Ineq_τ and Neg_τ are empty for every rule τ.

Since we are interested in Datalog programs that express queries, we assume that the schema σ that a Datalog program P is defined over has distinguished input and output relation names. We denote these by $\mathit{in}(P) \subseteq \sigma$ and $\mathit{out}(P) \subseteq \sigma$, respectively. When not explicitly mentioned, we assume that $\mathit{in}(P)$ coincides with the extensional relation names in P and that $\mathit{out}(P) = \{\mathsf{Output}^{(k)}\}$, for some integer $k \geq 0$.

▶ **Example 2.1.** As a running example throughout this paper, we consider semi-positive Datalog program P_Δ. This program is defined over schema $\sigma := \{\mathsf{Edge}^{(2)}, \mathsf{T1}^{(2)}, \mathsf{T2}^{(2)}, \mathsf{Output}^{(2)}\}$, with $\mathit{in}(P_\Delta) = \{\mathsf{Edge}\}$ and $\mathit{out}(P_\Delta) = \{\mathsf{Output}\}$, and has the following rules:

$$\mathsf{T1}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \neg\mathsf{Edge}(\mathsf{z},\mathsf{x}). \tag{1}$$

$$\mathsf{T2}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \neg\mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{z},\mathsf{x}). \tag{2}$$

$$\mathsf{Output}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{z},\mathsf{x}). \tag{3}$$

$$\mathsf{Output}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{T1}(\mathsf{x},\mathsf{y}), \mathsf{T2}(\mathsf{x},\mathsf{y}), \mathsf{x} \neq \mathsf{y}. \tag{4}$$

Intuitively, P_Δ expects a directed graph as input and asks for edges (a, b) for which one of the following properties is true: (a, b) is part of a triangle; or (a, b) is part of two open triangles, one in which the edge to a is missing, the other in which the edge from b is missing. We will later show (in Proposition 3.12) that P_Δ is of particular interest because it expresses a monotone query.

2.4 Proof Tree Semantics

The semantics of Datalog is usually defined bottom-up, in terms of an immediate consequence operator. We use the equivalent top-down definition via proof trees. (We refer to [2] for a detailed discussion on their equivalence.) For a formal definition, we first define a *prevaluation v for a Datalog rule τ* as a mapping from the variables occurring in τ to values from **dom**. A prevaluation v for τ is a *valuation for* τ if each inequality $x \neq y \in \mathit{Ineq}_\tau$ admits $v(x) \neq v(y)$. Before defining the concept proof tree, we first define a slightly weaker concept, which we call a candidate proof tree:

▶ **Definition 2.2.** *Given a fact $\boldsymbol{f}$, instance I and program $P \in \mathit{Sp\text{-}Datalog}(\neq)$, a* candidate proof tree $\mathcal{T}$ of $\boldsymbol{f}$ from I and P *is a labeled tree with the following properties:*

1. *Each vertex is labeled with a fact;*
2. *Each leaf is labeled with a fact $\boldsymbol{g}$ over an extensional relation name, and with either sign '+', if $\boldsymbol{g} \in I$, or '-', if $\boldsymbol{g} \notin I$.*
3. *The root is labeled with $\boldsymbol{f}$;*
4. *Each intermediary vertex is associated with a rule $\tau \in P$ and prevaluation v for τ, such that its label equals $v(\mathit{head}_\tau)$, and for each atom $A \in \mathit{Body}_\tau$ it has a child whose label equals $v(A)$. If A has an extensional relation name this child must have a sign that equals '+' if $A \in \mathit{Pos}_\tau$ and '-' if $A \in \mathit{Neg}_\tau$.*

Unless stated otherwise, we assume throughout the paper that the root of a candidate proof tree is always labeled with a fact from an output relation. Further, we denote by $\mathit{Fringe}^+_\mathcal{T}$ the set of all extensional facts that occur as labels for leafs in $\mathcal{T}$ with sign '+', and by $\mathit{Fringe}^-_\mathcal{T}$ the set of extensional facts for leafs with sign '−'.

▶ **Definition 2.3.** *A* proof tree $\mathcal{T}$ of $\boldsymbol{f}$ from I and $P \in \mathit{Sp\text{-}Datalog}(\neq)$ *is a candidate proof tree of $\boldsymbol{f}$ from I and P in which all prevaluations are valuations (for the respective rule), $\mathit{Fringe}^+_\mathcal{T} \subseteq I$, and $\mathit{Fringe}^-_\mathcal{T} \cap I = \emptyset$.*

We sometimes refer to a (candidate) proof tree $\mathcal{T}$ from program P without specifying a fact or instance. In that case, we assume that $\mathcal{T}$ is a (candidate) proof tree of the fact that its root is labeled with, and that it is a proof tree from *some* instance I and P.

Every semi-positive Datalog program P expresses a unique query $q_P : \mathit{in}(P) \mapsto \mathit{out}(P)$ that is defined by P and its distinguished input and output relation names. Its evaluation over instances $I \subseteq \mathit{Facts}(\mathit{in}(P))$, denoted by $q_P(I)$, is defined by the set of all facts $\boldsymbol{f} \in \mathit{Facts}(\mathit{out}(P))$ for which a proof tree of $\boldsymbol{f}$ from I and P exists. Henceforth, we say that two programs P_1 and P_2 having the same input and output relation names are *equivalent* if they express the same query, thus with $q_{P_1}(I) = q_{P_2}(I)$ for every instance $I \subseteq \mathit{Facts}(\mathit{in}(P_1))$. For every class of Datalog programs we consider also the class of queries that are expressed by these programs. To distinguish between the two, the latter are always in boldface. For example, **Sp-Datalog**($\neq$) refers to the class of queries expressible with *Sp-Datalog*($\neq$) programs.

3 Conflicts

In this section, we introduce the main machinery that we use to reason about the relationship between proof trees from different programs. We start with two simple constructions. Given a semi-positive Datalog program P, P^+ denotes the program in *Pos-Datalog*($\neq$) obtained by removing all negated atoms from rules in P; P^* denotes the program in *Pos-Datalog* obtained by removing, in addition to the negated atoms, also all inequalities from rules in P^+. We leave the schema definitions untouched, thus $\mathit{in}(P) = \mathit{in}(P^+) = \mathit{in}(P^*)$, and $\mathit{out}(P) = \mathit{out}(P^+) = \mathit{out}(P^*)$.

▶ **Example 3.1.** For an example of the constructions, take program $P_\triangle$ from Example 2.1. Then, $P_\triangle^+$ contains the following rules:

$$\begin{aligned} \mathsf{T1}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}).\\ \mathsf{T2}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{z},\mathsf{x}).\\ \mathsf{Output}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{z},\mathsf{x}).\\ \mathsf{Output}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{T1}(\mathsf{x},\mathsf{y}), \mathsf{T2}(\mathsf{x},\mathsf{y}), \mathsf{x} \neq \mathsf{y}.\end{aligned}$$

Program $P_\triangle^*$ has the next rules:

$$\begin{aligned} \mathsf{T1}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}).\\ \mathsf{T2}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{z},\mathsf{x}).\\ \mathsf{Output}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{z},\mathsf{x}).\\ \mathsf{Output}(\mathsf{x},\mathsf{y}) &\leftarrow \mathsf{T1}(\mathsf{x},\mathsf{y}), \mathsf{T2}(\mathsf{x},\mathsf{y}).\end{aligned}$$

Both constructions result in well-defined Datalog programs. Particularly notice that the constructions preserve safeness of the programs by only removing negated atoms and inequalities, whose variables all occur also in non-negated atoms (due to safeness of the original program P).

We conclude this section with the following observation:

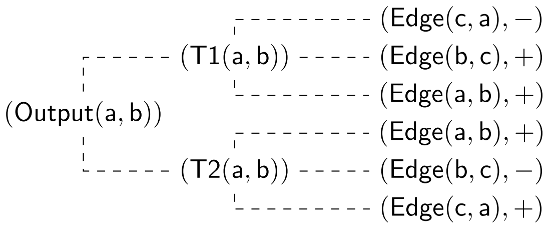
▶ **Proposition 3.2.** *Let σ be a database schema and P a semi-positive Datalog program over σ. Then, $q_P \subseteq q_{P^+} \subseteq q_{P^*}$.*

3.1 Fringe and Inequality Conflicts

To reason about the other direction of the containments in Proposition 3.2 (that is, $q_{P^*} \subseteq q_P$ and $q_{P^+} \subseteq q_P$), we need to reason about subtle differences in the proof trees that these programs admit. For this purpose, our distinction between proof trees and candidate proof trees comes in handy. First, recall that every proof tree $\mathcal{T}$ from an instance I and program $P \in$ *Sp-Datalog*($\neq$) is a candidate proof tree from I and P (by definition). The opposite direction is not true, because a candidate proof tree $\mathcal{T}$ may admit

- *fringe conflicts*, a term we use to refer to facts in $\mathit{Fringe}^{+}_{\mathcal{T}} \cap \mathit{Fringe}^{-}_{\mathcal{T}}$; and
- *inequality conflicts*, a term we use to refer to inequalities in rules (associated to vertices of $\mathcal{T}$) that are not made true by the associated prevaluation.

▶ **Example 3.3.** For an example of a proof tree with fringe conflicts, consider the proof tree for program P_{Δ} from Example 2.1 that is given below.



The following relationship applies:

▶ **Proposition 3.4.** *A candidate proof tree $\mathcal{T}$ from semi-positive Datalog program P is a proof tree from P if and only if $\mathcal{T}$ is free of fringe and inequality conflicts.*

3.2 Expansion Trees

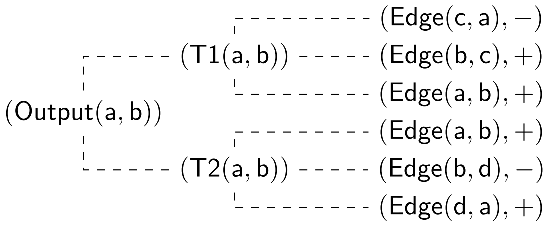
A special type of candidate proof tree is the (unfolding) expansion tree [9], which we generalize here for semi-positive Datalog:

▶ **Definition 3.5.** *An* expansion tree *is a candidate proof tree $\mathcal{T}$ in which every intermediate vertex n (including the root) is associated with a rule τ and prevaluation V, such that V maps every pair of different variables not occurring in head_{τ} onto different values that all occur only in the subtree of $\mathcal{T}$ with n as root.*

While a candidate proof tree for a semi-positive Datalog program is not necessarily an expansion tree, it always is the homomorphic image of an expansion tree. Henceforth we use the following naming conventions: We denote expansion trees by $\mathcal{T}_e$. For a mapping g over $\mathbf{dom}$ and candidate proof tree $\mathcal{T}$, $g(\mathcal{T})$ denotes the candidate tree obtained by substituting all facts $\boldsymbol{f}$ occurring as labels in $\mathcal{T}$ and all valuations v associated to vertices in $\mathcal{T}$ by their respective images $g(\boldsymbol{f})$ and $g \circ v$ under g.

▶ **Proposition 3.6.** *Let $P \in$ Sp-Datalog($\neq$). For every candidate proof tree $\mathcal{T}$ from P, there is an expansion tree $\mathcal{T}_e$ and mapping g such that $g(\mathcal{T}_e) = \mathcal{T}$. Moreover, if $\mathcal{T}$ is free of fringe and inequality conflicts, then $\mathcal{T}_e$ is free of fringe and inequality conflicts.*

▶ **Example 3.7.** The below tree is an example expansion tree for P_{Δ}.



3.3 Conflicts and Monotonicity

In the remainder of this section, we show the relevance of conflicts in the relationship between monotone and positive programs. First, we distinguish two categories of programs depending on which conflicts their candidate proof trees admit.

▶ **Definition 3.8.** *Let $P \in$ Sp-Datalog($\neq$). Program P is* conflict-free *if each candidate proof tree from P without inequality conflicts is without fringe conflicts. Program P is* free of explicit conflicts *if each expansion tree from P is without fringe and inequality conflicts.*

The intuition behind the term explicit conflict is that expansion trees are less diverged from the rules of the program than arbitrary candidate proof trees are, and that conflicts in expansion trees therefore can be observed more easily than fringe conflicts by looking at the wiring of variables throughout rules in the program.

▶ **Proposition 3.9.** *For $P \in$ Sp-Datalog($\neq$) we have the following equivalences:*

1. $q_P = q_{P^*}$ *if $q_P \in \mathcal{H}$ and P is free of explicit conflicts;*
2. $q_P = q_{P^+}$ *if $q_P \in \mathcal{M}$ and P is conflict-free.*

Proof. Since $q_P \subseteq q_{P^*}$ and $q_P \subseteq q_{P^+}$ follow from Proposition 3.2, we need to show only $q_{P^*} \subseteq q_P$ and $q_{P^+} \subseteq q_P$.

(1) Let I be an arbitrary instance and $\boldsymbol{f}$ an arbitrary fact in $q_{P^*}(I)$. Let $\mathcal{T}^*$ denote the proof tree of $\boldsymbol{f}$ from I and P^*. We show $\boldsymbol{f} \in q_P(I)$.

By Proposition 3.6, there exists an expansion tree $\mathcal{T}_e^*$ for P^* that is without fringe and inequality conflicts and a mapping g, with $g(\mathcal{T}_e^*) = \mathcal{T}^*$. It follows from the construction of P^* that $\mathcal{T}_e^*$ can be extended to an expansion tree $\mathcal{T}_e$ from P: For vertices n, let τ_n^* and V_n denote its associated rule and valuation. Then let $\mathcal{T}_e$ be the candidate proof tree obtained from $\mathcal{T}_e^*$ by replacing, for each vertex n, its rule τ_n^* by a rule τ from P with $Pos_\tau = Pos_{\tau^*}$; and by adding, for every fact $\boldsymbol{g}$ in $V_n(Neg_\tau)$, a leaf under n with label $\boldsymbol{g}$ and sign '$-$'. Note that rules τ exist by definition of P^* and that $\mathcal{T}_e$ is an expansion tree because rules in P are safe and $\mathcal{T}_e^*$ is an expansion tree. Furthermore, $root_{\mathcal{T}_e} = root_{\mathcal{T}_e^*}$ and $Fringe^+_{\mathcal{T}_e} = Fringe^+_{\mathcal{T}_e^*}$.

It follows from the assumption that P is without explicit conflicts that $\mathcal{T}_e$ is free of fringe and inequality conflicts, thus with $root_{\mathcal{T}_e} \in q_P(Fringe^+_{\mathcal{T}_e})$, and from $q_P \in \mathcal{H}$ and homomorphism g with $g(Fringe^+_{\mathcal{T}_e}) = I$, that $\boldsymbol{f} = g(root_{\mathcal{T}_e^*}) = g(root_{\mathcal{T}_e}) \in q_P(I)$.

(2) Let I be an arbitrary instance over $in(P)$, and $\boldsymbol{f}$ an arbitrary fact in $q_{P^+}(I)$.

To show $\boldsymbol{f} \in q_P(I)$, we observe that the presence of $\boldsymbol{f}$ in $q_{P^+}(I)$ implies the existence of a proof tree $\mathcal{T}^+$ for P^+ with root $\boldsymbol{f}$ and $Fringe^+_{\mathcal{T}^+} \subseteq I$. By Proposition 3.6, there is an expansion tree $\mathcal{T}_e^+$ for P^+ that is without fringe and inequality conflicts; and a mapping g, with $g(\mathcal{T}_e^+) = \mathcal{T}$. We observe that $\mathcal{T}_e^+$ can be extended to an expansion tree $\mathcal{T}_e$ for P by adding '$-$' signed leafs. Indeed, for every intermediate vertex n in $\mathcal{T}_e^+$, say with label $\boldsymbol{f}_n$ and associated rule τ_n and valuation v_n, there is a rule $\tau \in P$ that differs from τ_n only w.r.t. the set of negated atoms (by construction of P^+). Due to safeness of rules, negated atoms do not introduce new variables that are not already in τ_n, hence, for every $A \in Neg_\tau$ we just add a new leaf under vertex n with label $v_n(A)$ and sign '$-$'.

Another consequence of the safeness of rules is that g is a total mapping for $\mathcal{T}_e$. Moreover, since P is free of fringe conflicts, and $g(\mathcal{T}_e)$ is a correctly defined candidate proof tree, it must be that $g(\mathcal{T}_e)$ is free of fringe conflicts. Since all inequalities in $\mathcal{T}_e$ already exist in $\mathcal{T}_e^+$, the fact that $g(\mathcal{T}_e^+)$ is free of inequality conflicts transfers to $g(\mathcal{T}_e)$.

As a consequence, $\mathcal{T} := g(\mathcal{T}_e)$ is a proof tree for $\boldsymbol{f}$ from P, with $Fringe^+_{\mathcal{T}} = Fringe^+_{\mathcal{T}^+}$, and $Fringe^+_{\mathcal{T}} \cap Fringe^-_{\mathcal{T}} = \emptyset$, thus $\boldsymbol{f} \in q_P(Fringe^+_{\mathcal{T}})$. We conclude from $q_P \in \mathcal{M}$ and $Fringe^+_{\mathcal{T}} = Fringe^+_{\mathcal{T}^+} \subseteq I$ that $\boldsymbol{f} \in q_P(I)$. ◀

Proposition 3.9(1) reveals a relation that was also observed by Feder and Vardi [12], albeit less explicit, to show the following result:

▶ **Theorem 3.10** ([12]). ***Sp-Datalog***$(\neq) \cap \mathcal{H} =$ ***Pos-Datalog***.

More precisely, the construction in [12] implies the next proposition, which together with Proposition 3.9 is a proof for Theorem 3.10.

▶ **Proposition 3.11.** *For every program $P \in$ Sp-Datalog$(\neq)$ there is a program $P' \in$ Sp-Datalog$(\neq)$ that is free of explicit conflicts and with $q_P = q_{P'}$.*

For an example of a program that is monotone but not conflict-free, one can take program $P_\triangle$ from Example 2.1.

▶ **Proposition 3.12.** *Program $P_\triangle$ expresses a monotone query and is not conflict-free.*

Since each program in *Pos-Datalog*$(\neq)$ is conflict-free by definition, it is immediate that a query $q \in$ **Sp-Datalog**$(\neq) \cap \mathcal{M}$ is in **Pos-Datalog**$(\neq)$ if and only if it can be expressed by a program in *Sp-Datalog*$(\neq)$ that is conflict-free. We conclude this section by showing that conflict freedom is a decidable property.

▶ **Theorem 3.13.** *The problem of deciding whether a given program in Sp-Datalog$(\neq)$ is conflict-free is polynomial-time equivalent with the non-satisfiability problem for programs in Pos-Datalog$(\neq)$:*

- *EXP-complete in general; and*
- *coNP-complete if a bound on the arity of relations is assumed.*

4 Semi-Positive Datalog

In this section, we answer one of the central questions of the paper and show that not all monotone queries expressible in semi-positive Datalog have an equivalent in positive Datalog.

▶ **Theorem 4.1.** ***Sp-Datalog***$(\neq) \cap \mathcal{M} \not\subseteq$ ***Pos-Datalog***$(\neq)$.

The proof for Theorem 4.1 is similar to a recent proof by Rudolph and Thomazo [23], which shows that the homomorphically closed queries expressible in order-invariant semi-positive Datalog do not all have an equivalent in order-invariant Datalog without negation (and without inequality). Before proceeding with the details, we first give a definition of order-invariant Datalog.

4.1 Order-Invariant Semi-Positive Datalog

Let σ be a database schema and $\sigma_\leq$ the extension of σ with relation names $\mathsf{Succ}^{(2)}$, $\mathsf{Min}^{(1)}$, and $\mathsf{Max}^{(1)}$. (We assume of course that Succ, Min and Max do not already occur in σ.) Then for an instance I over σ, by $I_\leq$ we denote the extension of I over $\sigma_\leq$ in which Succ is interpreted as the successor relation of some linear order over the active domain of I, and in which Min and Max are interpreted to contain exactly the minimal, respectively maximal, value that occurs in I according to the assumed linear order.

An *order-invariant Sp-Datalog$(\neq)$ program* P over schema σ then is defined as an *Sp-Datalog*$(\neq)$ program, say P', over schema $\sigma_\leq$, whose output is independent of the chosen linear order. That is, $q_{P'}(I_\leq) = q_{P'}(I'_\leq)$, for every instance I over σ and pair of extensions $I_\leq$ and $I'_\leq$ of I. Due to the latter, the semantics of q_P itself can be defined in terms of instances I over σ, as $q_P(I) := q_P(I_\leq)$ for arbitrary extension $I_\leq$ of I. Henceforth, we refer by *Sp-Datalog*$(\leq, \neq)$ to the class of order-invariant *Sp-Datalog*$(\neq)$ programs.

4.2 The perfect Matching Problem over Ordered Graphs

The query q_{PM} that we use to show $\mathbf{Sp\text{-}Datalog}(\neq) \cap \mathcal{M} \subsetneq \mathbf{Pos\text{-}Datalog}(\neq)$ expresses a variant of the perfect matching problem. Given a graph $G = (V, E)$, the *perfect matching* problem asks whether a subset $M \subseteq E$ of edges exists such that every vertex in V is incident to exactly one edge in M. For the definition of q_{PM}, consider schemas $\sigma_1 := \{\mathsf{Edge}^{(2)}, \mathsf{Next}^{(2)}, \mathsf{First}^{(1)}, \mathsf{Last}^{(1)}\}$ and $\sigma_2 := \{\mathsf{Output}^{(0)}\}$. Given an instance I over σ_1, we denote by G_e the graph obtained by interpreting relation Edge as edge relation and its set of end-points as vertices. Graph G_n is defined analogously, by interpreting relation Next as edge relation and its set of end-points as vertices. We say that a vertex has label `first` if its associated value is in relation First and that it has label `last` if its value is in relation Last.

▶ **Definition 4.2** (Separating Query). *Let q_{PM} be the boolean query over input schema σ_1 and output schema σ_2, and with* Output() $\in q_{PM}(I)$ *if (and only if) one of the following conditions is true:*

1. *At least two different vertices have label* `first` *or* `last`*;*
2. *Some vertex in G_n has* either *label* `first` *and an incoming edge, label* `last` *and an outgoing edge, two incoming edges, two outgoing edges, or a self-loop; or*
3. *For set C, defined as the vertices in connected components of G_e that connect a vertex with label* `first` *to one with label* `last`*, graph G_e induced by the vertices in C has a perfect matching.*

Next, we show that q_{PM} has the desired properties.

▶ **Proposition 4.3.** *The following properties are true for q_{PM}:*

1. *q_{PM} is in **Sp-Datalog**$(\neq)$;*
2. *q_{PM} is monotone; and*
3. *q_{PM} is not in **Pos-Datalog**$(\neq)$.*

Proof Sketch. (1) Since the perfect matching problem is well-known to be in PTIME, we can assume an implementation in *Sp-Datalog*$(\neq)$. The latter is due to another well-known result, that the language *Sp-Datalog*$(\leq, \neq)$ captures exactly the PTIME computable queries [2].

To write a program in *Sp-Datalog*$(\neq)$ that expresses q_{PM}, we make use of program P (in which the relations Succ, Min, and Max are now intensional and no longer interpreted), and feed it a conservative fragment of the extensional relations Next, First, and Last.

(2) Monotonicity can be verified easily from Definition 4.2.

(3) The proof is analogous to a recent proof by Rudolph and Thomazo [23] for the statement $\mathbf{Sp\text{-}Datalog}(\leq) \cap \mathcal{H} \subsetneq \mathbf{Pos\text{-}Datalog}(\leq)$. While our query is slightly different to the query used in [23] and admits inequalities, it uses the same key ingredients:

(a) A result by Razborov [21], which states that no family of monotone boolean circuits exists that answers the perfect matching problem and has circuits of polynomial size in the number of input gates.

(b) The existence of an algorithm that converts programs in *Pos-Datalog*$(\neq)$ that express q_{PM} into a family of monotone boolean circuits that answers the perfect matching problem and has circuits of polynomial size in the number of input gates. ◀

5 Semi-Positive Datalog without Inequalities

This section is devoted to showing Theorem 5.1.

▶ **Theorem 5.1.** ***Sp-Datalog*** $\cap \mathcal{M} =$ ***Pos-Datalog****.*

Theorem 5.1 is a consequence of the observation that, for semi-positive Datalog without inequalities, the fragment of monotone queries and the fragment of queries preserved under homomorphisms collapses (cf. Proposition 5.2). That is, **Sp-Datalog** $\cap \mathcal{M} =$ **Sp-Datalog** $\cap \mathcal{H}$. Given this observation, Theorem 5.1 follows directly from Theorem 3.10.

▶ **Proposition 5.2.** *For a program $P \in$ Sp-Datalog, $q_P \in \mathcal{M}$ implies $q_P \in \mathcal{H}$.*

Proof. We show that for an arbitrary fact $\boldsymbol{f}$, instance I, and homomorphism h from I to $h(I)$, the fact $h(\boldsymbol{f})$ is in $q_P(h(I))$. The proof is by an iterative procedure that searches for an instance $I^{(i)}$, with the following properties:

1. There is a proof tree $\mathcal{T}$ of $\boldsymbol{f}$ from $I^{(i)}$ and P;
2. $adom(I^{(i)}) = adom(I)$;
3. $h(I^{(i)}) = h(I)$; and
4. $h(Fringe^{-}_{\mathcal{T}}) \cap h(Fringe^{+}_{\mathcal{T}}) = \emptyset$.

Suppose that the described instance $I^{(i)}$ exists, then the combination of Property (1) and Property (2) allows to apply homomorphism h to all valuations and facts associated to vertices in $\mathcal{T}$. The result is a candidate proof tree $\mathcal{T}'$ from P with $Fringe^{+}_{\mathcal{T}'} \subseteq h(I^{(i)})$ and $root_{\mathcal{T}'} = h(\boldsymbol{f})$. Since rules in P have no inequalities, $\mathcal{T}'$ is without inequality conflicts. Property (4) implies that $\mathcal{T}'$ is also without fringe conflicts, thus $\mathcal{T}'$ is a proof tree of $h(\boldsymbol{f})$ from P and $Fringe^{+}_{\mathcal{T}'}$, witnessing $h(\boldsymbol{f}) \in q_P(Fringe^{+}_{\mathcal{T}'})$. Now, the desired result $h(\boldsymbol{f}) \in q_P(h(I))$ follows from Property (3), implying $Fringe^{+}_{\mathcal{T}'} \subseteq h(I^{(i)}) = h(I)$, and $q_P \in \mathcal{M}$.

It remains to describe the procedure to find $I^{(i)}$, which uses an inductive argument taking conditions (1), (2), and (3) as invariants over the tentative instances that are being considered. As base case, we observe that the three invariants are true on I itself, by taking as $\mathcal{T}$ the proof tree of $\boldsymbol{f}$ from I and P. We now refer to I as $I^{(0)}$.

If Property (4) is true on the currently considered instance $I^{(i)}$, then we terminate the procedure. Otherwise, (†) there must be a fact $\boldsymbol{g} \in Fringe^{-}_{\mathcal{T}}$, such that $h(\boldsymbol{g}) \in h(Fringe^{+}_{\mathcal{T}}) \cap h(Fringe^{-}_{\mathcal{T}})$, with $\mathcal{T}$ the proof tree as defined by Property (1). It also follows from Property (1) that $\boldsymbol{g} \notin Fringe^{+}_{\mathcal{T}} \subseteq I^{(i)}$.

We now construct a new instance $I^{(i+1)}$ by adding $\boldsymbol{g}$ to $I^{(i)}$. Clearly, Property (3) is true, since $h(\boldsymbol{g}) \in h(Fringe^{+}_{\mathcal{T}}) \subseteq h(I^{(i)}) = h(I)$, which implies $h(I^{(i+1)}) = h(I^{(i)}) = h(I)$. Property (2) is straightforward as well, since the safeness of rules in P and $\boldsymbol{g} \in Fringe^{-}_{\mathcal{T}}$ imply $adom(\boldsymbol{g}) \subseteq adom(Fringe^{+}_{\mathcal{T}}) \subseteq adom(I^{(i)}) = adom(I)$. Finally, $I^{(i)} \subseteq I^{(i+1)}$ and $q_P \in \mathcal{M}$ imply $\boldsymbol{f} \in q_P(I^{(i+1)})$, which means that Property (1) is also as well.

Since the active domain of I is fixed and the number of facts $\boldsymbol{g} \notin I$ with $adom(\boldsymbol{g}) \subseteq adom(I)$ is finite, eventually $I^{(i)}$ cannot grow further and (†) must fail. From this, we conclude that the desired instance $I^{(i)}$ exists. ◀

6 Negation-Bounded Datalog

In Section 4, we have formally shown that some monotone queries in **Sp-Datalog**($\neq$) have no equivalent in *Pos-Datalog*($\neq$). This result implies that restricting ourselves to write programs in the positive variant of *Sp-Datalog*($\neq$) as a convenient way to write monotone programs in *Sp-Datalog*($\neq$), comes at the cost of some loss in expressive power. While a theoretician may be satisfied with this observation alone, a practitioner would likely wonder whether this gap matters in practice, for example, within a specific application domain. To help answer this question, an interesting direction is to consider conservative fragments of *Pos-Datalog*($\neq$) for which the monotone and positive fragment coincide. In Section 5, we have already seen that programs in *Sp-Datalog* have this property.

In this section, we define an orthogonal fragment, which we call negation-bounded Datalog.

▶ **Definition 6.1.** *Let $P \in$ Sp-Datalog($\neq$) be over some schema σ and $R \in \sigma$. Program P is* negation-bounded *if a positive integer k exists, such that for every instance I over σ and fact $\boldsymbol{f} \in q_P(I)$, there is a proof tree $\mathcal{T}$ of $\boldsymbol{f}$ from I and P with $|Fringe^{-}_{\mathcal{T}}| \leq k$.*

We immediately proceed with the main result of this section:

▶ **Theorem 6.2.** *For every program P in Sp-Datalog($\neq$) that is negation bounded, $q_P \in \mathcal{M}$ implies there is a program P' in Pos-Datalog($\neq$), with $q_{P'} = q_P$.*

For a proof of Theorem 6.2, we combine Proposition 3.9 with the below result.

▶ **Proposition 6.3.** *For every program P in Sp-Datalog($\neq$) that is negation bounded, there is a conflict-free program P' in Sp-Datalog($\neq$), with $q_P = q_{P'}$.*

The proof of Proposition 6.3 uses a technique that is inspired by the indexing technique in [12] to show Theorem 3.10, but rather than statically annotating relation names with associated dependencies, we encode indexes in a prefix of the intensional relations, which serve as pivot through the program evaluation. More precisely, program P' encodes the facts whose absence it observes while simulating program P in the prefix of intensional relation names and fires a rule of P in the simulation only if it is consistent with the index at hand. That is, if the index does not encode a fact that is required by the rule or any of its children. The latter is enforced via inequalities. (We note that similar techniques are used in, *e.g.*, [27, 8, 17].) As the proof of Proposition 6.3 is tedious, we illustrate the construction by an example.

▶ **Example 6.4.** Let $P_\triangle$ be again the program from Example 2.1. We notice that expansion trees for $P_\triangle$ have at most two negated atoms, corresponding, respectively, to the negated atom in the first and second rule of program $P_\triangle$. Program $P_\triangle$ is thus clearly negation bounded. After applying the construction underlying Proposition 6.3, we obtain the following rules. First, two rules to collect in relation Adom the active domain of the input instance:

$$\mathsf{Adom(x) \leftarrow Edge(x,y)}. \qquad \mathsf{Adom(x) \leftarrow Edge(y,x)}.$$

Then, for every choice $\beta \in \{x \neq z, y \neq x\}$, $\gamma \in \{y \neq z, z \neq x\}$, $\chi \in \{x \neq y, y \neq z\}$ of inequalities, we consider variants of the T1 and T2 generating rules in $P_\triangle$, in which their negated facts are encoded as a prefix in the head of the rule:

$$\mathsf{T1(z,x,x_2,y_2,x,y) \leftarrow Edge(x,y), Edge(y,z), \neg Edge(z,x), Adom(x_2), Adom(y_2)}, \beta, \gamma.$$
$$\mathsf{T2(x_1,y_1,y,z,x,y) \leftarrow Edge(x,y), \neg Edge(y,z), Edge(z,x), Adom(x_1), Adom(y_1)}, \chi, \gamma.$$

Rules without negated atoms forward the prefix of body atoms that are over intensional relation names, or (if no intensional relation name occurs in the body) generate facts with arbitrary prefix:

$$\begin{aligned}\mathsf{Output'(x_1,y_1,x_2,y_2,x,y)} &\leftarrow \mathsf{Edge(x,y), Edge(y,z), Edge(z,x),}\\ &\qquad \mathsf{Adom(x_1), Adom(y_1), Adom(x_2), Adom(y_2)}.\\ \mathsf{Output'(x_1,y_1,x_2,y_2,x,y)} &\leftarrow \mathsf{T1(x_1,y_1,x_2,y_2,x,y), T2(x_1,y_1,x_2,y_2,x,y), x \neq y}.\\ \mathsf{Output(x,y)} &\leftarrow \mathsf{Output'(x_1,y_1,x_2,y_2,x,y)}.\end{aligned}$$

Finally, we have to deal also with proof trees for $P_\triangle$ that have no leafs with sign '$-$'. To support this case, we augment the program with all rules from $P_\triangle$ that are without negation:

$$\mathsf{Output}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{z},\mathsf{x}).$$

It is easy to verify that the program consisting of the above listed rules is conflict-free, since every fringe conflict now implies an inequality conflict. Equivalence follows from the observation that the constructed program simulates $P_\triangle$ with some additional bookkeeping.

We remark that the concept negation boundedness is related to the well-known concept boundedness for Datalog programs: A program P in *Sp-Datalog*($\neq$) is *bounded* if there is a positive integer k such that, for every instance I and fact $\boldsymbol{f} \in q_P(I)$, there is a proof tree of $\boldsymbol{f}$ from I and P with depth at most k. Since the latter implies existence of a bound on the size of $\mathit{Fringe}^+_\mathcal{T}$, and thus on the domain of $\mathit{Fringe}^+_\mathcal{T} \cup \mathit{Fringe}^-_\mathcal{T}$, it follows directly that boundedness implies negation boundedness. Not surprisingly, the decision problem that asks whether a given program $P \in$ *Sp-Datalog*($\neq$) is negation bounded is undecidable.

▶ **Proposition 6.5.** *No algorithm exists that decides for an arbitrary program $P \in$ Sp-Datalog($\neq$) whether it is negation bounded.*

Analogously to the classical result that the class of bounded programs is equally expressive as *UCQ*($\neg,\neq$) (the subset of *Sp-Datalog*($\neq$) programs in which the body of rules are constructed solely out of extensional relation names), we have the following syntactical characterisation:

▶ **Proposition 6.6.** *For every negation bounded program P in Sp-Datalog($\neq$) there is an equivalent program P' in Sp-Datalog($\neq$) that has a stratification P_1, P_2, with $P_1 \in$ Pos-Datalog($\neq$) and $P_2 \in$ UCQ($\neg,\neq$) (with no intensional relation name of P_2 occuring in P_1).*

Finally, we remark that the class of negation bounded programs can be extended a little, for example, by requiring a bound on the number of negated atoms only for relation symbols that occur positively in the program; or, by extension, for relation symbols that are not excluded from generating fringe conflicts due to some other syntactic reason. It is currently unclear whether a more fundamental generalization of Theorem 5.1 and Theorem 6.2 exists.

7 Stratified Datalog

A *stratified Datalog program* P is a set of rules as defined in Section 2.3, for which a stratification exists in a sequence of disjoint subprograms $P_1, \ldots, P_m$, with the following constraints: Every intensional relation name R in P occurs as a head in at most one subprogram P_i (we refer to P_i as the stratum in which R is defined).

- If an intensional relation name occurs positively in the body of a rule in subprogram P_i, then it is defined in a subprogram P_j, with $j \leq i$.
- If an intensional relation name occurs negated in the body of a rule in subprogram P_i, then it is defined in a subprogram P_j, with $j < i$.

We denote the class of stratified Datalog programs by *Str-Datalog*($\neq$). Since the subprograms P_i can be considered semi-positive, the semantics is defined as follows: $q_P(I) = q_{P_m} \circ q_{P_{m-1}} \circ \cdots \circ q_{P_1}(I)$. Here, we assume that for the subprograms P_i, with $i < m$ all relation names are output relation names, and for P_m only the distinguished output relations as defined by P. Similarly as before, we write **Str-Datalog**($\neq$) to denote the class of all queries expressible by programs in *Str-Datalog*($\neq$).

The following corollary is a straightforward consequence of Theorem 4.1.

▶ **Corollary 7.1.** *$\textbf{Str-Datalog}(\neq) \cap \mathcal{M} \not\subseteq \textbf{Pos-Datalog}(\neq)$.*

Despite Theorem 5.1, the gap between the monotone and positive fragment of stratified Datalog remains also without the interpreted inequality relation. The proof argument combines Theorem 4.1 with the observation that inequality is expressible through negation over intensional relation names.

▶ **Theorem 7.2.** *$\textbf{Str-Datalog} \cap \mathcal{M} \not\subseteq \textbf{Pos-Datalog}(\neq)$.*

8 A Best-Effort Approach to Negation Elimination

In this section, we consider the scenario in which an arbitrary *Str-Datalog*($\neq$) program is given and we are interested in finding an equivalent program with better computational properties (i.e., with less exposure to negation). Here, a program without negation is the ideal. Unfortunately, results like Proposition 3.9 do not help much to find such a program:

Firstly, the question if a given program in semi-positive Datalog is monotone is undecidable. Therefore, we cannot automatically infer whether a given program is in one of the desired subclasses.

▶ **Proposition 8.1.** *Testing whether a program in Sp-Datalog is monotone is undecidable.*

Secondly, these results only indicate whether an ideal equivalent rewriting (*i.e.*, a rewriting to a positive program) "certainly exists" or "may not exist". They do not help, especially in the latter case, to find such a program. Thirdly, even if no equivalent positive program exists, we may still be interested in finding an equivalent program with less exposure to negation.

The section proceeds as follows: In Section 8.1, we define a formal cost measure that allows to compare programs with negation. In Sections 8.2 and 8.3, we describe best-effort approaches towards improving the cost of a program as defined by this cost measure.

8.1 Cost Measure

We base our cost measure on observations from distributed Datalog evaluation (cf. Section 1): In an asynchronous distributed context (*e.g.*, [19, 16]), deciding the absence of a fact is significantly more difficult than deciding its presence, as it requires a round of consensus between the participating machines. One way to translate this observation into a formal cost measure is by hypothesising a correlation between the time it takes to derive an output fact for the first time and the minimal number of negated facts that its proof trees admit.

Notice that, in this hypothesis, positive programs are a conservative ideal (because proof trees of positive programs admit no negated facts), but programs that admit negated atoms are not necessarily considered worse (*i.e.*, if the rules with negated atoms are redundant).

For a formal definition, let P be an arbitrary program in *Sp-Datalog*($\neq$). We call a proof tree $\mathcal{T}$ from P *minimal* if no other proof tree $\mathcal{T}'$ from P exists that agrees with $\mathcal{T}$ on the label of its root, and has $\mathit{Fringe}^{+}_{\mathcal{T}'} \subseteq \mathit{Fringe}^{+}_{\mathcal{T}}$ and $\mathit{Fringe}^{-}_{\mathcal{T}'} \subseteq \mathit{Fringe}^{-}_{\mathcal{T}}$. Clearly, for every instance I and fact $\boldsymbol{f} \in q_P(I)$, we can always assume that a witnessing proof tree $\mathcal{T}$ exists that is minimal. Now, for two programs $P_1, P_2 \in$ *Sp-Datalog*($\neq$), we write $\mathit{cost}(P_1) \leq \mathit{cost}(P_2)$ if for every minimal proof tree $\mathcal{T}_1$ for P_1, with root an output fact, there is a proof tree $\mathcal{T}_2$ for P_2 that agrees with $\mathcal{T}_1$ on the label of its root, and with $\mathit{Fringe}^{+}_{\mathcal{T}_2} \subseteq \mathit{Fringe}^{+}_{\mathcal{T}_1}$ and $\mathit{Fringe}^{-}_{\mathcal{T}_2} \subseteq \mathit{Fringe}^{-}_{\mathcal{T}_1}$. We write $\mathit{cost}(P_1) < \mathit{cost}(P_2)$ if $\mathit{cost}(P_1) \leq \mathit{cost}(P_2)$ and for at least one such pair of proof trees $\mathcal{T}_1$ and $\mathcal{T}_2$, we have $\mathit{Fringe}^{-}_{\mathcal{T}_2} \subsetneq \mathit{Fringe}^{-}_{\mathcal{T}_1}$. We notice that our cost measure is defined over *Sp-Datalog*($\neq$) programs only, as we will use it to compare single-stratum fragments of *Str-Datalog*($\neq$) programs.

As can be expected, deciding properties about $cost(\cdot)$ quickly become undecidable.

▶ **Proposition 8.2.** *No algorithm exists that can decide for arbitrary equivalent programs $P_1, P_2 \in$ Sp-Datalog($\neq$) if $cost(P_1) < cost(P_2)$.*

8.2 Containment Testing

We start with an exploration of containment tests. Given a program $P \in$ *Sp-Datalog*($\neq$), we call a program $P' \in$ *Sp-Datalog*($\neq$) a *superior equivalent of* P if it is obtained by removing (some) negated atoms in rules from P. Clearly, P^+ is the extreme case, but now we are interested in programs P' that remain equivalent to the original program P. We note that $q_P \subseteq q_{P'}$ holds for every superior equivalent P' of P (the proof is a simple generalization of the proof argument for Proposition 3.2). Unfortunately, the other direction is undecidable.

▶ **Proposition 8.3.** *No procedure exists that decides $q_{P'} \subseteq q_P$ for arbitrary programs $P \in$ Sp-Datalog($\neq$) and superiorequivalent P' of P.*

To overcome this limitation, we test for $UCQ(\neg, \neq)$ containment instead (which is coNEXP-complete [13], respectively Π_2^p-complete [26, 20], if the arity of relations is bounded).

To formulate the next proposition, we need some additional notation: Given a program $P \in$ *Sp-Datalog*($\neq$), we denote by $\#(P)$ the program in $UCQ(\neg, \neq)$ obtained by adding a prime to all relation names occurring in the heads of rules (*i.e.*, $\mathsf{T}(\mathsf{x}, \mathsf{y}) \leftarrow \mathsf{E}(\mathsf{x}, \mathsf{z}), \mathsf{T}(\mathsf{z}, \mathsf{y})$ becomes $\mathsf{T}'(\mathsf{x}, \mathsf{y}) \leftarrow \mathsf{E}(\mathsf{x}, \mathsf{y}), \mathsf{T}(\mathsf{z}, \mathsf{y})$). Then, for $P_1, P_2 \in$ *Sp-Datalog*($\neq$) we test $q_{\#(P_1)} \subseteq q_{\#(P_2)}$ instead of $q_{P_1} \subseteq \#(P_2)$. Alternatively, one can consider uniform containment [24], which means that containment is tested for the queries described by P_1 and P_2, but taking as input schema the set of all extensional and intensional relation names of P_1 (resp, P_2) and as output schema the set of all intensional relation names of P_1 (resp, P_2). For the bounded arity case, a Π_2^p-completeness result is known due to Eiter and Fink [11]. The complexity for the non-bounded case appears to be open (*albeit* at least coNEXP-hard due to the earlier mentioned result for $UCQ(\neg, \neq)$ containment).

▶ **Proposition 8.4.** *For a program $P \in$ Sp-Datalog($\neq$) and superior equivalent P' of P, $\#(P') \subseteq \#(P)$ implies $q_{P'} = q_P$ and $cost(P') \leq cost(P)$.*

Let P be a stratified Datalog program. Then Proposition 8.4 admits a naive procedure, which we call NEG-ELIM, that applies to every stratum of P the following steps:

1. Test for every combination of negated atoms whether q_P is contained in $q_{P'}$, with P' the superior equivalent of P in which the selected atoms are removed.
2. Choose the superior equivalent of P that minimizes the total number of negated atoms, among those for which the test succeeds.

▶ **Theorem 8.5.** *Procedure NEG-ELIM runs with exponential space (respectively polynomial space, if a bound on the arity of considered relations is assumed).*

We remark that the special structure of $\#(P)$ and $\#(P')$ (*i.e.*, P' is a superior equivalent of P), does not admit a more efficient containment test.

▶ **Proposition 8.6.** *Testing for an arbitrary program $P \in$ Sp-Datalog($\neq$) and superior equivalent P' of P whether $q_{\#(P')} \subseteq q_{\#(P)}$ is coNEXP-hard (respectively Π_2^p-hard, if a bound on the arities of considered relations is assumed).*

8.3 Rule Expansions

One way to make the procedure from the previous section more powerful, is by doing containment tests for partially expanded program.

Let $P \in \mathit{Sp\text{-}Datalog}(\neq)$ and $\tau \in P$. By $\mathit{exp}(P)$ we denote the set of all 1-step expansions of rules in P. That is, the set of rules obtain from P by considering all possible replacements of intensional atoms in rules by the bodies of rules whose head matches the respective atom (after doing the necessary variable renaming).

▶ **Proposition 8.7.** *For every $P \in \mathit{Sp\text{-}Datalog}(\neq)$, $q_{\mathit{exp}(P)} = q_P$ and $\mathit{cost}(\mathit{exp}(P)) = \mathit{cost}(P)$.*

▶ **Example 8.8.** For an example illustrating the use of expansions in combination with algorithm NEG-ELIM, consider the two stratum program P, whose second stratum is program $P_\triangle$ from Example 2.1, and whose first stratum is a program P' over schema $\sigma' := \{\mathsf{Arc}^{(2)}, \mathsf{Edge}^{(2)}, \mathsf{Adom}^{(1)}\}$ with the next rules:

$$\begin{aligned} \mathsf{Adom}(\mathsf{x}) &\leftarrow \mathsf{Arc}(\mathsf{x},\mathsf{y}). \\ \mathsf{Adom}(\mathsf{x}) &\leftarrow \mathsf{Arc}(\mathsf{y},\mathsf{x}). \\ \mathsf{Edge}(\mathsf{x},\mathsf{y}) &\leftarrow \neg\mathsf{Arc}(\mathsf{x},\mathsf{y}), \mathsf{Adom}(\mathsf{x}), \mathsf{Adom}(\mathsf{y}). \end{aligned}$$

Now consider the expansion $\mathit{exp}(P_\triangle)$ of the second stratum:

$$\begin{aligned} &\mathsf{Output}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{z},\mathsf{x}). \\ &\mathsf{Output}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \neg\mathsf{Edge}(\mathsf{z},\mathsf{x}), \mathsf{Edge}(\mathsf{x},\mathsf{y}), \neg\mathsf{Edge}(\mathsf{y},\mathsf{w}), \mathsf{Edge}(\mathsf{w},\mathsf{x}), \mathsf{x} \neq \mathsf{y}. \end{aligned}$$

While directly applying algorithm NEG-ELIM over $P_\triangle$ does not improve the program, an application over $\mathit{exp}(P_\triangle)$ finds a negation-free equivalent:

$$\begin{aligned} &\mathsf{Output}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{z},\mathsf{x}). \\ &\mathsf{Output}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{y},\mathsf{z}), \mathsf{Edge}(\mathsf{x},\mathsf{y}), \mathsf{Edge}(\mathsf{w},\mathsf{x}), \mathsf{x} \neq \mathsf{y}. \end{aligned}$$

Hence, while P is not monotone, and therefore has no equivalent in $\mathit{Pos\text{-}Datalog}(\neq)$, we do obtain an equivalent single-stratum program $P_1 \cup P_2'$.

9 Conclusion

Motivated by applications in network programming, we studied fundamental questions about the relationship between the monotone and positive fragments of several variants of Datalog with negation (an overview is given by Figure 1). We also showed how the amount of negation that such programs admit can be decreased independently of whether they are monotone.

Related to monotonicity is the concept preservation under extensions ($\mathcal{E}$). While it is known that $\mathit{Sp\text{-}Datalog}(\neq) \subseteq \mathcal{E}$ [3], to the best of our knowledge, it is still an open question whether $\mathit{Str\text{-}Datalog}(\neq) \cap \mathcal{E} \overset{?}{=} \mathit{Sp\text{-}Datalog}(\neq)$. The latter question is of particular interest, because $\mathcal{E}$ is another notion that is associated with coordination in distributed systems [7].

The techniques that we discuss in Section 8, to remove negation from stratified Datalog programs, are by no means exhaustive. We also do not provide formal guarantees on the effectiveness of the approach. An interesting question therefore is whether other decidable techniques exist that can be of use for this purpose. Additionally, it would be interesting to perform an experimental study to see if these techniques can be combined into an effective procedure that is of use for real-live programs. The techniques that we present in Section 8 aim for a best-effort approach to automatically reduce the need for consensus in a program, which in the context of Datalog, translates to the elimination of (stratified) negation. While it is obvious that such a procedure cannot beat optimization by hand, we see it useful in complex systems, that a program may be composed out of multiple programs and views. Then, optimization of individual subprograms does not necessarily imply optimization of the program as a whole, and a best-effort approach may be the only way to achieve improvement.

References

1 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Émilien Antoine. A rule-based language for web data management. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2011)*, pages 293–304, 2011. doi:10.1145/1989284.1989320.

2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

3 Foto N. Afrati, Stavros S. Cosmadakis, and Mihalis Yannakakis. On Datalog vs. Polynomial Time. *Journal of Computer and System Sciences*, 51(2):177–196, 1995. doi:10.1006/jcss.1995.1060.

4 Miklós Ajtai and Yuri Gurevich. Monotone versus positive. *Journal of the ACM*, 34(4):1004–1015, 1987. doi:10.1145/31846.31852.

5 Miklós Ajtai and Yuri Gurevich. Datalog vs First-Order Logic. *J. Comput. Syst. Sci.*, 49(3):562–588, 1994. doi:10.1016/S0022-0000(05)80071-6.

6 Tom J. Ameloot, Jan Van den Bussche, William R. Marczak, Peter Alvaro, and Joseph M. Hellerstein. Putting logic-based distributed systems on stable grounds. *Theory and Practice of Logic Programming*, 16(4):378–417, 2016. doi:10.1017/S1471068415000381.

7 Tom J. Ameloot, Bas Ketsman, Frank Neven, and Daniel Zinn. Weaker Forms of Monotonicity for Declarative Networking: A More Fine-Grained Answer to the CALM-Conjecture. *ACM Transactions on Database Systems*, 40(4):21:1–21:45, 2016. doi:10.1145/2809784.

8 Tom J. Ameloot, Bas Ketsman, Frank Neven, and Daniel Zinn. Datalog Queries Distributing over Components. *ACM Transactions on Computational Logic*, 18(1):5:1–5:35, 2017. doi:10.1145/3022743.

9 Surajit Chaudhuri and Moshe Y. Vardi. On the Equivalence of Recursive and Nonrecursive Datalog Programs. In *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1992)*, pages 55–66, 1992.

10 Anuj Dawar and Stephan Kreutzer. On Datalog vs. LFP. In *Automata, Languages and Programming, 35th International Colloquium, (ICALP 2008)*, pages 160–171, 2008. doi:10.1007/978-3-540-70583-3_14.

11 Thomas Eiter and Michael Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In *Logic Programming, 19th International Conference (ICLP 2003)*, pages 224–238, 2003. doi:10.1007/978-3-540-24599-5_16.

12 Tomás Feder and Moshe Y. Vardi. Homomorphism Closed vs. Existential Positive. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 311–320, 2003. doi:10.1109/LICS.2003.1210071.

13 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Containment for Conjunctive Queries with Union and Negation. In *19th International Conference on Database Theory (ICDT 2016)*, pages 9:1–9:17, 2016. doi:10.4230/LIPIcs.ICDT.2016.9.

14 Jim N. Gray. Notes on data base operating systems. *Lecture Notes in Computer Science*, 60:393–481, 1978.

15 Rachid Guerraoui and Jingjing Wang. How Fast can a Distributed Transaction Commit? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, (PODS'17)*, pages 107–122, 2017. doi:10.1145/3034786.3034799.

16 Joseph M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010. doi:10.1145/1860702.1860704.

17 Bas Ketsman, Aws Albarghouthi, and Paraschos Koutris. Distribution Policies for Datalog. In *21st International Conference on Database Theory (ICDT 2018)*, pages 17:1–17:22, 2018. doi:10.4230/LIPIcs.ICDT.2018.17.

18 Phokion G. Kolaitis and Moshe Y. Vardi. On the Expressive Power of Datalog: Tools and a Case Study. *J. Comput. Syst. Sci.*, 51(1):110–134, 1995. doi:10.1006/jcss.1995.1055.

19 Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: language, execution and optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006)*, pages 97–108, 2006. doi:10.1145/1142473.1142485.

20 Marie-Laure Mugnier, Geneviève Simonet, and Michaël Thomazo. On the complexity of entailment in existential conjunctive first-order logic with atomic negation. *Information and Computation*, 215:8–31, 2012. doi:10.1016/j.ic.2012.03.001.

21 Aleksandr A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Doklady Akademii Nauk SSSR*, 281:798–801, 1985.

22 Benjamin Rossman. Homomorphism preservation theorems. *Journal of the ACM*, 55(3):15:1–15:53, 2008. doi:10.1145/1379759.1379763.

23 Sebastian Rudolph and Michaël Thomazo. Expressivity of Datalog Variants - Completing the Picture. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 1230–1236, 2016. URL: https://hal.inria.fr/hal-01302832.

24 Yehoshua Sagiv. Optimizing Datalog Programs. In *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1987)*, pages 349–362, 1987. doi:10.1145/28659.28696.

25 Alexei P. Stolboushkin. Finitely Monotone Properties. In *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science*, pages 324–330, 1995. doi:10.1109/LICS.1995.523267.

26 Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000. doi:10.1016/S0304-3975(99)00219-4.

27 Ouri Wolfson and Abraham Silberschatz. Distributed Processing of Logic Programs. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD 1988)*, pages 329–336, 1988. doi:10.1145/50202.50242.

28 Daniel Zinn, Todd J. Green, and Bertram Ludäscher. Win-Move is Coordination-Free (Sometimes). *CoRR*, abs/1312.2919, 2013. arXiv:1312.2919.

The Shapley Value of Tuples in Query Answering

Ester Livshits
Technion, Haifa, Israel
esterliv@cs.technion.ac.il

Leopoldo Bertossi[1]
Univ. Adolfo Ibañez, Santiago, Chile
RelationalAI Inc., Toronto, Canada
leopoldo.bertossi@uai.cl

Benny Kimelfeld
Technion, Haifa, Israel
bennyk@cs.technion.ac.il

Moshe Sebag
Technion, Haifa, Israel
moshesebag@campus.technion.ac.il

Abstract

We investigate the application of the Shapley value to quantifying the contribution of a tuple to a query answer. The Shapley value is a widely known numerical measure in cooperative game theory and in many applications of game theory for assessing the contribution of a player to a coalition game. It has been established already in the 1950s, and is theoretically justified by being the very single wealth-distribution measure that satisfies some natural axioms. While this value has been investigated in several areas, it received little attention in data management. We study this measure in the context of conjunctive and aggregate queries by defining corresponding coalition games. We provide algorithmic and complexity-theoretic results on the computation of Shapley-based contributions to query answers; and for the hard cases we present approximation algorithms.

2012 ACM Subject Classification Theory of computation → Data provenance

Keywords and phrases Shapley value, query answering, conjunctive queries, aggregate queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.20

Related Version A full version of the paper is available at `https://arxiv.org/abs/1904.08679`.

Funding The work of Ester Livshits, Benny Kimelfeld and Moshe Sebag was supported by the Israel Science Foundation (ISF), grants 1295/15 and 768/19, and the Deutsche Forschungsgemeinschaft (DFG) project 412400621 (DIP program). The work of Ester Livshits was also supported by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel Cyber Bureau.

1 Introduction

The Shapley value is named after Lloyd Shapley who introduced the value in a seminal 1952 article [33]. He considered a *cooperative game* that is played by a set A of players and is defined by a *wealth function* v that assigns, to each coalition $S \subseteq A$, the wealth $v(S)$. For instance, in our running example the players are researchers, and $v(S)$ is the total number of citations of papers with an author in S. As another example, A might be a set of politicians, and $v(S)$ the number of votes that a poll assigns to the party that consists of the candidates in S. The question is how to distribute the wealth $v(A)$ among the players, or from a different perspective, how to quantify the contribution of each player to the overall wealth.

[1] Member of Millenium Institute on Foundations of Data (IMFD), Chile.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 20; pp. 20:1–20:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For example, the removal of a researcher r may have zero impact on the overall number of citations, since each paper has co-authors from A. Does it mean that r has no contribution at all? What if the removal in turns of *every* individual author has no impact? Shapley considered distribution functions that satisfy a few axioms of good behavior. Intuitively, the axioms state that the function should be invariant under isomorphism, the sum over all players should be equal to the total wealth, and the contribution to a sum of wealths is equal to the sum of separate contributions. Quite remarkably, Shapley has established that there is a *single* such function, and this function has become known as the *Shapley value*.

The Shapley value is informally defined as follows. Assume that we select players one by one, randomly and without replacement, starting with the empty set. Whenever we select the player p, its addition to the set S of players selected so far may cause a change in wealth from $v(S)$ to $v(S \cup \{p\})$. The Shapley value of p is the expectation of change that p causes in this probabilistic process.

The Shapley value has been applied in various areas and fields beyond cooperative game theory (e.g., [1, 2]), such as bargaining foundations in economics [14], takeover corporate rights in law [27], pollution responsibility in environmental management [20, 29], influence measurement in social network analysis [26], and utilization of multiple Internet Service Providers (ISPs) in networks [22]. Closest to database manegement is the application of the Shapley value to attributing a level of *inconsistency* to a statement in an inconsistent knowledge base [17, 36]; the idea is natural: as wealth, adopt a measure of inconsistency for a set of logical sentences [12], and then associate to each sentence its Shapley value.

In this paper, we apply the Shapley value to quantifying the contribution of database facts (tuples) to query results. As in previous work on quantification of contribution of facts [24, 31], we view the database as consisting of two types of facts: *endogenous* facts and *exogenous* facts. Exogenous facts are taken as given (e.g., inherited from external sources) without questioning, and are beyond experimentation with hypothetical or counterfactual scenarios. On the other hand, we may have control over the endogenous facts, and these are the facts for which we reason about existence and marginal contribution. Our focus is on queries that can be viewed as mapping databases to numbers. These include Boolean queries (mapping databases to zero and one) and aggregate queries (e.g., count the number of tuples in a multiway join). As a cooperative game, the endogenous facts take the role of the players, and the result of the query is the wealth. The core computational problem for a query is then: given a database and an endogenous fact, compute the Shapley value of the fact.

We study the complexity of computing the Shapley value for Conjunctive Queries (CQs) and aggregate functions over CQs. Our main results are as follows. We first establish a dichotomy in data complexity for the class of Boolean CQs without self-joins. Interestingly, our dichotomy is the same as that of query inference in tuple-independent probabilistic databases [9]: if the CQ is hierarchical, then the problem is solvable in polynomial time, and otherwise, it is $\mathrm{FP}^{\#\mathrm{P}}$-complete (i.e., complete for the intractable class of polynomial-time algorithms with an oracle to, e.g., a counter of the satisfying assignments of a propositional formula). The proof, however, is more challenging than that of Dalvi and Suciu [9], as the Shapley value involves coefficients that do not seem to easily factor out. Since the Shapley value is a probabilistic expectation, we show how to use the linearity of expectation to extend the dichotomy to arbitrary summations over CQs without self-joins. For non-hierarchical queries (and, in fact, all unions of CQs), we show that both Boolean and summation versions are efficiently approximable (i.e., have a multiplicative FPRAS) via Monte Carlo sampling.

The general conclusion is that computing the exact Shapley value is notoriously hard, but the picture is optimistic if approximation is allowed under strong guarantees of error boundedness. Our results immediately generalize to non-Boolean CQs and group-by operators, where the goal is to compute the Shapley value of a fact to each tuple in the answer of a query. For aggregate functions other than summation (where we cannot apply the linearity of expectation), the picture is far less complete, and remains for future investigation. Nevertheless, we give some positive preliminary results about special cases of the minimum and maximum aggregate functions.

Various formal measures have been proposed for quantifying the contribution of a fact f to a query answer. Meliou et al. [24] adopted the quantity of *responsibility* that is inversely proportional to the minimal number of endogenous facts that should be removed to make f counterfactual (i.e., removing f transitions the answer from true to false). This measure adopts earlier notions of formal causality by Halpern and Pearl [16]. This measure, however, is fundamentally designed for non-numerical queries, and it is not at all clear whether it can incorporate the numerical contribution of a fact (e.g., recognizing that some facts contribute more than others due to high numerical attributes). Salimi et al. [31] proposed the *causal effect*: assuming endogenous facts are randomly removed independently and uniformly, what is the difference in the expected query answer between assuming the presence and the absence of f? Interestingly, as we show here, this value is the same as the *Banzhaf power index* that has also been studied in the context of wealth distribution in cooperative games [11], and is different from the Shapley value [30, Chapter 5]. While the justification to measuring fact contribution using one measure over the other is yet to be established, we believe that the suitability of the Shapley value is backed by the aforementioned theoretical justification as well as its massive adoption in a plethora of fields. In addition, the complexity of measuring the causal effect has been left open, and we conjecture that all of our complexity results are applicable to (and, in fact, simpler to prove in) the causal-effect framework.

The remainder of the paper is organized as follows. In the next section, we give preliminary concepts, definitions and notation. In Section 3, we present the Shapley value to measure the contribution of a fact to a query answer, along with illustrating examples. In Section 4, we study the complexity of calculating the Shapley value. Finally, we discuss past contribution measures in Section 5 and conclude in Section 6. For lack of space, missing proofs are given in the extended version of the paper [21].

2 Preliminaries

Databases. A (relational) *schema* $\mathbf{S}$ is a collection of *relation symbols* with each relation symbol R in $\mathbf{S}$ having an associated arity that we denote by $ar(R)$. We assume a countably infinite set Const of *constants* that are used as database values. If $\vec{c} = (c_1, \dots, c_k)$ is a tuple of constants and $i \in \{1, \dots, k\}$, then we use $\vec{c}[i]$ to refer to the constant c_i. A *relation* r is a set of tuples of constants, each having the same arity (length) that we denote by $ar(r)$. A *database* D (over the schema $\mathbf{S}$) associates with each relation symbol R a finite relation r, which we denote by R^D, such that $ar(R) = ar(R^D)$. We denote by $\mathrm{DB}(\mathbf{S})$ the set of all databases over the schema $\mathbf{S}$. Notationally, we identify a database D with its finite set of *facts* $R(c_1, \dots, c_k)$, stating that the relation R^D over the k-ary relation symbol R contains the tuple $(c_1, \dots, c_k) \in \mathsf{Const}^k$. In particular, two databases D and D' over $\mathbf{S}$ satisfy $D \subseteq D'$ if and only if $R^D \subseteq R^{D'}$ for all relation symbols R of $\mathbf{S}$.

Following prior work on explanations and responsibility of facts to query answers [23, 25], we view the database as consisting of two types of facts: *exogenous* facts and *endogenous* facts. Exogenous facts represent a context of information that is taken for granted and assumed not to claim any contribution or responsibility to the result of a query. Our concern is about *the role of the endogenous facts* in establishing the result of the query. In notation, we denote by D_{x} and D_{n} the subsets of D that consist of the exogenous and endogenous facts, respectively. Hence, in our notation we have that $D = D_{\mathsf{x}} \cup D_{\mathsf{n}}$.

AUTHOR (endo)

	name	*affil*
f_1^{a}	`Alice`	`UCLA`
f_2^{a}	`Bob`	`NYU`
f_3^{a}	`Cathy`	`UCSD`
f_4^{a}	`David`	`MIT`
f_5^{a}	`Ellen`	`UCSD`

INST (exo)

	name	*state*
f_1^{i}	`UCLA`	`CA`
f_2^{i}	`UCSD`	`CA`
f_3^{i}	`NYU`	`NY`
f_4^{i}	`MIT`	`MA`

PUB (exo)

	author	*pub*
f_1^{p}	`Alice`	`A`
f_2^{p}	`Alice`	`B`
f_3^{p}	`Bob`	`C`
f_4^{p}	`Cathy`	`C`
f_5^{p}	`Cathy`	`D`
f_6^{p}	`David`	`C`

CITATIONS (exo)

	paper	*cits*
f_1^{c}	`A`	`18`
f_2^{c}	`B`	`2`
f_2^{c}	`C`	`8`
f_3^{c}	`D`	`12`

Figure 1 The database of the running example.

▶ **Example 1.** Figure 1 depicts the database D of our running example from the domain of academic publications. The relation AUTHOR stores authors along with their affiliations, which are stored with their states in INST. The relation PUB associates authors with their publications, and CITATIONS stores the number of citations for each paper. For example, publication `C` has `8` citations and it is written jointly by `Bob` from `NYU` of `NY` state, `Cathy` from `UCSD` of `CA` state, and `David` from `MIT` of `MA` state. All AUTHOR facts are endogenous, and all remaining facts are exogenous. Hence, $D_{\mathsf{n}} = \{f_1^{\mathrm{a}}, f_2^{\mathrm{a}}, f_3^{\mathrm{a}}, f_4^{\mathrm{a}}, f_5^{\mathrm{a}}\}$ and D_{x} consists of all f_j^x for $x \in \{\mathrm{i}, \mathrm{p}, \mathrm{c}\}$ and relevant j. ◀

Relational and conjunctive queries. Let $\mathbf{S}$ be a schema. A *relational query* is a function that maps databases to relations. More formally, a relational query q of arity k is a function $q : \mathrm{DB}(\mathbf{S}) \to \mathsf{Const}^k$ that maps every database over $\mathbf{S}$ to a finite relation $q(D)$ of arity k. We denote the arity of q by $ar(q)$. Each tuple $\vec{c}$ in $q(D)$ is an *answer* to q on D. If the arity of q is zero, then we say that q is a *Boolean* query; in this case, $D \models q$ denotes that $q(D)$ consists of the empty tuple $()$, while $D \not\models q$ denotes that $q(D)$ is empty.

Our analysis will focus on the special case of *Conjunctive Queries* (CQs). A CQ over the schema $\mathbf{S}$ is a relational query definable by a first-order formula of the form $\exists y_1 \cdots \exists y_m \theta(\vec{x}, y_1, \dots, y_m)$, where θ is a conjunction of atomic formulas of the form $R(\vec{t})$ with variables among those in $\vec{x}, y_1, \dots, y_m$. In the remainder of the paper, a CQ q will be written shortly as a logic rule, that is, an expression of the form

$$q(\vec{x}) \text{ :- } R_1(\vec{t}_1), \dots, R_n(\vec{t}_n)$$

where each R_i is a relation symbol of $\mathbf{S}$, each $\vec{t}_i$ is a tuple of variables and constants with the same arity as R_i, and $\vec{x}$ is a tuple of k variables from $\vec{t}_1, \dots, \vec{t}_n$. We call $q(\vec{x})$ the *head* of q, and $R_1(\vec{t}_1), \dots, R_n(\vec{t}_n)$ the *body* of q. Each $R_i(\vec{t}_i)$ is an *atom* of q. The variables occurring in the head are called the *head variables*, and we make the standard safety assumption that every head variable occurs at least once in the body. The variables occurring in the body but not in the head are existentially quantified, and are called the *existential variables*. The answers to q on a database D are the tuples $\vec{c}$ that are obtained by projecting to $\vec{x}$ all homomorphisms from q to D, and replacing each variable with the constant it is mapped to. A homomorphism from q to D is a mapping of the variables in q to the constants of D, such that every atom in q is mapped to a fact in D.

A *self-join* in a CQ q is a pair of distinct atoms over the same relation symbol. For example, in the query $q() \text{ :- } R(x, y), S(x), R(y, z)$, the first and third atoms constitute a self-join. We say that q is *self-join-free* if it has no self-joins, or in other words, every relation symbol occurs at most once in the body.

Let q be a CQ. For a variable y of q, let A_y be the set of atoms $R_i(\vec{t_i})$ of q that contain y (that is, y occurs in $\vec{t_i}$). We say that q is *hierarchical* if for all existential variables y and y' it holds that $A_y \subseteq A_{y'}$, or $A_{y'} \subseteq A_y$, or $A_y \cap A_{y'} = \emptyset$ [8]. For example, every CQ with at most two atoms is hierarchical. The smallest non-hierarchical CQ is the following.

$$\mathsf{q}_{\mathsf{RST}}() \text{ :- } R(x), S(x,y), T(y) \tag{1}$$

On the other hand, the query $q(x)$:- $R(x), S(x,y), T(y)$, which has a single existential variable, is hierarchical.

Let q be a Boolean query and D a database, both over the same schema, and let $f \in D_{\mathsf{n}}$ be an endogenous fact. We say that f is a *counterfactual cause* (*for* q *w.r.t.* D) [23, 24] if the removal of f causes q to become false; that is, $D \models q$ and $D \setminus \{f\} \not\models q$.

▶ **Example 2.** We will use the following queries in our examples.

$$\begin{aligned}
q_1() &\text{ :- } \textsc{Author}(x,y), \textsc{Pub}(x,z)\\
q_2() &\text{ :- } \textsc{Author}(x,y), \textsc{Pub}(x,z), \textsc{Citations}(z,w)\\
q_3(z,w) &\text{ :- } \textsc{Author}(x,y), \textsc{Pub}(x,z), \textsc{Citations}(z,w)\\
q_4(z,w) &\text{ :- } \textsc{Author}(x,y), \textsc{Pub}(x,z), \textsc{Citations}(z,w), \textsc{Inst}(y,\texttt{CA})
\end{aligned}$$

Note that q_1 and q_2 are Boolean, whereas q_3 and q_4 are not. Also note that q_1 and q_3 are hierarchical, and q_2 and q_4 are not. Considering the database D of Figure 1, none of the Author facts is a counterfactual cause for q_1, since the query remains true even if the fact is removed. The same applies to q_2. However, the fact f_1^{a} is a counterfactual cause for the Boolean CQ $q_1'()$:- $\textsc{Author}(x,\texttt{UCLA}), \textsc{Pub}(x,z)$, asking whether there is a publication with an author from UCLA, since D satisfies q_1' but the removal of Alice causes q_1' to be violated by D, as no other author from UCLA exists. ◀

Numerical and aggregate-relational queries. A *numerical query* α is a function that maps databases to numbers. More formally, a numerical query α is a function $\alpha : \mathrm{DB}(\mathbf{S}) \to \mathbb{R}$ that maps every database D over $\mathbf{S}$ to a real number $\alpha(D)$.

A special form of a numerical query α is what we refer to as an *aggregate-relational query*: a k-ary relational query q followed by an aggregate function $\gamma : \mathcal{P}(\mathsf{Const}^k) \to \mathbb{R}$ (where $\mathcal{P}(\mathsf{Const}^k)$ is the power set of Const^k that consists of all subsets of Const^k) that maps the resulting relation $q(D)$ into a single number $\gamma(q(D))$. We denote this aggregate-relational query as $\gamma[q]$; hence, $\gamma[q](D) \overset{\text{def}}{=} \gamma(q(D))$.

Special cases of aggregate-relational queries include the functions of the form $\gamma = F\langle\varphi\rangle$ that transform every tuple $\vec{c}$ into a number $\varphi(\vec{c})$ via a *feature function* $\varphi : \mathsf{Const}^k \to \mathbb{R}$, and then contract the resulting bag of numbers into a single number. Formally, we define $F\langle\varphi\rangle[q](D) \overset{\text{def}}{=} F(\{\!\{\varphi(\vec{c}) \mid \vec{c} \in q(D)\}\!\})$ where $\{\!\{\cdot\}\!\}$ is used for bag notation. For example, if we assume that the ith attribute of $q(D)$ takes a numerical value, then φ can simply copy this number (i.e., $\varphi(\vec{c}) = \vec{c}[i]$); we denote this φ by $[i]$. As another example, φ can be the product of two attributes: $\varphi = [i] \cdot [j]$. We later refer to the following aggregate-relational queries.

$$\mathsf{sum}\langle\varphi\rangle[q](D) \overset{\text{def}}{=} \sum_{\vec{c} \in q(D)} \varphi(\vec{c})$$

$$\mathsf{max}\langle\varphi\rangle[q](D) \overset{\text{def}}{=} \begin{cases} \max\{\varphi(\vec{c}) \mid \vec{c} \in q(D)\} & \text{if } q(D) \neq \emptyset;\\ 0 & \text{if } q(D) = \emptyset. \end{cases}$$

Other popular examples include the minimum (defined analogously to maximum), average and median over the feature values. A special case of $\mathsf{sum}\langle\varphi\rangle[q]$ is $\mathsf{count}[q]$ that counts the number of answers for q. That is, $\mathsf{count}[q]$ is $\mathsf{sum}\langle\mathbf{1}\rangle[q]$, where "**1**" is the feature function that maps every k-tuple to the number 1. A special case of $\mathsf{count}[q]$ is when q is Boolean; in this case, we may abuse the notation and identify $\mathsf{count}[q]$ with q itself. Put differently, we view q as the numerical query α defined by $\alpha(D) = 1$ if $D \models q$ and $\alpha(D) = 0$ if $D \not\models q$.

▶ **Example 3.** Following are examples of aggregate-relational queries over the relational queries of Example 2.

- $\alpha_1 \overset{\text{def}}{=} \mathsf{sum}\langle[2]\rangle[q_3]$ calculates the total number of citations of all published papers.
- $\alpha_2 \overset{\text{def}}{=} \mathsf{count}[q_3]$ counts the papers in CITATIONS with an author in the database.
- $\alpha_3 \overset{\text{def}}{=} \mathsf{sum}\langle[2]\rangle[q_4]$ calculates the total number of citations of papers by Californians.
- $\alpha_4 \overset{\text{def}}{=} \mathsf{max}\langle[2]\rangle[q_3]$ calculates the number of citations for the most cited paper.

For D of Figure 1 we have $\alpha_1(D) = \mathtt{40}$, $\alpha_2(D) = \mathtt{4}$, $\alpha_3(D) = \mathtt{40}$ and $\alpha_4(D) = \mathtt{18}$. ◀

In terms of presentation, when we mention general functions γ and φ, we make the implicit assumption that they are computable in polynomial time with respect to the representation of their input. Also, observe that our modeling of an aggregate-relational query does not allow for *grouping*, since a database is mapped to a single number. This is done for simplicity of presentation, and all concepts and results of this paper generalize to grouping as in traditional modeling (e.g., [6]). This is explained in the next section.

Shapley value. Let A be a finite set of *players*. A *cooperative game* is a function $v : \mathcal{P}(A) \to \mathbb{R}$, such that $v(\emptyset) = 0$. The value $v(S)$ represents a value, such as wealth, jointly obtained by S when the players of S cooperate. The *Shapley value* [33] measures the share of each individual player $a \in A$ in the gain of A for the cooperative game v. Intuitively, the gain of a is as follows. Suppose that we form a team by taking the players one by one, randomly and uniformly without replacement; while doing so, we record the change of v due to the addition of a as the random contribution of a. Then the Shapley value of a is the expectation of the random contribution.

$$\mathrm{Shapley}(A, v, a) \overset{\text{def}}{=} \frac{1}{|A|!} \sum_{\sigma \in \Pi_A} \big(v(\sigma_a \cup \{a\}) - v(\sigma_a)\big) \tag{2}$$

where Π_A is the set of all possible permutations over the players in A, and for each permutation σ we denote by σ_a the set of players that appear before a in the permutation.

An alternative formula for the Shapley value is the following.

$$\mathrm{Shapley}(A, v, a) \overset{\text{def}}{=} \sum_{B \subseteq A \setminus \{a\}} \frac{|B|! \cdot (|A| - |B| - 1)!}{|A|!} \Big(v(B \cup \{a\}) - v(B)\Big) \tag{3}$$

Note that $|B|! \cdot (|A| - |B| - 1)!$ is the number of permutations over A such that all players in B come first, then a, and then all remaining players. For further reading, we refer the reader to the book by Roth [30].

3 Shapley Value of Database Facts

Let α be a numerical query over a schema $\mathbf{S}$, and let D be a database over $\mathbf{S}$. We wish to quantify the contribution of every endogenous fact to the result $\alpha(D)$. For that, we view α as a cooperative game over D_{n}, where the value of every subset E of D_{n} is $\alpha(E \cup D_{\mathsf{x}})$.

▶ **Definition 4** (Shapley Value of Facts). *Let* $\mathbf{S}$ *be a schema,* α *a numerical query,* D *a database, and* f *an endogenous fact of* D*. The* Shapley value *of* f *for* α*, denoted* $\mathrm{Shapley}(D, \alpha, f)$*, is the value* $\mathrm{Shapley}(A, v, a)$ *as given in* (2)*, where:*

- $A = D_{\mathsf{n}}$;
- $v(E) = \alpha(E \cup D_{\mathsf{x}}) - \alpha(D_{\mathsf{x}})$ *for all* $E \subseteq A$;
- $a = f$.

That is, $\mathrm{Shapley}(D, \alpha, f)$ *is the Shapley value of* f *in the cooperative game that has the endogenous facts as the set of players and values each team by the quantity it adds to* α*.*

As a special case, if q is a Boolean query, then $\mathrm{Shapley}(D, q, f)$ is the same as the value $\mathrm{Shapley}(D, \mathsf{count}[q], f)$. In this case, the corresponding cooperative game takes the values 0 and 1, and the Shapley value then coincides with the *Shapley-Shubik index* [32]. Some fundamental properties of the Shapley value [33] are reflected here as follows:

- $\mathrm{Shapley}(D, a \cdot \alpha + b \cdot \beta, f) = a \cdot \mathrm{Shapley}(D, \alpha, f) + b \cdot \mathrm{Shapley}(D, \beta, f)$.
- $\alpha(D) = \alpha(D_{\mathsf{x}}) + \sum_{f \in D_{\mathsf{n}}} \mathrm{Shapley}(D, \alpha, f)$.

▶ Remark 5. Note that $\mathrm{Shapley}(D, \alpha, f)$ is defined for a general numerical query α. The definition is immediately extendible to queries with *grouping* (producing tuples of database constants and numbers [6]), where we would measure the responsibility of f for an answer tuple $\vec{a}$ and write something like $\mathrm{Shapley}(D, \alpha, \vec{a}, f)$. In that case, we treat every group as a separate numerical query. We believe that focusing on numerical queries (without grouping) allows us to keep the presentation considerably simpler while, at the same time, retaining the fundamental challenges. ◀

In the remainder of this section, we illustrate the Shapley value on our running example.

▶ **Example 6.** We begin with a Boolean CQ, and specifically q_1 from Example 2. Recall that the endogenous facts correspond to the authors. As Ellen has no publications, her addition to any $D_{\mathsf{x}} \cup E$ where $E \subseteq D_{\mathsf{n}}$ does not change the satisfaction of q_1. Hence, its Shapley value is zero: $\mathrm{Shapley}(D, q_1, f_5^{\mathrm{a}}) = 0$. The fact f_1^{a} changes the query result if it is either the first fact in the permutation, or it is the second fact after f_5^{a}. There are 4! permutations that satisfy the first condition, and 3! permutations that satisfy the second. The contribution of f_1^{a} to the query result is one in each of these permutations, and zero otherwise. Therefore, we have $\mathrm{Shapley}(D, q_1, f_1^{\mathrm{a}}) = \frac{4!+3!}{120} = \frac{1}{4}$. The same argument applies to f_2^{a}, f_3^{a} and f_4^{a}, and so, $\mathrm{Shapley}(D, q_1, f_2^{\mathrm{a}}) = \mathrm{Shapley}(D, q_1, f_3^{\mathrm{a}}) = \mathrm{Shapley}(D, q_1, f_4^{\mathrm{a}}) = \frac{1}{4}$. We get the same numbers for q_2, since every paper is mentioned in the CITATIONS relation. Note that the value of the query q_1 on the database is 1, and it holds that $\sum_{i=1}^{5} \mathrm{Shapley}(D, q_1, f_i^{\mathrm{a}}) = 4 \cdot \frac{1}{4} + 0 = 1$; hence, the second fundamental property of the Shapley value mentioned above is satisfied.

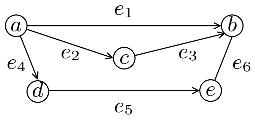
While Alice, Bob, Cathy and David have the same Shapley value for q_1, things change if we consider the relation PUB endogenous as well: the Shapley value of Alice and Cathy will be higher than Bob's and David's values, since they have more publications. Specifically, the fact f_1^{a}, for example, will change the query result if and only if at least one of f_1^{p} or f_2^{p} appears earlier in the permutation, and no pair among $\{f_2^{\mathrm{a}}, f_3^{\mathrm{p}}\}$, $\{f_3^{\mathrm{a}}, f_3^{\mathrm{p}}\}$, $\{f_3^{\mathrm{a}}, f_4^{\mathrm{p}}\}$, and $\{f_4^{\mathrm{a}}, f_3^{\mathrm{p}}\}$ appears earlier than f_1^{a}. By rigorous counting, we can show that there are: 2 such sets of size one, 17 such sets of size two, 56 such sets of size three, 90 such sets of size four, 73 such sets of size five, 28 such sets of size six, and 4 such sets of size seven. Therefore, the Shapley value of f_1^{a} is:

$$\begin{aligned}\mathrm{Shapley}(D, q_1, f_1^{\mathrm{a}}) = 2 \cdot \frac{(11-2)!1!}{11!} + 17 \cdot \frac{(11-3)!2!}{11!} + 56 \cdot \frac{(11-4)!3!}{11!} + 90 \cdot \frac{(11-5)!4!}{11!} \\ + 73 \cdot \frac{(11-6)!5!}{11!} + 28 \cdot \frac{(11-7)!6!}{11!} + 4 \cdot \frac{(11-8)!7!}{11!} = \frac{442}{2520}\end{aligned}$$

We can similarly compute the Shapley value for the rest of the authors, concluding that $\text{Shapley}(D, q_1, f_2^{\text{a}}) = \text{Shapley}(D, q_1, f_4^{\text{a}}) = \frac{241}{2520}$ and $\text{Shapley}(D, q_1, f_3^{\text{a}}) = \frac{442}{2520}$. Hence, the Shapley value is the same for Alice and Cathy, who have two publications each, and lower for Bob and David, that have only one publication. ◀

The following example, taken from Salimi et al. [31], illustrates the Shapley value on (Boolean) graph reachability.

▶ **Example 7.** Consider the following database G defined via the relation symbol EDGE/2.



Here, we assume that all edges e_i are endogenous facts. Let p_{ab} be the Boolean query (definable in, e.g., Datalog) that determines whether there is a path from a to b. Let us calculate $\text{Shapley}(G, p_{ab}, e_i)$ for different edges e_i. Intuitively, we expect e_1 to have the highest value since it provides a direct path from a to b, while e_2 contributes to a path only in the presence of e_3, and e_4 enables a path only in the presence of both e_5 and e_6. We show that, indeed, it holds that $\text{Shapley}(G, p_{ab}, e_1) > \text{Shapley}(G, p_{ab}, e_2) > \text{Shapley}(G, p_{ab}, e_4)$.

To illustrate the calculation, observe that there are 2^5 subsets of G that do not contain e_1, and among them, the subsets that satisfy p_{ab} are the supersets of $\{e_2, e_3\}$ and $\{e_4, e_5, e_6\}$. Hence, we have that $\text{Shapley}(G, p_{ab}, e_1) = \frac{35}{60}$ (the detailed computation is in the extended version of the paper [21]). A similar reasoning shows that $\text{Shapley}(G, p_{ab}, e_2) = \text{Shapley}(G, p_{ab}, e_3) = \frac{8}{60}$, and that $\text{Shapley}(G, p_{ab}, e_i) = \frac{3}{60}$ for $i = 4, 5, 6$. ◀

Lastly, we consider aggregate functions over conjunctive queries.

▶ **Example 8.** We consider the queries α_1, α_2, and α_4 from Example 3. Ellen has no publications; hence, $\text{Shapley}(D, \alpha_j, f_5^{\text{a}}) = 0$ for $j \in \{1, 2, 4\}$. The contribution of f_1^{a} is the same in every permutation (20 for α_1 and 2 for α_2) since Alice is the single author of two published papers that have a total of 20 citations. Hence, $\text{Shapley}(D, \alpha_1, f_1^{\text{a}}) = 20$ and $\text{Shapley}(D, \alpha_2, f_1^{\text{a}}) = 2$. The total number of citations of Cathy's papers is also 20; however, Bob and David are her coauthors on paper C. Hence, if the fact f_3^{a} appears before f_2^{a} and f_4^{a} in a permutation, its contribution to the query result is 20 for α_1 and 2 for α_2, while if f_3^{a} appears after at least one of f_2^{a} or f_4^{a} in a permutation, its contribution is 12 for α_1 and 1 for α_2. Clearly, f_2^{a} appears before both f_3^{a} and f_4^{a} in one-third of the permutations. Thus, we have that $\text{Shapley}(D, \alpha_1, f_3^{\text{a}}) = \frac{1}{3} \cdot 20 + \frac{2}{3} \cdot 12 = \frac{44}{3}$ and $\text{Shapley}(D, \alpha_2, f_3^{\text{a}}) = \frac{1}{3} \cdot 2 + \frac{2}{3} \cdot 1 = \frac{4}{3}$. Using similar computations we obtain that $\text{Shapley}(D, \alpha_1, f_2^{\text{a}}) = \text{Shapley}(D, \alpha_1, f_4^{\text{a}}) = \frac{8}{3}$ and $\text{Shapley}(D, \alpha_2, f_2^{\text{a}}) = \text{Shapley}(D, \alpha_2, f_4^{\text{a}}) = \frac{1}{3}$.

Hence, the Shapley value of Alice, who is the single author of two papers with a total of 20 citations, is higher than the Shapley value of Cathy who also has two papers with a total of 20 citations, but shares one paper with other authors. Bob and David have the same Shapley value, since they share a single paper, and this value is the lowest among the four, as they have the lowest number of papers and citations.

Finally, consider α_4. The contribution of f_1^{a} in this case depends on the maximum value before adding f_1^{a} in the permutation (which can be 0, 8 or 12). For example, if f_1^{a} is the first fact in the permutation, its contribution is 18 since $\alpha_4(\emptyset) = 0$. If f_1^{a} appears after f_3^{a}, then its contribution is 6, since $\alpha_4(S) = 12$ whenever $f_3^{\text{a}} \in S$. We have that $\text{Shapley}(D, \alpha_4, f_1^{\text{a}}) = 10$, $\text{Shapley}(D, \alpha_4, f_2^{\text{a}}) = \text{Shapley}(D, \alpha_4, f_4^{\text{a}}) = 2$ and $\text{Shapley}(D, \alpha_4, f_3^{\text{a}}) = 4$ (we omit the computations here). We see that the Shapley value of f_1^{a} is much higher than the rest, since

Alice significantly increases the maximum value when added to any prefix. If the number of citations of paper C increases to 16, then $\mathrm{Shapley}(D, \alpha_4, f_1^{\mathsf{a}}) = 6$, hence lower. This is because the next highest value is closer; hence, the contribution of f_1^{a} diminishes. ◀

4 Complexity Results

In this section, we give complexity results on the computation of the Shapley value of facts. We begin with exact evaluation for Boolean CQs (Section 4.1), then move on to exact evaluation on aggregate-relational queries (Section 4.2), and finally discuss approximate evaluation (Section 4.3). In the first two parts we restrict the discussion to CQs without self-joins, and leave the problems open in the presence of self-joins. However, the approximate treatment in the third part covers the general class of CQs (and beyond).

4.1 Boolean Conjunctive Queries

We investigate the problem of computing the (exact) Shapley value w.r.t. a Boolean CQ without self-joins. Our main result in this section is a full classification of (i.e., a dichotomy in) the data complexity of the problem. As we show, the classification criterion is the same as that of query evaluation over tuple-independent probabilistic databases [9]: hierarchical CQs without self-joins are tractable, and non-hierarchical ones are intractable.

▶ **Theorem 9.** *Let q be a Boolean CQ without self-joins. If q is hierarchical, then computing* $\mathrm{Shapley}(D, q, f)$ *can be done in polynomial time, given D and f. Otherwise, the problem is* $\mathrm{FP}^{\#\mathrm{P}}$*-complete.*

Recall that $\mathrm{FP}^{\#\mathrm{P}}$ is the class of functions computable in polynomial time with an oracle to a problem in #P (e.g., counting the number of satisfying assignments of a propositional formula). This complexity class is considered intractable, and is known to be above the polynomial hierarchy (Toda's theorem [35]).

▶ **Example 10.** Consider the query q_1 from Example 2. This query is hierarchical; hence, by Theorem 9, $\mathrm{Shapley}(D, q_1, f)$ can be calculated in polynomial time, given D and f. On the other hand, the query q_2 is not hierarchical. Thus, Theorem 9 asserts that computing $\mathrm{Shapley}(D, q_2, f)$ is $\mathrm{FP}^{\#\mathrm{P}}$-complete. ◀

In the rest of this subsection, we discuss the proof of Theorem 9. While the tractability condition is the same as that of Dalvi and Suciu [9], it is not clear whether and/or how we can use their dichotomy to prove ours, in each of the two directions (tractability and hardness). The difference is mainly in that they deal with a random subset of probabilistically independent (endogenous) facts, whereas we reason about random *permutations* over the facts. We stary by discussing the algorithm for computing the Shapley value in the hierarchical case, and then we discuss the proof of hardness for the non-hierarchical case.

Tractability side. Let D be a database, let f be an endogenous fact, and let q be a Boolean query. The computation of $\mathrm{Shapley}(D, q, f)$ easily reduces to the problem of counting the k-sets (i.e., sets of size k) of endogenous facts that, along with the exogenous facts, satisfy q. More formally, the reduction is to the problem of computing $|\mathrm{Sat}(D, q, k)|$ where $\mathrm{Sat}(D, q, k)$ is the set of all subsets E of D_{n} such that $|E| = k$ and $(D_{\mathsf{x}} \cup E) \models q$. The reduction is as follows, where we denote $m = |D_{\mathsf{n}}|$ and slightly abuse the notation by viewing q as a 0/1-numerical query, where $q(D') = 1$ if and only if $D' \models q$.

$$\mathrm{Shapley}(D,q,f) = \sum_{E\subseteq(D_{\mathsf{n}}\setminus\{f\})} \frac{|E|!(m-|E|-1)!}{m!}\Big(q(D_{\mathsf{x}}\cup E\cup\{f\}) - q(D_{\mathsf{x}}\cup E)\Big) \tag{4}$$

$$= \sum_{E\subseteq(D_{\mathsf{n}}\setminus\{f\})} \frac{|E|!(m-|E|-1)!}{m!}\Big(q(D_{\mathsf{x}}\cup E\cup\{f\})\Big) - \sum_{E\subseteq(D_{\mathsf{n}}\setminus\{f\})} \frac{|E|!(m-|E|-1)!}{m!}\Big(q(D_{\mathsf{x}}\cup E)\Big)$$

$$= \left(\sum_{k=0}^{m-1} \frac{k!(m-k-1)!}{m!} \times |\mathrm{Sat}(D',q,k)|\right) - \left(\sum_{k=0}^{m-1} \frac{k!(m-k-1)!}{m!} \times |\mathrm{Sat}(D\setminus\{f\},q,k)|\right)$$

In the last expression, D' is the same as D, except that f is viewed as *exogenous* instead of *endogenous*. Hence, to prove the positive side of Theorem 9, it suffices to show the following.

▶ **Theorem 11.** *Let q be a hierarchical Boolean CQ without self-joins. There is a polynomial-time algorithm for computing the number $|\mathrm{Sat}(D,q,k)|$ of subsets E of D_{n} such that $|E| = k$ and $(D_{\mathsf{x}}\cup E)\models q$, given D and k.*

To prove Theorem 11, we show a polynomial-time algorithm for computing $|\mathrm{Sat}(D,q,k)|$ for q as in the theorem. The pseudocode is depicted in Figure 2.

We assume in the algorithm that D_{n} contains only facts that are homomorphic images of atoms of q (i.e., facts f such that there is a mapping from an atom of q to f). In the terminology of Conitzer and Sandholm [7], the function defined by q *concerns* only the subset C of D_{n} consisting of these facts (i.e., the satisfaction of q by any subset of D does not change if we intersect with C), and so, the Shapley value of every fact in $D_{\mathsf{n}}\setminus C$ is zero and the Shapley value of any other fact is unchanged when ignoring $D_{\mathsf{n}}\setminus C$ [7, Lemma 4]. Moreover, these facts can be found in polynomial time.

As expected for a hierarchical query, our algorithm is a recursive procedure that acts differently in three different cases: *(a)* q has no variables (only constants), *(b)* there is a *root* variable x, that is, x occurs in all atoms of q, or *(c)* q consists of two (or more) subqueries that do not share any variables. Since q is hierarchical, at least one of these cases always applies [10].

In the first case (lines 1-7), every atom a of q can be viewed as a fact. Clearly, if one of the facts in q is not present in D, then there is no subset E of D_{n} of any size such that $(D_{\mathsf{x}}\cup E)\models q$, and the algorithm will return 0. Otherwise, suppose that A is the set of endogenous facts of q (and the remaining atoms of q, if any, are exogenous). Due to our assumption that every fact of D_{n} is a homomorphic image of an atom of q, the single choice of a subset of facts that makes the query true is A; therefore, the algorithm returns 1 if $k = |A|$ and 0 otherwise.

Next, we consider the case where q has a root variable x (lines 9-21). We denote by V_x the set $\{v_1,\dots,v_n\}$ of values that D has in attributes that correspond to an occurrence of x. For example, if q contains the atom $R(x,y,x)$ and D contains a fact $R(\mathtt{a},\mathtt{b},\mathtt{a})$, then $\mathtt{a}$ is one of the values in V_x. We also denote by $q_{[x\to v_i]}$ the query that is obtained from q by substituting v_i for x, and by D^{v_i} the subset of D that consists of facts with the value v_i in every attribute where x occurs in q.

We solve the problem for this case using a simple dynamic program. We denote by P_i^ℓ the number of subsets of size ℓ of $\bigcup_{r=1}^{i} D_{\mathsf{n}}^{v_r}$ that satisfy the query (together with the exogenous facts in $\bigcup_{r=1}^{i} D_{\mathsf{x}}^{v_r}$). Our goal is to find P_n^k, which is the number of subsets E of size k of $\bigcup_{r=1}^{n} D_{\mathsf{n}}^{v_r}$. Note that this union is precisely D_{n}, due to our assumption that D_{n} contains only facts that can be obtained from atoms of q via an assignment to the variables. First, we compute, for each value v_i, and for each $j\in\{0,\dots,k\}$, the number $f_{i,j}$ of subsets E of size j of $D_{\mathsf{n}}^{v_i}$ such that $(D_{\mathsf{x}}^{v_i}\cup E)\models q$, using a recursive call. In the recursive call, we replace q with $q_{[x\to v_i]}$, as D^{v_i} contains only facts that use the value v_i for the variable x; hence, we can

Algorithm 1 $\mathsf{CntSat}(D, q, k)$.

```
if $\mathsf{Vars}(q) = \emptyset$ then
    if $\exists a \in \mathsf{Atoms}(q)$ s.t. $a \notin D$ then
        return 0
    $A = \mathsf{Atoms}(q) \cap D_{\mathsf{n}}$
    if $|A| = k$ then
        return 1
    return 0
result $\leftarrow 0$
if $q$ has a root variable that occurs in all atoms then
    $x \leftarrow$ a root variable of $q$
    $V_x \leftarrow$ the set $\{v_1, \dots, v_n\}$ of values for $x$
    for all $i \in \{1, \dots, |V_x|\}$ do
        for all $j \in \{0, \dots, k\}$ do
            $f_{i,j} =\leftarrow \mathsf{CntSat}(D^{v_i}, q_{[x \to v_i]}, j)$
    $P_1^\ell = f_{1,\ell}$ for all $\ell \in \{0, \dots, k\}$
    for all $i \in \{2, \dots, |V_x|\}$ do
        for all $\ell \in \{0, \dots, k\}$ do
            $P_i^\ell \leftarrow 0$
            for all $j \in \{0, \dots, \ell\}$ do
                $P_i^\ell \leftarrow P_i^\ell + P_{i-1}^{\ell-j} \cdot f_{i,j} + \left[\binom{\sum_{r=1}^{i-1} |D_{\mathsf{n}}^{v_r}|}{\ell-j} - P_{i-1}^{\ell-j}\right] \cdot f_{i,j} + P_{i-1}^{\ell-j} \cdot \left[\binom{|D_{\mathsf{n}}^{v_i}|}{j} - f_{i,j}\right]$
    result $\leftarrow P_n^k$
else
    let $q = q_1 \wedge q_2$ where $\mathsf{Vars}(q_1) \cap \mathsf{Vars}(q_2) = \emptyset$
    let $D^1$ and $D^2$ be the restrictions of $D$ to the relations of $q_1$ and $q_2$, respectively
    for all $k_1, k_2$ s.t. $k_1 + k_2 = k$ do
        result $\leftarrow$ result $+ \mathsf{CntSat}(D^1, q_1, k_1) \cdot \mathsf{CntSat}(D^2, q_2, k_2)$
return result
```

Figure 2 An algorithm for computing $|\mathrm{Sat}(D, q, k)|$ where q is a hierarchical Boolean CQ without self-joins.

reduce the number of variables in q by substituting x with v_i. Then, for each $\ell \in \{0, \dots, k\}$ it clearly holds that $P_1^\ell = f_{1,\ell}$. For each $i \in \{2, \dots, |V_x|\}$ and $\ell \in \{0, \cdots, k\}$, we compute P_i^ℓ in the following way. Each subset E of size ℓ of $\bigcup_{r=1}^{i} D_{\mathsf{n}}^{v_r}$ contains a set E_1 of size j of facts from $D_{\mathsf{n}}^{v_i}$ (for some $j \in \{0, \dots, \ell\}$) and a set E_2 of size $\ell - j$ of facts from $\bigcup_{r=1}^{i-1} D_{\mathsf{n}}^{v_r}$. If the subset E satisfies the query, then precisely one of the following holds:

1. $(D_{\mathsf{x}}^{v_i} \cup E_1) \models q$ and $(\bigcup_{r=1}^{i-1} D_{\mathsf{x}}^{v_r} \cup E_2) \models q$,
2. $(D_{\mathsf{x}}^{v_i} \cup E_1) \models q$, but $(\bigcup_{r=1}^{i-1} D_{\mathsf{x}}^{v_r} \cup E_2) \not\models q$,
3. $(D_{\mathsf{x}}^{v_i} \cup E_1) \not\models q$, but $(\bigcup_{r=1}^{i-1} D_{\mathsf{x}}^{v_r} \cup E_2) \models q$.

Hence, we add to P_i^ℓ the value $P_{i-1}^{\ell-j} \cdot f_{i,j}$ that corresponds to Case (1), the value

$$\left(\binom{\bigcup_{r=1}^{i-1} |D_{\mathsf{n}}^{v_r}|}{\ell - j} - P_{i-1}^{\ell-j}\right) \cdot f_{i,j}$$

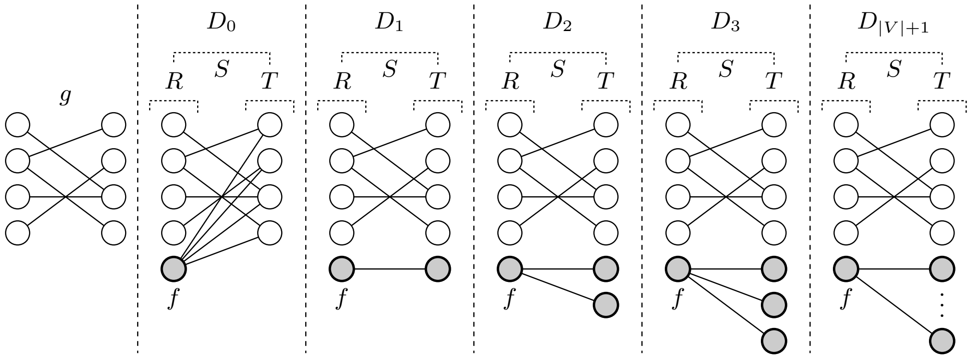


Figure 3 Constructions in the reduction of the proof of Lemma 12. Relations $R/1$ and $T/1$ consist of endogenous facts and $S/2$ consists of exogenous facts.

that corresponds to Case (2), and the value

$$P_{i-1}^{\ell-j} \cdot \left(\binom{|D_{\mathsf{n}}^{v_i}|}{j} - f_{i,j}\right)$$

that corresponds to Case (3). Note that we have all the values $P_{i-1}^{\ell-j}$ from the previous iteration of the for loop of line 16.

Finally, we consider the case where q has two nonempty subqueries q_1 and q_2 with disjoint sets of variables (lines 23-26). For $j \in \{1,2\}$, we denote by D^j the set of facts from D that appear in the relations of q_j. (Recall that q has no self-joins; hence, every relation can appear in either q_1 or q_2, but not in both.) Every subset E of D that satisfies q must contain a subset E_1 of D^1 that satisfies q_1 and a subset E_2 of D^2 satisfying q_2. Therefore, to compute $|\mathrm{Sat}(D,q,k)|$, we consider every pair (k_1,k_2) of natural numbers such that $k_1+k_2=k$, compute $|\mathrm{Sat}(D^1,q_1,k_1)|$ and $|\mathrm{Sat}(D^2,q_2,k_2)|$ via a recursive call, and add the product of the two to the result.

Hardness side. We now sketch the proof of the negative side of Theorem 9. (The complete proof is in [21].) Membership in $\mathrm{FP}^{\#\mathrm{P}}$ is straightforward since, as aforementioned in Equation (4), the Shapley value can be computed in polynomial time given an oracle to the problem of counting the number of subsets $E \subseteq D_{\mathsf{n}}$ of size k such that $(D_{\mathsf{x}} \cup E) \models q$, and this problem is in #P. Similarly to Dalvi and Suciu [9], our proof of hardness consists of two steps. First, we prove the $\mathrm{FP}^{\#\mathrm{P}}$-hardness of computing $\mathrm{Shapley}(D, \mathsf{q_{RST}}, f)$, where $\mathsf{q_{RST}}$ is given in (1). Second, we reduce the computation of $\mathrm{Shapley}(D, \mathsf{q_{RST}}, f)$ to the problem of computing $\mathrm{Shapley}(D,q,f)$ for any non-hierarchical CQ q without self-joins. The second step is the same as that of Dalvi and Suciu [9], so we do not discuss it here. Hence, in what follows, we focus on the first step – hardness of computing $\mathrm{Shapley}(D, \mathsf{q_{RST}}, f)$, as stated next by Lemma 12. The proof, which we discuss after the lemma, is considerably more involved than the corresponding proof of Dalvi and Suciu [9] that computing the probability of $\mathsf{q_{RST}}$ in a tuple-independent probabilistic database (TID) is $\mathrm{FP}^{\#\mathrm{P}}$-hard.

▶ **Lemma 12.** *Computing* $\mathrm{Shapley}(D, \mathsf{q_{RST}}, f)$ *is* $\mathrm{FP}^{\#\mathrm{P}}$*-complete.*

The proof of Lemma 12 is by a (Turing) reduction from the problem of computing the number $|\mathsf{IS}(g)|$ of independent sets of a given bipartite graph g, which is the same (via immediate reductions) as the problem of computing the number of satisfying assignments of a bipartite monotone 2-DNF formula, which we denote by #biSAT. Dalvi and Suciu [9] also proved the hardness of $\mathsf{q}_{\mathsf{RST}}$ (for the problem of query evaluation over TIDs) by reduction from #biSAT. Their reduction is a simple construction of a single input database, followed by a multiplication of the query probability by a number. It is not at all clear to us how such an approach can work in our case and, indeed, our proof is more involved. Our reduction takes the general approach that Dalvi and Suciu [10] used (in a different work) for proving that the CQ $q()$:- $R(x,y), R(y,z)$ is hard over TIDs: solve several instances of the problem for the construction of a full-rank set of linear equations. The problem itself, however, is quite different from ours. This general technique has also been used by Aziz et al. [2] for proving the hardness of computing the Shapley value for a *matching game* on unweighted graphs, which is again quite different from our problem.

In more detail, the idea is as follows. Given an input bipartite graph $g = (V, E)$ for which we wish to compute $|\mathsf{IS}(g)|$, we construct $n+1$ different input instances (D_j, f), for $j = 1, \dots, n+1$, of the problem of computing Shapley$(D_j, \mathsf{q}_{\mathsf{RST}}, f)$, where $n = |V|$. Each instance provides us with an equation over the numbers $|\mathsf{IS}(g,k)|$ of independent sets of size k in g for $k = 0, \dots, n$. We then show that the set of equations constitutes a non-singular matrix that, in turn, allows us to extract the $|\mathsf{IS}(g,k)|$ in polynomial time (e.g., via Gaussian elimination). This is enough, since $|\mathsf{IS}(g)| = \sum_{k=0}^{n} |\mathsf{IS}(g,k)|$.

Our reduction is illustrated in Figure 3. Given the graph g (depicted in the leftmost part), we construct $n+2$ graphs by adding new vertices and edges to g. For each such graph, we build a database that contains an endogenous fact $R(v)$ for every left vertex v, an endogenous fact $T(u)$ for every right vertex u, and an exogenous fact $S(v,u)$ for every edge (v,u). In each constructed database D_j, the fact f represents a new left node, and we compute Shapley$(D_j, \mathsf{q}_{\mathsf{RST}}, f)$. In D_0, the node of f is connected to every right vertex. We use Shapley$(D_0, \mathsf{q}_{\mathsf{RST}}, f)$ to compute a specific value that we refer to later on. For $j = 1, \dots, n+1$, the database D_j is obtained from g by adding f and facts of j new right nodes, all connected to f. We show the following for all $j = 1, \dots, n+1$.

$$\text{Shapley}(D_j, \mathsf{q}_{\mathsf{RST}}, f) = 1 - \frac{c_j \cdot v_0 + \sum_{k=0}^{n} |\mathsf{IS}(g,k)| \cdot k!(n+j-k)!}{(n+j+1)!}$$

where v_0 is a value computed using Shapley$(D_0, \mathsf{q}_{\mathsf{RST}}, f)$, and c_j is a constant that depends on j. From these equations we extract a system $Ax = y$ of $n+1$ equations over $n+1$ variables (i.e., $|\mathsf{IS}(g,0)|, \dots, |\mathsf{IS}(g,n)|$), where each S_j stands for Shapley$(D_j, \mathsf{q}_{\mathsf{RST}}, f)$.

$$\begin{pmatrix} 0!(n+1)! & 1!n! & \dots & n!1! \\ 0!(n+2)! & 1!(n+1)! & \dots & n!2! \\ \vdots & \vdots & \vdots & \vdots \\ 0!(2n+1)! & 1!(2n)! & \dots & n!(n+1)! \end{pmatrix} \begin{pmatrix} |\mathsf{IS}(g,0)| \\ |\mathsf{IS}(g,1)| \\ \vdots \\ |\mathsf{IS}(g,n)| \end{pmatrix} = \begin{pmatrix} (n+2)!S_1 - c_1 v_0 \\ (n+3)!S_2 - c_2 v_0 \\ \vdots \\ (2n+2)!S_{n+1} - c_{n+1} v_0 \end{pmatrix}$$

By an elementary algebraic manipulation of A, we obtain the matrix with the coefficients $a_{i,j} = (i+j+1)!$ that Bacher [3] proved to be non-singular (and, in fact, that $\prod_{i=0}^{n-1} i!(i+1)!$ is its determinant). We then solve the system as discussed earlier to obtain $|\mathsf{IS}(g,k)|$.

4.2 Aggregates over Conjunctive Queries

Next, we study the complexity of aggregate-relational queries, where the internal relational query is a CQ. We begin with hardness. The following theorem generalizes the hardness side of Theorem 9 and states that it is $\mathrm{FP}^{\#\mathrm{P}}$-complete to compute Shapley(D, α, f) whenever α is of the form $\gamma[q]$, as defined in Section 2, and q is a non-hierarchical CQ without self-joins. The only exception is when α is a *constant* numerical query (i.e., $\alpha(D) = \alpha(D')$ for all databases D and D'); in that case, Shapley$(D, \alpha, f) = 0$ always holds.

▶ **Theorem 13.** *Let $\alpha = \gamma[q]$ be a fixed aggregate-relational query where q is a non-hierarchical CQ without self-joins. Computing* $\text{Shapley}(D, \alpha, f)$*, given D and f, is* $\text{FP}^{\#\text{P}}$*-complete, unless α is constant.*

For instance, it follows from Theorem 13 that, whenever q is a non-hierarchical CQ without self-joins, it is $\text{FP}^{\#\text{P}}$-complete to compute the Shapley value for the aggregate-relational queries $\mathsf{count}[q]$, $\mathsf{sum}\langle\varphi\rangle[q]$, $\mathsf{max}\langle\varphi\rangle[q]$, and $\mathsf{min}\langle\varphi\rangle[q]$, unless $\varphi(\vec{c}) = 0$ for all databases D and tuples $\vec{c} \in q(D)$. Additional examples follow.

▶ **Example 14.** Consider the numerical query α_3 from Example 3. Since q_4 is not hierarchical, Theorem 13 implies that computing $\text{Shapley}(D, \alpha_4, f)$ is $\text{FP}^{\#\text{P}}$-complete. Actually, computing $\text{Shapley}(D, \alpha, f)$ is $\text{FP}^{\#\text{P}}$-complete for any non-constant aggregate-relational query over q_4. Hence, computing the Shapley value w.r.t. $\mathsf{count}[q_4]$ (which counts the number of papers in Citations with an author from California) or w.r.t. $\mathsf{max}\langle[2]\rangle[q_4]$ (which calculates the number of citations for the most cited paper by a Californian) is $\text{FP}^{\#\text{P}}$-complete as well. ◀

To prove hardness in Theorem 13, we break q into connected components $q_1, \dots, q_m$, such that $\mathsf{Vars}(q_i) \cap \mathsf{Vars}(q_j) = \emptyset$ for all $i, j \in \{1, \dots, m\}$. Since q is non-hierarchical, at least one of these connected components is non-hierarchical. We assume, without loss of generality, that this is q_1. Next, since α is not a constant function, there exists a database $\widetilde{D}$ such that $\alpha(\widetilde{D}) \neq \alpha(\emptyset)$. We select one answer $\vec{a}$ from $q(\widetilde{D})$ and substitute the free variables of q_1 with the corresponding constants from $\vec{a}$ to obtain the Boolean CQ q_1'. Theorem 9 states that computing $\text{Shapley}(D, q_1', f)$ is $\text{FP}^{\#\text{P}}$-complete. We then reduce the problem of computing $\text{Shapley}(D, q_1', f)$ to the problem of computing $\text{Shapley}(D, \alpha, f)$, and show that

$$\text{Shapley}(D, q_1', f) = \frac{\text{Shapley}(D', \alpha, f)}{\alpha(\widetilde{D}) - \alpha(\emptyset)}$$

where D' is a database obtained by combining facts from D with facts from $\widetilde{D}$. As usual, the full proof is given in the extended version of the paper [21].

Interestingly, it turns out that Theorem 13 captures precisely the hard cases for computing the Shapley value w.r.t. any summation over CQs without self-joins. In particular, the following argument shows that $\text{Shapley}(D, \mathsf{sum}\langle\varphi\rangle[q], f)$ can be computed in polynomial time if q is a hierarchical CQ without self-joins. Let $q = q(\vec{x})$ be an arbitrary CQ. For $\vec{a} \in q(D)$, let $q_{[\vec{x} \to \vec{a}]}$ be the Boolean CQ obtained from q by substituting every free variable x_j with the value of x_j in $\vec{a}$. Hence, we have that $\mathsf{sum}\langle\varphi\rangle[q] = \sum_{\vec{a} \in q(D)} \varphi(\vec{a}) \cdot q_{[\vec{x} \to \vec{a}]}$. The linearity of the Shapley value (stated as a fundamental property in Section 3) implies that

$$\text{Shapley}(D, \mathsf{sum}\langle\varphi\rangle[q], f) = \sum_{\vec{a} \in q(D)} \varphi(\vec{a}) \cdot \text{Shapley}(D, q_{[\vec{x} \to \vec{a}]}, f)\,. \tag{5}$$

Then, from Theorem 9 we conclude that if q is a hierarchical CQ with self-joins, then $\text{Shapley}(D, q_{[\vec{x} \to \vec{a}]}, f)$ can be computed in polynomial time for every $\vec{a} \in q(D)$. Hence, we have the following corollary of Theorem 9.

▶ **Corollary 15.** *Let q be a hierarchical CQ without self-joins. If α is an aggregate-relational query* $\mathsf{sum}\langle\varphi\rangle[q]$*, then* $\text{Shapley}(D, \alpha, f)$ *can be computed in polynomial time, given D and f. In particular,* $\text{Shapley}(D, \mathsf{count}[q], f)$ *can be computed in polynomial time.*

Together with Theorem 13, we get a full dichotomy for $\mathsf{sum}\langle\varphi\rangle[q]$ over CQs without self-joins.

The complexity of computing $\text{Shapley}(D, \alpha, f)$ for other aggregate-relational queries remains an open problem for the general case where q is a hierarchical CQ without self-joins. We can, however, state a positive result for $\mathsf{max}\langle\varphi\rangle[q]$ and $\mathsf{min}\langle\varphi\rangle[q]$ for the special case where q consists of a single atom (i.e., aggregation over a single relation).

▶ **Proposition 16.** *Let q be a CQ with a single atom. Then,* $\mathrm{Shapley}(D, \mathsf{max}\langle\varphi\rangle[q], f)$ *and* $\mathrm{Shapley}(D, \mathsf{min}\langle\varphi\rangle[q], f)$ *can be computed in polynomial time.*

As an example, if α is the query $\mathsf{max}\langle[2]\rangle[q]$, where q is given by $q(x, y)$:- $\textsc{Citations}(x, y)$, then we can compute in polynomial time $\mathrm{Shapley}(D, \alpha, f)$, determining the responsibility of each publication (in our running example) to the maximum number of citations.

4.3 Approximation

In computational complexity theory, a conventional feasibility notion of arbitrarily tight approximations is via the *Fully Polynomial-Time Approximation Scheme*, FPRAS for short. Formally, an FPRAS for a numeric function f is a randomized algorithm $A(x, \epsilon, \delta)$, where x is an input for f and $\epsilon, \delta \in (0, 1)$, that returns an ϵ-approximation of $f(x)$ with probability $1 - \delta$ (where the probability is over the randomness of A) in time polynomial in x, $1/\epsilon$ and $\log(1/\delta)$. To be more precise, we distinguish between an *additive* (or *absolute*) FPRAS:

$$\Pr\left[f(x) - \epsilon \le A(x, \epsilon, \delta) \le f(x) + \epsilon\right] \ge 1 - \delta\,,$$

and a *multiplicative* (or *relative*) FPRAS:

$$\Pr\left[\frac{f(x)}{1+\epsilon} \le A(x, \epsilon, \delta) \le (1 + \epsilon)f(x)\right] \ge 1 - \delta\,.$$

Using the Chernoff-Hoeffding bound, we easily get an additive FPRAS of $\mathrm{Shapley}(D, q, f)$ when q is *any* monotone Boolean query computable in polynomial time, by simply taking the ratio of successes over $O(\log(1/\delta)/\epsilon^2)$ trials of the following experiment:

1. Select a random permutation $(f_1, \dots, f_n)$ over the set of all endogenous facts.
2. Suppose that $f = f_i$, and let $D_{i-1} = D_{\mathsf{x}} \cup \{f_1, \dots, f_{i-1}\}$. If $q(D_{i-1})$ is false and $q(D_{i-1} \cup \{f\})$ is true, then report "success;" otherwise, "failure."

In general, an additive FPRAS of a function f is not necessarily a multiplicative one, since $f(x)$ can be very small. For example, we can get an additive FPRAS of the satisfaction of a propositional formula over Boolean i.i.d. variables by, again, sampling the averaging, but there is no multiplicative FPRAS for such formulas unless BPP = NP. Nevertheless, the situation is different for $\mathrm{Shapley}(D, q, f)$ when q is a CQ, since the Shapley value is never too small (assuming data complexity).

▶ **Proposition 17.** *Let q be a fixed Boolean CQ. There is a polynomial p such that for all databases D and endogenous facts f of D it is the case that* $\mathrm{Shapley}(D, q, f)$ *is either zero or at least* $1/(p(|D|))$.

Proof. We denote $m = |D_{\mathsf{n}}|$. If there is no subset S of D_{n} such that f is a counterfactual cause for q w.r.t. S, then $\mathrm{Shapley}(D, q, f) = 0$. Otherwise, let S be a minimal such set (i.e., for every $S' \subset S$, we have that $(S' \cup D_{\mathsf{x}}) \not\models q$). Clearly, it holds that $S \le k$, where k is the number of atoms of q. The probability to choose a permutation σ, such that σ_f is exactly $S \setminus \{f\}$ is $\frac{(|S|-1)!(m-|S|)!}{m!} \ge \frac{(m-k)!}{m!}$ (recall that σ_f is the set of facts that appear before f in σ). Hence, we have that $\mathrm{Shapley}(D, q, f) \ge \frac{1}{(m-k+1)\cdot\ldots\cdot m}$, and that concludes our proof. ◀

It follows that whenever $\mathrm{Shapley}(D, q, f) = 0$, the above additive approximation is also zero, and when $\mathrm{Shapley}(D, q, f) > 0$, the additive FPRAS also provides a multiplicative FPRAS. Hence, we have the following.

▶ **Corollary 18.** *For every fixed Boolean CQ, the Shapley value has both an additive and a multiplicative FPRAS.*

Interestingly, Corollary 18 generalizes to a multiplicative FPRAS for summation (including counting) over CQs. By combining Corollary 18 with Equation (5), we immediately obtain a multiplicative FPRAS for $\mathrm{Shapley}(D, \mathsf{sum}\langle\varphi\rangle[q], f)$, in the case where all the features $\varphi(\vec{a})$ in the summation have the same sign (i.e., they are either all negative or all non-negative). In particular, there is a multiplicative FPRAS for $\mathrm{Shapley}(D, \mathsf{count}[q], f)$.

▶ **Corollary 19.** *For every fixed CQ q, $\mathrm{Shapley}(D, \mathsf{sum}\langle\varphi\rangle[q], f)$ has a multiplicative FPRAS if either $\varphi(\vec{a}) \geq 0$ for all $\vec{a} \in q(D)$ or $\varphi(\vec{a}) \leq 0$ for all $\vec{a} \in q(D)$.*

Observe that the above FPRAS results allow the CQ q to have self-joins. This is in contrast to the complexity results we established in the earlier parts of this section, regarding exact evaluation. In fact, an easy observation is that Proposition 17 continues to hold when considering *unions of conjunctive queries* (UCQs). Therefore, Corollaries 18 and 19 remain correct in the case where q is a UCQ.

5 Related Measures

Causality and causal responsibility [15, 28] have been applied in data management [24], defining a fact as a *cause* for a query result as follows: For an instance $D = D_{\mathsf{x}} \cup D_{\mathsf{n}}$, a fact $f \in D_{\mathsf{n}}$ is an *actual cause* for a Boolean CQ q, if there exists $\Gamma \subseteq D_{\mathsf{n}}$, called a *contingency set* for f, such that f is a counterfactual cause for q in $D \setminus \Gamma$. The responsibility of an actual cause f for q is defined by $\rho(f) := \frac{1}{|\Gamma|+1}$, where $|\Gamma|$ is the size of a smallest contingency set for f. If f is not an actual cause, then $\rho(f)$ is zero [24]. Intuitively, facts with higher responsibility provide stronger explanations.[2]

▶ **Example 20.** Consider the database of our running example, and the query q_1 from Example 2. The fact f_1^{a} an actual cause with minimal contingency set $\Gamma = \{f_2^{\mathrm{a}}, f_3^{\mathrm{a}}, f_4^{\mathrm{a}}\}$. So, its responsibility is $\frac{1}{4}$. Similarly, f_2^{a}, f_3^{a} and f_4^{a} are actual causes with responsibility $\frac{1}{4}$.

▶ **Example 21.** Consider the database G and the query p_{ab} from Example 7. All facts in G are actual causes since every fact appears in a path from a to b. It is easy to verify that all the facts in D have the same causal responsibility, $\frac{1}{3}$, which may be considered as counter-intuitive given that e_1 provides a direct path from a to b.

As shown in Example 7, the Shapley value gives a more intuitive degree of contribution of facts to the query result than causal responsibility. Actually, Example 7 was used in [31] as a motivation to introduce an alternative to the notion of causal responsibility, that of *causal effect*, that we now briefly review.

To quantify the contribution of a fact to the query result, Salimi et al. [31] view the database as a tuple-independent probabilistic database where the probability of each endogenous fact is 0.5 and the probability of each exogenous fact is 1 (i.e., it is certain). The *causal effect* of a fact $f \in D_{\mathsf{n}}$ on a numerical query α is a difference of expected values [31]:

$$\mathrm{CE}(D, \alpha, f) \stackrel{\mathrm{def}}{=} \mathbb{E}(\alpha(D) \mid f) - \mathbb{E}(\alpha(D) \mid \neg f)\,.$$

where f is the event that the fact f is present in the database, and $\neg f$ is the event that the fact f is absent from the database.

[2] These notions can be applied to any monotonic query (i.e., whose answer set can only grow when the database grows, e.g., UCQs and Datalog queries) [4, 5].

▶ **Example 22.** Consider again the database of our running example, and the query q_1 from Example 2. We compute $\mathrm{CE}(D, q_1, f_1^{\mathrm{a}})$. It holds that: $\mathbb{E}(q_1 \mid \neg f_1^{\mathrm{a}}) = 0 \cdot P(q_1 = 0 \mid \neg f_1^{\mathrm{a}}) + 1 \cdot P(q_1 = 1 \mid \neg f_1^{\mathrm{a}}) = 1 - P(\neg f_2^{\mathrm{a}} \wedge \neg f_3^{\mathrm{a}} \wedge \neg f_4^{\mathrm{a}}) = \frac{7}{8}$. Similarly, we have that $\mathbb{E}(q_1 \mid f_1^{\mathrm{a}}) = P(q_1 = 1 \mid f_1^{\mathrm{a}}) = 1$. Then, $\mathrm{CE}(D, q_1, f_1^{\mathrm{a}}) = 1 - \frac{7}{8} = \frac{1}{8}$. Using similar computations we obtain that $\mathrm{CE}(D, q_1, f_2^{\mathrm{a}}) = \mathrm{CE}(D, q_1, f_3^{\mathrm{a}}) = \mathrm{CE}(D, q_1, f_4^{\mathrm{a}}) = \frac{1}{8}$.

For G and p_{ab} of Example 7, we have that $\mathrm{CE}(G, p_{ab}, e_1) = 0.65625$, $\mathrm{CE}(G, p_{ab}, e_2) = \mathrm{CE}(G, p_{ab}, e_3) = 0.21875$, $\mathrm{CE}(G, p_{ab}, e_4) = \mathrm{CE}(G, p_{ab}, e_5) = \mathrm{CE}(G, p_{ab}, e_6) = 0.09375$. ◀

Although the values in the two examples above are different from the Shapley values computed in Example 6 and Example 7, respectively, if we order the facts according to their contribution to the query result, we will obtain the same order in both cases. Note that unlike the Shapley value, for causal effect the sum of the values over all facts is not equal to the query result on the whole database. In the next example we consider aggregate queries.

▶ **Example 23.** Consider the query α_1 of Example 3. If f_1^{a} is in the database, then the result can be either 20, 28, or 40. If f_1^{a} is absent, then the query result can be either 0, 8, or 20. By computing the expected value in both cases, we obtain that $\mathrm{CE}(D, \alpha_1, f_1^{\mathrm{a}}) = 20$. Similarly, it holds that $\mathrm{CE}(D, \alpha_1, f_2^{\mathrm{a}}) = \mathrm{CE}(D, \alpha_1, f_4^{\mathrm{a}}) = 1$, and $\mathrm{CE}(D, \alpha_1, f_3^{\mathrm{a}}) = 14$. ◀

Interestingly, the causal effect coincides with a well known wealth-distribution function in cooperative games, namely the *Banzhaf Power Index* (BPI) [11, 18, 19]. This measure is defined similarly to the definition of the Shapley value in Equation (3), except that we replace the ratio $\frac{|B|! \cdot (|A| - |B| - 1)!}{|A|!}$ with $\frac{1}{2^{|A|-1}}$.

▶ **Proposition 24.** *Let α be a numerical query, D be a database, and $f \in D_{\mathsf{n}}$. Then,*

$$\mathrm{CE}(D, \alpha, f) = \frac{1}{2^{|D_{\mathsf{n}}|-1}} \cdot \sum_{E \subseteq (D_{\mathsf{n}} \setminus \{f\})} [\alpha(D_{\mathsf{x}} \cup E \cup \{f\}) - \alpha(D_{\mathsf{x}} \cup E)]$$

Hence, the causal effect coincides with the BPI.

We conjecture that *all* of the complexity results (exact and approximate) obtained in this work for the Shapley value apply to the causal effect (and BPI), with some of them being easier to obtain than for the Shapley value, via a connection to probabilistic databases [34].

6 Conclusions

We introduced the problem of quantifying the contribution of database facts to query results via the Shapley value. We investigated the complexity of the problem for Boolean CQs and for aggregates over CQs. Our dichotomy in the complexity of the problem establishes that computing the exact Shapley value is often intractable. Nevertheless, we also showed that the picture is far more optimistic when allowing approximation with strong precision guarantees.

Many questions, some quite fundamental, remain for future investigation. While we have a thorough understanding of the complexity for Boolean CQs without self-joins, very little is known in the presence of self-joins. For instance, the complexity is open even for the simple query $q() \coloneq R(x, y), R(y, z)$. We also have just a partial understanding of the complexity for aggregate functions over CQs, beyond the general hardness result for non-hierarchical queries (Theorem 13). In particular, it is important to complete the complexity analysis for maximum and minimum, and to investigate other common aggregate functions such as average, median, percentile, and standard deviation. Another direction is to investigate whether and how properties of the database, such as low treewidth, can reduce the (asymptotic and empirical) running time of computing the Shapley value. Interestingly, the implication of a low treewidth to Shapley computation has been studied for a different problem [13].

References

1 Robert J Aumann and Roger B Myerson. Endogenous formation of links between players and of coalitions: An application of the Shapley value. In *Networks and Groups*, pages 207–220. Springer, 2003.

2 Haris Aziz and Bart de Keijzer. Shapley meets Shapley. In *STACS*, pages 99–111, 2014.

3 Roland Bacher. Determinants of matrices related to the Pascal triangle. *Journal de Théorie des Nombres de Bordeaux*, 14, January 2002.

4 Leopoldo E. Bertossi and Babak Salimi. Causes for query answers from databases: Datalog abduction, view-updates, and integrity constraints. *Int. J. Approx. Reasoning*, 90:226–252, 2017.

5 Leopoldo E. Bertossi and Babak Salimi. From Causes for Database Queries to Repairs and Model-Based Diagnosis and Back. *Theory Comput. Syst.*, 61(1):191–232, 2017.

6 Sara Cohen, Werner Nutt, and Yehoshua Sagiv. Deciding equivalences among conjunctive aggregate queries. *J. ACM*, 54(2):5, 2007.

7 Vincent Conitzer and Tuomas Sandholm. Computing Shapley Values, Manipulating Value Division Schemes, and Checking Core Membership in Multi-issue Domains. In *AAAI*, pages 219–225. AAAI Press, 2004.

8 Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

9 Nilesh N. Dalvi and Dan Suciu. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

10 Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30:1–30:87, 2012.

11 Pradeep Dubey and Lloyd S. Shapley. Mathematical Properties of the Banzhaf Power Index. *Mathematics of Operations Research*, 4(2):99–131, 1979.

12 John Grant and Anthony Hunter. Measuring inconsistency in knowledgebases. *J. Intell. Inf. Syst.*, 27(2):159–184, 2006.

13 Gianluigi Greco, Francesco Lupia, and Francesco Scarcello. Structural Tractability of Shapley and Banzhaf Values in Allocation Games. In *IJCAI*, pages 547–553, 2015.

14 Faruk Gul. Bargaining foundations of Shapley value. *Econometrica: Journal of the Econometric Society*, pages 81–95, 1989.

15 Joseph Y. Halpern. *Actual Causality.* The MIT Press, 2016.

16 Joseph Y. Halpern and Judea Pearl. Causes and Explanations: A Structural-Model Approach: Part 1: Causes. In *UAI*, pages 194–202, 2001.

17 Anthony Hunter and Sébastien Konieczny. On the measure of conflicts: Shapley Inconsistency Values. *Artif. Intell.*, 174(14):1007–1026, 2010.

18 Werner Kirsch and Jessica Langner. Power indices and minimal winning coalitions. *Social Choice and Welfare*, 34(1):33–46, January 2010.

19 Dennis Leech. Power indices and probabilistic voting assumptions. *Public Choice*, 66(3):293–299, September 1990.

20 Zhenliang Liao, Xiaolong Zhu, and Jiaorong Shi. Case study on initial allocation of Shanghai carbon emission trading based on Shapley value. *Journal of Cleaner Production*, 103:338–344, 2015.

21 Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. The Shapley Value of Tuples in Query Answering. *CoRR*, abs/1904.08679, 2019.

22 Richard TB Ma, Dah Ming Chiu, John Lui, Vishal Misra, and Dan Rubenstein. Internet Economics: The use of Shapley value for ISP settlement. *IEEE/ACM Transactions on Networking (TON)*, 18(3):775–787, 2010.

23 Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y. Halpern, Christoph Koch, Katherine F. Moore, and Dan Suciu. Causality in Databases. *IEEE Data Eng. Bull.*, 33(3):59–67, 2010.

24 Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *PVLDB*, 4(1):34–45, 2010.

25 Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. WHY so? or WHY no? functional causality for explaining query answers. In *MUD*, volume WP10-04 of *CTIT Workshop Proceedings Series*, pages 3–17. CTIT, 2010.

26 Ramasuri Narayanam and Yadati Narahari. A Shapley value-based approach to discover influential nodes in social networks. *IEEE Transactions on Automation Science and Engineering*, 8(1):130–147, 2011.

27 Tatiana Nenova. The value of corporate voting rights and control: A cross-country analysis. *Journal of financial economics*, 68(3):325–351, 2003.

28 Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

29 Leon Petrosjan and Georges Zaccour. Time-consistent Shapley value allocation of pollution cost reduction. *Journal of economic dynamics and control*, 27(3):381–398, 2003.

30 Alvin E Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

31 Babak Salimi, Leopoldo E. Bertossi, Dan Suciu, and Guy Van den Broeck. Quantifying Causal Effects on Query Answering in Databases. In *TAPP*, 2016.

32 Lloyd Shapley and Martin Shubik. A Method for Evaluating the Distribution of Power in a Committee System. *American Political Science Review*, 48(03):787–792, 1954.

33 Lloyd S Shapley. A Value for n-Person Games. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953.

34 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

35 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

36 Bruno Yun, Srdjan Vesic, Madalina Croitoru, and Pierre Bisquert. Inconsistency Measures for Repair Semantics in OBDA. In *IJCAI*, pages 1977–1983. ijcai.org, 2018.

Optimal Joins Using Compact Data Structures

Gonzalo Navarro
University of Chile, Santiago, Chile
IMFD, Santiago, Chile

Juan L. Reutter
Pontificia Universidad Católica de Chile, Santiago, Chile
IMFD, Santiago, Chile

Javiel Rojas-Ledesma
University of Chile, Santiago, Chile
IMFD, Santiago, Chile

Abstract

Worst-case optimal join algorithms have gained a lot of attention in the database literature. We now count with several algorithms that are optimal in the worst case, and many of them have been implemented and validated in practice. However, the implementation of these algorithms often requires an enhanced indexing structure: to achieve optimality we either need to build completely new indexes, or we must populate the database with several instantiations of indexes such as B+-trees. Either way, this means spending an extra amount of storage space that may be non-negligible.

We show that optimal algorithms can be obtained directly from a representation that regards the relations as point sets in variable-dimensional grids, without the need of extra storage. Our representation is a compact quadtree for the static indexes, and a dynamic quadtree sharing subtrees (which we dub a qdag) for intermediate results. We develop a compositional algorithm to process full join queries under this representation, and show that the running time of this algorithm is worst-case optimal in data complexity. Remarkably, we can extend our framework to evaluate more expressive queries from relational algebra by introducing a lazy version of qdags (lqdags). Once again, we can show that the running time of our algorithms is worst-case optimal.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory); Theory of computation → Data structures and algorithms for data management

Keywords and phrases Join algorithms, Compact data structures, Quadtrees, AGM bound

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.21

Funding This work was partially funded by Millennium Institute for Foundational Research on Data (IMFD), and by project CONICYT Fondecyt/Postdoctorado No. 3190550.

1 Introduction

The state of the art in query processing has recently been shaken by a new generation of join algorithms with strong optimality guarantees based on the AGM bound of queries: the maximum size of the output of the query over all possible relations with the same cardinalities [2]. One of the basic principles of these algorithms is to disregard the traditional notion of a query plan, favoring a strategy that can take further advantage of the structure of the query, while at the same time taking into account the actual size of the database [14, 16].

Several of these algorithms have been implemented and tested in practice with positive results [8, 18], especially when handling queries with several joins. Because they differ from what is considered standard in relational database systems, the implementation of these algorithms often requires additional data structures, a database that is heavily indexed, or heuristics to compute the best computation path given the indexes that are present. For example, algorithms such as Leapfrog [21], Minesweeper [15], or InsideOut [10] must select a *global order* on the attributes, and assume that relations are indexed in a way that is consistent with these attributes [18]. If one wants to use these algorithms with more flexibility

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 21; pp. 21:1–21:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the way attributes are processed, then one would probably need to instantiate several combinations of B+ trees or other indexes [8]. On the other hand, more involved algorithms such as Tetris [9] or Panda [11] require heavier data structures that allow reasoning over potential tuples in the answer.

Our goal is to develop optimal join algorithms that minimize the storage for additional indexes while at the same time being independent of a particular ordering of attributes. We address this issue by resorting to compact data structures: indexes using a nearly-optimal amount of space while supporting all operations we need to answer join queries.

We show that worst-case optimal algorithms can be obtained when one assumes that the input data is represented as quadtrees, and stored under a compact representation for cardinal trees [4]. Quadtrees are geometric structures used to represent data points in grids of size $\ell \times \ell$ (which can be generalized to any dimension). Thus, a relation R with attributes $A_1, \dots, A_d$ can be naturally viewed as a set of points over grids of dimension d, one point per tuple of R: the value of each attribute A_i is the i-th coordinate of the corresponding point.

To support queries under this representation, our main tool is a new dynamic version of quadtrees, which we denote qdags, where some nodes may share complete subtrees. Using qdags, we can reduce the computation of a full join query $J = R_1 \bowtie \cdots \bowtie R_n$ with d attributes, to an algorithm that first extends the quadtrees for $R_1, \dots, R_n$ into qdags of dimension d, and then intersects them to obtain a quadtree. Our first result shows that such algorithm is indeed worst-case optimal:

▶ **Theorem 1.1.** *Let $R_1(\mathcal{A}_1), \dots, R_n(\mathcal{A}_n)$ be n relations with attributes in $[0, \ell-1]$, and let $d = |\bigcup_i \mathcal{A}_i|$. We can represent the relations using only $\sum_i(|\mathcal{A}_i|+2+o(1))|R_i|\log\ell+O(n\log d)$ bits, so that the result of a join query $J = R_1 \bowtie \cdots \bowtie R_n$ over a database instance D, can be computed in $\tilde{O}(AGM)$ time*[1]*.*

Note that just storing the tuples in any R_i requires $|\mathcal{A}_i||R_i|\log\ell$ bits, thus our representation adds only a small extra space of $(2+o(1))|R_i|\log\ell + O(n\log d)$ bits (basically, two words per tuple, plus a negligible amount that only depends on the schema). Instead, any classical index on the raw data (such as hash tables or B+-trees) would pose a linear extra space, $O(|\mathcal{A}_i||R_i|\log\ell)$ bits, often multiplied by a non-negligible constant (especially if one needs to store multiple indexes on the data).

Our join algorithm works in a rather different way than the most popular worst-case algorithms. To illustrate this, consider the triangle query $J = R(A,B) \bowtie S(B,C) \bowtie T(A,C)$. The most common way of processing this query optimally is to follow what Ngo et al. [16] define as the *generic* algorithm: select one of the attributes of the query (say A), and iterate over all elements $a \in A$ that could be an answer to this query, that is, all $a \in \pi_a(R) \cap \pi_a(T)$. Then, for each of these elements, iterate over all $b \in B$ such that the tuple (a,b) can be an answer: all (a,b) in $(R \bowtie \pi_B(S)) \bowtie \pi_A(T)$, and so on.

Instead, quadtrees divide the output space, which corresponds to a grid of size ℓ^3, into 8 subgrids of size $(\ell/2)^3$, and for each of these grids it recursively evaluates the query. As it turns out, this strategy is as good as the generic strategy defined by Ngo et al. [16] to compute joins, and can even be extended to other relational operations, as we explain next.

Our join algorithm boils down to two simple operations on quadtrees: an Extend operation that lifts the quadtree representation of a grid to a higher-dimensional grid, and an And operation that intersects trees. But there are other operations that we can define and implement. For example, the synchronized Or of two quadtrees gives a compact

[1] $\tilde{O}$ hides poly-log N factors, for N the total input size, as well as factors that just depend on d and n (i.e., the query size), which are assumed to be constant. We provide a precise bound in Section 3.3.

representation of their union, and complementing the quadtree values can be done by a Not operation. We integrate all these operations in a single framework, and use it to answer more complex queries given by the combination of these expressions, as in relational algebra.

To support these operations we introduce lazy qdags, or lqdags for short, in which nodes may be additionally labeled with query expressions. The idea is to be able to delay the computation of an expression until we know such computation is needed to derive the output. To analyze our framework we extend the idea of a worst-case optimal algorithm to arbitrary queries: If a *worst-case optimal algorithm* to compute the output of a formula F takes time T over relations $R_1, \ldots, R_n$ of sizes $s_1, \ldots, s_n$, respectively, of a database D, then there exists a database D' with relations $R'_1, \ldots, R'_n$ of sizes $O(s_1), \ldots, O(s_n)$, respectively, where the output of F over $R'_1, \ldots, R'_n$ is of size $\Omega(T)$. We prove that lqdags allow us to maintain optimality in the presence of union and negation operators:

▶ **Theorem 1.2.** *Let Q be a relational algebra query built with joins, union and complement, and where no relation appears more than once in Q. Then there is an algorithm to evaluate Q that is worst-case optimal in data complexity.*

Consider, for example, the query $J' = R(A,B) \bowtie S(B,C) \bowtie \overline{T}(A,C)$, which joins R and S with the complement $\overline{T}$ of T. One could think of two ways to compute this query. The first is just to join R and S and then see which of the resulting tuples are not in T. But if T is dense ($\overline{T}$ is small), it may be more efficient to first compute $\overline{T}$ and then proceed as on the usual triangle query. Our algorithm is optimal because it can choose locally between both strategies: by dividing into quadrants one finds dense regions of T in which computing $\overline{T}$ is cheaper, while in sparse regions the algorithm first computes the join of R and S.

Our framework is the first in combining worst-case time optimality with the use of compact data structures. The latter can lead to improved performance in practice, because relations can be stored in faster memory, higher in the memory hierarchy [13]. This is especially relevant when the compact representation fits in main memory while a heavily indexed representation requires resorting to the disk, which is orders of magnitude slower. Under the recent trend of maintaining the database in the aggregate main memory of a distributed system, a compact representation leads to using fewer computers, thus reducing hardware, communication, and energy costs, while improving performance.

2 Quadtrees

A Region Quadtree [6, 19] is a structure used to store points in two-dimensional grids of $\ell \times \ell$. We focus on the variant called MX-Quadtree [22, 19], which can be described as follows. Assume for simplicity that ℓ is a power of 2. If $\ell = 1$, then the grid has only one cell and the quadtree is an integer 1 (if the cell has a point) or 0 (if not). For $\ell > 1$, if the grid has no points, then the quadtree is a leaf. Otherwise, the quadtree is an internal node with four children, each of which is the quadtree of one of the four $\ell/2 \times \ell/2$ quadrants of the grid. (The deepest internal nodes, whose children are 1×1 grids, store instead four integers in $\{0, 1\}$ to encode their cells.)

Assume each data point is described using the binary representation of each of its coordinates (i.e., as a pair of $\log \ell$-bit vectors). We order the grid quadrants so that the first contains all points with coordinates of the form $(0 \cdot c_x, 0 \cdot c_y)$, for $\log \ell - 1$ bit vectors c_x and c_y, the second contains points $(0 \cdot c_x, 1 \cdot c_y)$, the third $(1 \cdot c_x, 0 \cdot c_y)$, and the last quadrant stores the points $(1 \cdot c_x, 1 \cdot c_y)$. Fig. 1 shows a grid and its deployment as a quadtree.

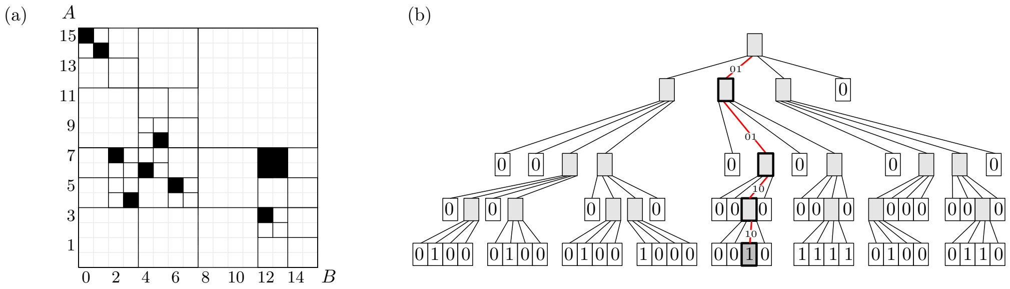


Figure 1 A quadtree representing $R(A, B) = \{(4, 3), (7, 2), (5, 6), (6, 4), (3, 12), (6, 12), (6, 13), (7, 12), (7, 13), (8, 5), (14, 1), (15, 0)\}$. (a) Representation of $R(A, B)$ in a $2^4 \times 2^4$ grid, and representation of the hierarchical partition defining the quadtree. The black cells correspond to points in R. (b) The quadtree representing R. The shadowed leaf of the tree corresponds to the point $p = (3, 12)$. Concatenating the labels in the path down to p yield the bit-string '**01011010**' which encodes the first (resp. second) coordinate of p in the bits at odd (resp. even) positions ($3 = 0011, 12 = 1100$).

Quadtrees can be generalized to higher dimensions. A quadtree of dimension d is a tree used to represent data points in a d-dimensional grid G of size ℓ^d. Here, an empty grid is represented by a leaf and a nonempty grid corresponds to an internal node with 2^d children representing the 2^d subspaces spanning from combining the first bits of each dimension. Generalizing the case $d = 1$, the children are ordered using the Morton [12] partitioning of the grid: a sequence of 2^d subgrids of size $(\ell/2)^d$ in which the i-th subgrid of the partition, represented by the binary encoding b_i of i, is defined by all the points $(b_{c_1}, \ldots, b_{c_d})$ in which the word formed by concatenating the first bit of each string b_{c_j} is precisely the string b_i.

A quadtree with p points has at most $p \log \ell$ nodes (i.e., root-to-leaf paths). A refined analysis in two dimensions [7, Thm. 1] shows that quadtrees have fewer nodes when the points are clustered: if the points distribute along c clusters, p_i of them inside a subgrid of size $\ell_i \times \ell_i$, then there are in total $O(c \log \ell + \sum_i p_i \log \ell_i)$ nodes in the quadtree. The result easily generalizes to d dimensions: the cells are of size ℓ_i^d and the quadtree has $O(c \log \ell + \sum_i p_i \log \ell_i)$ internal nodes, each of which stores 2^d pointers to children (or integers, in the last level).

Brisaboa et al. [4] introduced a compact quadtree representation called the k^d-tree. They represent each internal quadtree node as the 2^d bits telling which of its quadrants is empty (0) or nonempty (1). Leaves and single-cell nodes are not represented because their data is deduced from the corresponding bit of their parent. The k^d-tree is simply a bitvector V obtained by concatenating the 2^d bits of every (internal) node in levelwise order. Each node is identified with its order in this deployment, the root being 1. Navigation on the quadtree (from a parent to its children, and vice-versa) is simulated in constant time using $o(|V|)$ additional bits on top of V. On a quadtree in dimension d storing p points, the length of the bitvector V is $|V| \leq 2^d p \log \ell$, increasing exponentially with d. This bitvector is sparse, however, because it has at most $p \log \ell$ 1s, one per quadtree node. We then resort to a representation of high-arity cardinal trees introduced by Benoit et al. [3, Thm. 4.3], which requires only $(d+2)p \log \ell + o(p \log \ell) + O(\log d)$ bits, and performs the needed tree traversal operations in constant time.

▶ **Observation 2.1** (cf. Benoit et al. [3], Thm. 4.3). *Let Q be a quadtree storing p points in d dimensions with integer coordinates in the interval $[0, \log \ell - 1]$. Then, there is a representation of Q which uses $(d + 2 + o(1))p \log \ell + O(\log d)$ bits, can be constructed in linear expected time, and supports constant time parent-children navigation on the tree.*

From now on, by quadtree we refer to this compact representation. Next, we show how to represent relations using quadtrees and evaluate join queries over this representation.

3 Multi-way Joins using Qdags

We assume for simplicity that the domain $\mathcal{D}(A)$ of an attribute A consists of all binary strings of length $\log \ell$, representing the integers in $[0, \ell - 1]$, and that ℓ is a power of 2.

A relation $R(\mathcal{A})$ with attributes $\mathcal{A} = \{A_1, \ldots, A_d\}$ can be naturally represented as a quadtree: simply interpret each tuple in $R(\mathcal{A})$ as a data point over a d-dimensional grid with ℓ^d cells, and store those points in a d-dimensional quadtree. Thus, using quadtrees one can represent the relations in a database using compact space. The convenience of this representation to handle restricted join queries with naive algorithms has been demonstrated practically on RDF stores [1]. In order to obtain a general algorithm with provable performance, we introduce qdags, an enhanced version of quadtrees, together with a new algorithm to efficiently evaluate join queries over the compressed representations of the relations.

We start with an example to introduce the basics behind our algorithms and argue for the need of qdags. We then formally define qdags and explore their relation with quadtrees. Finally, we provide a complete description of the join algorithm and analyze its running time.

3.1 The triangle query: quadtrees vs qdags

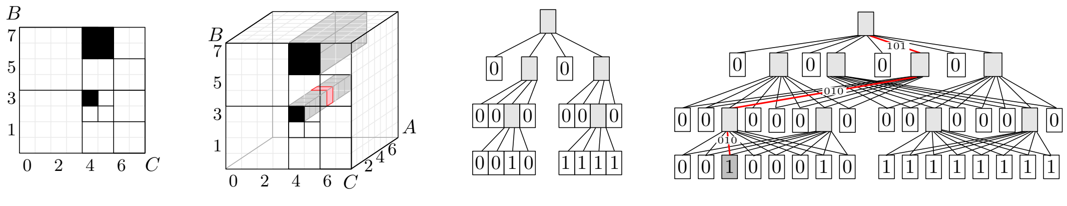
Let $R(A, B)$, $S(B, C)$, $T(A, C)$ be relations over the attributes $\{A, B, C\}$ denote the domains of A, B and C respectively, and consider the triangle query $R(A, B) \bowtie S(B, C) \bowtie T(A, C)$. The basic idea of the algorithm is as follows: we first compute a quadtree Q^*_R that represents the cross product $R(A, B) \times \text{All}(C)$, where $\text{All}(C)$ is a relation with an attribute C storing all elements in the domain $[0, \ell - 1]$. Likewise, we compute Q^*_S representing $S(B, C) \times \text{All}(A)$, and Q^*_T representing $T(A, C) \times \text{All}(B)$. Note that these quadtrees represent points in the three-dimensional grid with a cell for every possible value in $\mathcal{D}(A) \times \mathcal{D}(B) \times \mathcal{D}(C)$, where we assume that the domains $\mathcal{D}(\cdot)$ of the attributes are all $[0, \ell - 1]$. Finally, we traverse the three quadtrees in synchronization building a new quadtree that represents the intersection of Q^*_R, Q^*_S and Q^*_T. This quadtree represents the desired output because

$$R(A, B) \bowtie S(B, C) \bowtie T(A, C) =$$
$$(R(A, B) \times \text{All}(C)) \cap (S(B, C) \times \text{All}(A)) \cap (T(A, C) \times \text{All}(B)).$$

Though this algorithm is correct, it can perform poorly in terms of space and running time. The size of Q^*_R, for instance, can be considerably bigger than that of R, and even than the size of the output of the query. If, for example, the three relations have n elements each, the size of the output is bounded by $n^{3/2}$ [2], while building Q^*_R costs $\Omega(n\ell)$ time and space. This inefficiency stems from the fact that quadtrees are not smart to represent relations of the form $R^*(\mathcal{A}) = R(\mathcal{A}') \times \text{All}(\mathcal{A} \setminus \mathcal{A}')$, where $\mathcal{A}' \subset \mathcal{A}$, with respect to the size of a quadtree representing $R(\mathcal{A}')$. Due to its tree nature, a quadtree does not benefit from the regularities that appear in the grid representing $R^*(\mathcal{A})$. To remedy this shortcoming, we introduce qdags, quadtree-based data structures that represent sets of the form $R(\mathcal{A}') \times \text{All}(\mathcal{A} \setminus \mathcal{A}')$ by adding only *constant* additional space to the quadtree representing $R(\mathcal{A}')$, for any $\mathcal{A}' \subseteq \mathcal{A}$.

A qdag is an *implicit* representation of a d-dimensional quadtree Q^d (that has certain regularities) using only a reference to a d'-dimensional quadtree $Q^{d'}$, with $d' \leq d$, and an auxiliary *mapping function* that defines how to use $Q^{d'}$ to simulate navigation over Q^d. Qdags can then represent relations of the form $R(\mathcal{A}') \times \text{All}(\mathcal{A} \setminus \mathcal{A}')$ using only a reference to a quadtree representing $R(\mathcal{A}')$, and a constant-space mapping function.

To illustrate how a qdag works, consider a relation $S(B, C)$, and let Q^*_S be a quadtree representing $S^*(A, B, C) = \text{All}(A) \times S(B, C)$. Since Q^*_S stores points in the ℓ^3 cube, each node in Q^*_S has 8 children. As $\text{All}(A)$ contains all ℓ elements, for each original point (b, c) in



(a)

(b)

Figure 2 An illustration of a qdag for $S^*(\{A,B,C\}) = \text{All}(A) \times S(B,C)$, with $S(B,C) = \{(3,4),(6,4),(6,5),(7,4),(7,5)\}$. a) A geometric representation of $S(B,C)$ (left), and $S^*(\{A,B,C\})$ (right). b) A quadtree Q_S for $S(B,C)$ (left), and the directed acyclic graph induced by the qdag $(Q_S, M = [0,1,2,3,0,1,2,3])$, which represents $S^*(\{A,B,C\})$. The red cell in (a) corresponds to the point $p = (4,3,4)$. The leaf representing p in the qdag can be reached following the path highlighted in (b). Note the relation between the binary representation (**1**00,**0**10,**1**00) of p, and the Morton codes **101**, **010**, 010 of the nodes in the path from the root to the leaf for p.

S, S^* contains ℓ points corresponding to elements $(0,b,c),\ldots,(\ell-1,b,c)$. We can think of this as *extending* each point in S to a box of dimension $\ell \times 1 \times 1$. With respect to Q^*_S, this implies that, among the 8 children of a node, the last 4 children will always be identical to the first 4, and their values will in turn be identical to those of the corresponding nodes in the quadtree Q_S representing S. In other words, each of the four subgrids $1a_1a_2$ is identical to the subgrid $0a_1a_2$, and these in turn are identical to the subgrid a_1a_2 in S (see Fig. 2 for an example). Thus, we can implicitly represent Q^*_S by the pair $(Q_S, M = [0,1,2,3,0,1,2,3])$: the root of Q^*_S is the root of Q_S, and the i-child of the root of Q^*_S is represented by the pair (C, M), where C is the $M[i]$-th child of the root of Q_S.

3.2 Qdags for relational data

We now introduce a formal definition of the qdags, and describe the algorithms which allow the evaluation of multijoin queries in worst-case optimal time.

▶ **Definition 3.1** (qdag). *Let $Q^{d'}$ be a quadtree representing a relation with d' attributes. A qdag Q^d, for $d \geq d'$, is a pair $(Q^{d'}, M)$, with $M : [0, 2^d - 1] \to [0, 2^{d'} - 1]$. This qdag represents a quadtree Q, which is called the* completion *of Q^d, as follows:*

1. *If $Q^{d'}$ represents a single cell, then Q represents a single cell with the same value.*
2. *If $Q^{d'}$ represents a d'-dimensional grid empty of points, then Q represents a d-dimensional grid empty of points.*
3. *Otherwise, the roots of both $Q^{d'}$ and Q are internal nodes, and for all $0 \leq i < 2^d$, the i-th child of Q is the quadtree represented by the qdag $(C(Q^{d'}, M[i]), M)$, where $C(Q^{d'}, j)$ denotes the j-th child of the root node of quadtree $Q^{d'}$.*

We say that a qdag represents the same relation R represented by its completion. Note that, for any d-dimensional quadtree Q, one can generate a qdag whose completion is Q simply as the pair (Q, M), where M is the *identity mapping* $M[i] = i$, for all $0 \leq i < 2^d$. We can then describe all our operations over qdags. Note, in particular, that we can use mappings to represent any reordering of the attributes.

In terms of representation, the references to quadtree nodes consist of the identifier of the quadtree and the index of the node in level-wise order. This suffices to access the node in constant time from its compact representation. For a qdag $Q' = (Q, M)$, we denote by $|Q'|$ the number of internal nodes in the base quadtree Q, and by $||Q'||$ the number of internal nodes in the completion of Q'.

Algorithm 1 Value.

```
Require: qdag $(Q, M)$ with grid side $\ell$.
Ensure: The integer 1 if the grid is a single point, 0 if the grid is empty, and ½ otherwise.
if $\ell = 1$ then return the integer $Q$
if $Q$ is a leaf then return 0
return ½
```

Algorithm 2 ChildAt.

```
Require: qdag $(Q, M)$ on a grid of dimension $d$ and side $\ell$, and a child number $0 \leq i < 2^d$. Assumes $Q$ is not a leaf or an integer.
Ensure: A qdag $(Q', M)$ corresponding to the $i$-th child of $(Q, M)$.
return $(C(Q, M[i]), M)$
```

Algorithm 3 Extend.

```
Require: A qdag $(Q, M')$ representing a relation $R(\mathcal{A}')$, and a set $\mathcal{A}$ such that $\mathcal{A}' \subseteq \mathcal{A}$.
Ensure: A qdag $(Q, M)$ whose completion represents the relation $R(\mathcal{A}') \times \mathrm{All}(\mathcal{A} \setminus \mathcal{A}')$.
create array $M[0, 2^d - 1]$
$d \leftarrow |\mathcal{A}|$, $d' \leftarrow |\mathcal{A}'|$
for $i \leftarrow 0, \ldots, 2^d - 1$ do
    $m_d \leftarrow$ the $d$-bits binary representation of $i$
    $m_{d'} \leftarrow$ the projection of $m_d$ to the positions in which the attributes of $\mathcal{A}'$ appear in $\mathcal{A}$
    $i' \leftarrow$ the value in $[0, 2^{d'} - 1]$ corresponding to $m_{d'}$
    $M[i] \leftarrow M'[i']$
return $(Q, M)$
```

Algorithms 1 and 2, based on Definition 3.1, will be useful for the navigation of qdags. Operation Value yields a 0 iff the subgrid represented by the qdag is empty (thus the qdag is a leaf); a 1 if the qdag is a full single cell, and ½ if it is an internal node. Operation ChildAt lets us descend by a given child from internal nodes representing nonempty grids. The operations "integer Q", "Q is a leaf", and "$C(Q, j)$" are implemented in constant time on the compact representation of Q.

Operation Extend. We introduce an operation to obtain, from the qdag representing a relation R, a new qdag representing the relation R extended with new attributes.

▶ **Definition 3.2.** *Let $\mathcal{A}' \subseteq \mathcal{A}$ be sets of attributes, let $R(\mathcal{A}')$ be a relation over $\mathcal{A}'$, and let $Q_R = (Q, M)$ be a qdag that represents $R(\mathcal{A}')$. The operation* Extend$(Q_R, \mathcal{A})$ *returns a qdag $Q_R^* = (Q, M')$ that represents the relation $R \times All(\mathcal{A} \setminus \mathcal{A}')$.*

To provide intuition on its implementation, let $\mathcal{A}'$ be the set of attributes $\{A, B, D\}$ and let $\mathcal{A} = \{A, B, C, D\}$, and consider $R(\mathcal{A}')$, Q_R and Q_R^* from Definition 3.2. Each node of Q_R has 8 children, while each node of Q_R^* has 16 children. Consider the child at position $i = 12$ of Q_R^*. This node represents the grid with Morton code m_4=‘**1100**’ (i.e., 12 in binary), and contains the tuples whose coordinates in binary start with **1** in attributes A, B and with **0** in attributes C, D. This child has elements if and only if the child with Morton code m_3=‘**110**’ of Q_R (i.e., its child at position $j = 6$) has elements; this child is in turn the $M[6]$-th child of Q. Note that m_3 results from projecting m_4 to the positions 0,1,3 in which the attributes A, B, D appear in $\{A, B, C, D\}$. Since the Morton code **11**1**0**’ (i.e., 14 in binary) also projects to m_3, it holds that $M'[12] = M'[14] = M[6]$. We provide an implementation of the Extend operation for the general case in Algorithm 3. The following lemma states the time and space complexity of our implementation of Extend. For simplicity, we count the space in terms of computer words used to store references to the quadtrees and values of the mapping function M.

▶ **Lemma 3.3.** *Let $|\mathcal{A}| = d$ in Definition 3.2. Then, the operation* Extend$(Q_R, \mathcal{A})$ *can be supported in time $O(2^d)$ and its output takes $O(2^d)$ words of space.*

Algorithm 4 MultiJoin.

```
Require: Relations $R_1, \dots, R_n$, stored as qdags $Q_1, \dots, Q_n$; each relation $R_i$ is over attributes $\mathcal{A}_i$ and $\mathcal{A} = \bigcup \mathcal{A}_i$.
Ensure: A quadtree representing the output of $J = R_1 \bowtie \dots \bowtie R_n$.

for $i \leftarrow 1, \dots, n$ do
    $Q_i^* \leftarrow$ Extend($R_i, \mathcal{A}$)
return And($Q_1^*, \dots, Q_n^*$)
```

Algorithm 5 And.

```
Require: $n$ qdags $Q_1$, $Q_2$, ..., $Q_n$ representing relations $R_1(\mathcal{A}), R_2(\mathcal{A}), \dots, R_n(\mathcal{A})$.
Ensure: A quadtree representing the relation $\bigcap_{i=1}^n R_i(\mathcal{A})$.

$m \leftarrow \min\{$Value($Q_1$), ..., Value($Q_n$)$\}$
if $\ell = 1$ then return the integer $m$
if $m = 0$ then return a leaf
for $i \leftarrow 0, \dots, 2^d - 1$ do
    $C_i \leftarrow$ And(ChildAt($Q_1, i$), ..., ChildAt($Q_n, i$))
if $\max\{$Value($C_0$), ..., Value($C_{2^d-1}$)$\} = 0$ then return a leaf
return a quadtree with children $C_0, \dots, C_{2^d-1}$
```

Proof. We show that Algorithm 3 meets the conditions of the lemma. The computations of m_d and i' are immaterial (they just interpret a bitvector as a number or vice versa). The computation of m'_d is done with a constant table (that depends only on the database dimension d) of size $O(2^{3d})$:[2] The argument $\mathcal{A}$ is given as a bitvector of size d telling which attributes are in $\mathcal{A}$, the qdag on $\mathcal{A}'$ stores a bitvector of size d telling which attributes are in $\mathcal{A}'$, and the table receives both bitvectors and m_d and returns m'_d. ◀

3.3 Join algorithm

Now that we can efficiently represent relations of the form $R(\mathcal{A}') \times \text{All}(\mathcal{A} \setminus \mathcal{A}')$, for $\mathcal{A}' \subseteq \mathcal{A}$, we describe a worst-case optimal implementation of joins over the qdag representations of the relations. Our algorithm follows the idea discussed for the triangle query: we first extend every qdag to all the attributes that appear in the query, so that they all have the same dimension and attributes. Then we compute their intersection, building a quadtree representing the output of the query. The implementation of this algorithm is surprisingly simple (see Algorithms 4 and 5), yet worst-case optimal, as we prove later on. Using qdags is key for this result; this algorithm would not be at all optimal if computed over relational instances stored using standard representations such as B+ trees. First, we describe how to compute the intersection of several qdags, and then analyze the running time of the join.

Operation And. We introduce an operation And, which computes the intersection of several relations represented as qdags.

▶ **Definition 3.4.** *Let $Q_1, \dots, Q_n$ be qdags representing relations $R_1, \dots, R_n$, all over the attribute set $\mathcal{A}$. Operation* And$(Q_1, \dots, Q_n)$ *returns a quadtree Q that represents the relation $R_1 \cap \dots \cap R_n$.*

We implement this operation by simulating a synchronized traversal among the completions $C_1, \dots, C_n$ of $Q_1, \dots, Q_n$, respectively, obtaining the quadtree Q that stores the cells that are present in all the quadtrees C_i (see Algorithm 5).We proceed as follows. If $\ell = 1$, then all C_i are integers with values 0 or 1, and Q is an integer equal to the minimum of the n values. Otherwise, if any Q_i represents an empty subgrid, then Q is also a leaf representing an empty subgrid. Otherwise, every C_i is rooted by a node v_i with 2^d children, and so is Q, where the j-th child of its root v is the result of the And operation of the j-th children of the

[2] They can be reduced to two tables of size $O(2^{2d})$, but we omit the details for simplicity.

nodes $v_1, \ldots, v_n$. However, we need a final *pruning* step to restore the quadtree invariants (line 6 of Algorithm 5): if $\textsc{Value}(v_i) = 0$ for all the resulting children of v, then v must become a leaf and the children be discarded. Note that once the quadtree is computed, we can represent it succinctly in linear expected time so that, for instance, it can be cached for future queries involving the output represented by Q[3].

Analysis of the algorithm. We compute the output Q of $\textsc{And}(Q_1, \ldots, Q_n)$ in time $O(2^d \cdot (||Q_1|| + \cdots + ||Q_n||))$. More precisely, the time is bounded by $O(2^d \cdot |Q^+|)$, where Q^+ is the quadtree that would result from Algorithm 5 if we remove the pruning step of line 6. We name this quadtree Q^+ as the *non-pruned version* of Q. Although the size of the actual output Q can be much smaller than that of Q^+,we can still prove that our time is optimal in the worst case. We start with a technical result.

▶ **Lemma 3.5.** *The* $\textsc{And}$ *operation can be supported in time $O(M \cdot 2^d n \log \ell)$, where M is the maximum number of nodes in a level of Q^+.*

Proof. We show that Algorithm 5 meets the conditions of the lemma. Let m_j be the number of nodes of depth j in Q^+, and then $M = \max_{0 \le j < \log \ell} m_j$. The number of steps performed by Algorithm 5 is bounded by $n \cdot (\sum_{0 \le j < \log \ell} m_j \cdot 2^d) \le n \cdot M \cdot \log \ell \cdot 2^d$: In each depth we continue traversing all qdags $Q_1, \ldots, Q_n$ as long as they are all nonempty, and we generate the corresponding nodes in Q^+ (even if at the end some nodes will disappear in Q). ◀

All we need to prove (data) optimality is to show that $|Q^+|$ is bounded by the size of the real output of the query. Recall that, for a join query J on a database D, we use $2^{\rho^*(J,D)}$ to denote the AGM bound [2] of the query J over D, that is, the maximum size of the output of J over any relational database having the same number of tuples as D in each relation.

▶ **Theorem 3.6.** *Let $J = R_1 \bowtie \ldots \bowtie R_n$ be a full join query, and D a database over schema $\{R_1, \ldots, R_n\}$, with d attributes in total, and where the domains of the relations are in $[0, \ell-1]$. Let $\mathcal{A}_i$ be the set of attributes of R_i, for all $1 \le i \le n$ and $N = \sum_i |R_i|$ be the total amount of tuples in the database. The relations $R_1, \ldots, R_n$ can then be stored within $\sum_i (|\mathcal{A}_i| + 2 + o(1))|R_i| \log \ell + O(n \log d)$ bits, so that the output for J can be computed in time $O(2^{\rho^*(J,D)} \cdot 2^d n \log \min(\ell, N)) = \tilde{O}(2^{\rho^*(J,D)})$.*

Proof. The space usage is a simple consequence of Observation 2.1. As for the time, to solve the join query J we simply encapsulate the quadtrees representing $R_1, \ldots, R_n$ in qdags $Q_1, \ldots, Q_n$, and use Algorithm 4 to compute the result of the query. We now show that Algorithm 4 runs in time within the bound of the theorem. First, assume that $\log \ell$ is $O(\log N)$. Let each relation R_i be over attributes $\mathcal{A}_i$, and $\mathcal{A} = \bigcup \mathcal{A}_i$ with $d = |\mathcal{A}|$. Let $Q_i^* = \textsc{Extend}(Q_i, \mathcal{A})$, $Q = \textsc{And}(Q_1^*, \ldots, Q_n^*)$, and Q^+ be the non-pruned version of Q. The cost of the $\textsc{Extend}$ operations is only $O(2^d n)$, according to Lemma 3.3, so the main cost owes to the $\textsc{And}$ operation.

If the maximum M of Lemma 3.5 is reached at the lowest level of the decomposition, where we store integers 0 or 1, then we are done: each 1 at a leaf of Q^+ exists in Q as well because that single tuple is present in all the relations $R_1, \ldots, R_n$. Therefore, M is bounded by the AGM bound of J and the time of the $\textsc{And}$ operation is bounded by $O(2^{\rho^*(J,D)} \cdot 2^d n \log \ell)$.

Assume instead that M is the number of internal nodes at depth $0 < j < \log \ell$ of Q^+ (if M is reached at depth 0 then $M = 1$). Intuitively, we will take the relations at the granularity of level j, and show that there exists a database D' where such a $(2^j)^d$ relation arises in the last level and thus the answer has those M tuples.

[3] This consumes linear expected time due to the use of perfect hashing in the compact representation [3].

We then construct the following database D' with relations R'_i: For a binary string c, let $\mathrm{pre}(c, j)$ denote the first j bits of c. Then, for each relation R_i and each tuple $(c_1, \ldots, c_{d_i})$ in R_i, where $d_i = |\mathcal{A}_i|$, let R'_i contain the tuples $(0^{\log \ell - j}\mathrm{pre}(c_1, j), 0^{\log \ell - j}\mathrm{pre}(c_2, j) \ldots, 0^{\log \ell - j}\mathrm{pre}(c_{d_i}, j))$, corresponding to taking the first j bits of each coordinate and prepending them with a string of $\log \ell - j$ 0s. While this operation may send two tuples in a relation in D to a single tuple in D', we still have that each relation R'_i in D' contains at most as many tuples as relation R_i in D. Moreover, if we again store every R'_i as a qdag and process their join as in Algorithm 4, then by construction we have in this case that the leaves of the tree resulting from the AND operation contain exactly M nodes with 1, and that this is the maximum number of nodes in a level of this tree. Since the leaves represent tuples in the answer, we have that $M \leq 2^{\rho^*(J,D')} \leq 2^{\rho^*(J,D)}$, which completes the proof for the case when $\log \ell$ is $O(\log N)$.

Finally, when $\log N$ is $o(\log \ell)$, we can convert $O(\log \ell)$ to $O(\log N)$ in the time complexities by storing $R_1, \ldots, R_n$ using quadtrees, with a slight variation. We store the values of the attributes appearing in any relation in an auxiliary data structure (e.g., an array), and associate an $O(\log N)$-bits identifier to each different value in $[0, \ell - 1]$ that appears in D (e.g., the index of the corresponding value in the array). In this case, we represent the relations in quadtrees, but using the identifiers of the attribute values instead of the values themselves. This representation requires at most $dN \log \ell$ bits for the representation of the distinct attribute values and $O(dN \log N)$ bits for the representation of the quadtrees. Thus, in total it requires $dN \log \ell + O(dN \log N) = dN \log \ell(1 + O(\log N / \log \ell)) = dN \log \ell(1 + o(1))$, which is within the stated bound. Note that in both cases, the height of the quadtrees representing the relations is $O(\log N)$, and this is the multiplying factor in the time complexities. ◀

4 Extending Worst-Case Optimality to More General Queries

In this section we turn to design worst-case optimal algorithms for more expressive queries. At this point it should be clear that we can deal with set operations: we already studied the *intersection* (which corresponds to operation AND over the qdags), and will show that *union* (operation OR) and *complement* (operation NOT) can be solved optimally as well. What is most intriguing, however, is whether we can obtain worst-case optimality on combined relational formulas. We introduce a worst-case optimal algorithm to evaluate formulas expressed as combinations of join, union, and complement operations (which we refer to as JUC-queries; note that intersection is a particular case of join). We do not study other operations like *selection* and *projection* because these are easily solved in time essentially proportional to the size of the output, but refer to Appendix A for more details on how projection interplays with the rest of our framework.

The key ingredient of our algorithm is to deal with these operations in a *lazy* form, allowing unknown intermediate results so that all components of a formula are evaluated simultaneously. To do this we introduce lazy qdags (or lqdags), an alternative to qdags that can navigate over the quadtree representing the output of a formula without the need to entirely evaluate the formula. We then give a worst-case optimal algorithm to compute the *completion* of an lqdag, that is, the quadtree of the grid represented by the lqdag.

4.1 Lqdags for relational formulas

To support worst-case optimal evaluation of relational formulas we introduce two new ideas: we add “full leaves” to the quadtree representation to denote subgrids full of 1s, and we introduce lqdags to represent the result of a formula as an *implicit* quadtree that can be navigated without fully evaluating the formula.

Algorithm 6 Value on extended qdags.

```
Require: qdag $(Q, M)$ with grid side $\ell$.
Ensure: Value 0 or 1 if the grid represented by $Q$ is totally empty or full, respectively, otherwise ½.
if $Q$ is a leaf then return the integer 0 or 1 associated with $Q$
return ½
```

While quadtree leaves representing a single cell store the cell value, 0 or 1, quadtree leaves at higher levels always represent subgrids full of 0s. We now generalize the representation, so that quadtree leaves at any level store an integer, 0 or 1, which is the value of all the cells in the subgrid represented by the leaf. The generalization impacts on the way to compute Value, depicted in Algorithm 6. We will not use qdags in this section, however; the lqdags build directly on quadtrees. In terms of the compact representation, this generalization is implemented by resorting to an impossible quadtree configuration: an internal node with all zero children [5]. Note that replacing a full subgrid with this configuration can only decrease the size of the representation.

The second novelty, the lqdags, are defined as follows.

▶ **Definition 4.1** (lqdag). *An lqdag L is a pair (f, o), where f is a* functor *and o is a list of* operands. *The* completion *of L is the quadtree $Q_R = Q_R(\mathcal{A})$ representing relation $R(\mathcal{A})$ if L is as follows:*

1. $(\mathtt{QTREE}, Q_R)$*, where the lqdag just represents Q_R;*
2. $(\mathtt{NOT}, Q_{\overline{R}})$*, where $Q_{\overline{R}}$ is the quadtree representing the complement of Q_R;*
3. $(\mathtt{AND}, L_1, L_2)$*, where L_1 and L_2 are lqdags and Q_R represents the intersection of their completions;*
4. $(\mathtt{OR}, L_1, L_2)$*, where L_1 and L_2 are lqdags and Q_R represents the union of their completions;*
5. $(\mathtt{EXTEND}, L_1, \mathcal{A})$*, where lqdag L_1 represents $R'(\mathcal{A}')$, $\mathcal{A}' \subseteq \mathcal{A}$, and Q_R represents $R(\mathcal{A}) = R'(\mathcal{A}') \times All(\mathcal{A} \setminus \mathcal{A}')$.*

Note that, for a quadtree Q_R representing a relation $R(\mathcal{A}')$, and a set of attributes $\mathcal{A}$, the qdag $Q_R^* = (Q_R, M_\mathcal{A})$ that represents the relation $R \times \mathrm{All}(\mathcal{A} \setminus \mathcal{A}')$ can be expressed as the lqdag $(\mathtt{EXTEND}, (\mathtt{QTREE}, Q_R), \mathcal{A})$. In this sense, lqdags are extensions of qdags. To further illustrate the definition of lqdags, consider the triangle query $R(A, B) \bowtie S(B, C) \bowtie T(A, C)$, with $\mathcal{A} = \{A, B, C\}$ and the relations represented by quadtrees Q_R, Q_S, and Q_T. This query can then be represented as the lqdag

$$(\mathtt{AND}, (\mathtt{AND}, (\mathtt{EXTEND}, (\mathtt{QTREE}, Q_R), \mathcal{A}), (\mathtt{EXTEND}, (\mathtt{QTREE}, Q_S), \mathcal{A})), (\mathtt{EXTEND}, (\mathtt{QTREE}, Q_T), \mathcal{A})).$$

It is apparent that one can define other operations, like JOIN and DIFF, by combining the operations defined above:

$$\begin{aligned}(\mathtt{JOIN}, L_1(\mathcal{A}_1), L_2(\mathcal{A}_2)) &= (\mathtt{AND}, (\mathtt{EXTEND}, L_1, \mathcal{A}_1 \cup \mathcal{A}_2), (\mathtt{EXTEND}, L_2, \mathcal{A}_1 \cup \mathcal{A}_2))\\ (\mathtt{DIFF}, L_1, L_2) &= (\mathtt{AND}, L_1, (\mathtt{NOT}, L_2))\end{aligned}$$

Note that in the definition of the lqdag for NOT, the operand is a quadtree instead of an lqdag, and then, for example, L_2 should be a quadtree in the definition of DIFF, in principle. However, we can easily get around that restriction by pushing down the NOT operators until the operand is a quadtree or the NOT is cancelled with another NOT. For instance, a NOT over an lqdag $(\mathtt{AND}, Q_1, Q_2)$ is equivalent to $(\mathtt{OR}, (\mathtt{NOT}, Q_1), (\mathtt{NOT}, Q_2))$, and analogously with the other functors. The restriction, however, does limit the types of formulas for which we achieve worst-case optimality, as shown later.

Algorithm 7 Value function for NOT.

```
Require: A Quadtree $Q$.
Ensure: Value of the root of $(\texttt{NOT}, Q)$.
return $1 - \textsc{Value}(Q)$
```

Algorithm 8 ChildAt function for NOT.

```
Require: A Quadtree $Q$ in dimension $d$, and an integer $0 \leq i < 2^d$.
Ensure: An lqdag for the $i$-th child of $(\texttt{NOT}, Q)$.
return $(\texttt{NOT}, \textsc{ChildAt}(Q, i))$
```

Algorithm 9 Value function for AND.

```
Require: Lqdags $L_1$ and $L_2$.
Ensure: The value of the root of $(\texttt{AND}, L_1, L_2)$.
if $\textsc{Value}(L_1) = 0$ or $\textsc{Value}(L_2) = 0$ then return 0
if $\textsc{Value}(L_1) = 1$ then return $\textsc{Value}(L_2)$
if $\textsc{Value}(L_2) = 1$ then return $\textsc{Value}(L_1)$
return $\Diamond$
```

Algorithm 10 ChildAt function for AND.

```
Require: Lqdags $L_1$ and $L_2$ in dimension $d$, integer $0 \leq i < 2^d$.
Ensure: An lqdag for the $i$-th child of $(\texttt{AND}, L_1, L_2)$.
if $\textsc{Value}(L_1) = 1$ then return $\textsc{ChildAt}(L_2, i)$
if $\textsc{Value}(L_2) = 1$ then return $\textsc{ChildAt}(L_1, i)$
return $(\texttt{AND}, \textsc{ChildAt}(L_1, i), \textsc{ChildAt}(L_2, i))$
```

To understand why we call lqdags lazy, consider the operation Q_1 And Q_2 over quadtrees Q_1, Q_2. If any of the values at the roots of Q_1 or Q_2 is 0, then the result of the operation is for sure a leaf with value 0. If any of the values is 1, then the result of the operation is the other. However, if both values are ½, one cannot be sure of the value of the root until the And between the children of Q_1 and Q_2 has been evaluated. Solving this dependency eagerly would go against worst-case optimality: it forces us to fully evaluate parts of the formula without considering it as a whole. To avoid this, we allow the Value of a node represented by an lqdag to be, apart from 0, 1, and ½, the special value $\Diamond$. This indicates that one cannot determine the value of the node without computing the values of its children.

As we did for qdags, in order to simulate the navigation over the completion Q of an lqdag L we need to describe how to obtain the value of the root of Q, and how to obtain an lqdag whose completion is the i-th child of Q, for any given i. We implement those operations in Algorithms 7–14, all constant-time. Note that ChildAt can only be invoked when Value = ½ or $\Diamond$. The base case is $\textsc{Value}(\texttt{QTREE}, Q) = \textsc{Value}(Q)$ and $\textsc{ChildAt}((\texttt{QTREE}, Q), i) = \textsc{ChildAt}(Q, i)$, where we enter the quadtree and resort to the algorithms based on the compact representation of Q. We will assume that $\textsc{Value}(Q)$ returns ½ for internal nodes, and thus the implementation of Value for EXTEND is trivial (compare Algorithms 6 and 13 under this assumption).

Note that the recursive calls of Algorithms 7-14 traverse the nodes of the relational formula (fnodes, for short), and terminate immediately upon reaching an fnode of the form $(\texttt{QTREE}, Q)$. Therefore, their time complexity depends only on the size of the formula represented by the lqdag. We show next how, using these implementations of Value and ChildAt, one can efficiently evaluate a relational formula using lqdags.

4.2 Evaluating JUC queries

To evaluate a formula F represented as an lqdag L_F, we compute the completion Q_F of L_F, that is, the quadtree Q_F representing the output of F.

To implement this we introduce the idea of *super-completion* of an lqdag. The super-completion Q_F^+ of L_F is the quadtree induced by navigating L_F, and interpreting the values $\Diamond$ as ½ (see Algorithm 15). Note that, by interpreting values $\Diamond$ as ½, we are disregarding the possibility of pruning resulting subgrids full of 0s or 1s and replacing them by single leaves with values 0 or 1 in Q_F. Therefore, Q_F^+ is a *non-pruned* quadtree (just as Q^+ in

Algorithm 11 VALUE function for `OR`.

```
Require: Lqdags $L_1$ and $L_2$.
Ensure: The value of the root of $(\texttt{OR}, L_1, L_2)$.
if $\textsc{Value}(L_1) = 1$ or $\textsc{Value}(L_2) = 1$ then
    return 1
if $\textsc{Value}(L_1) = 0$ then return $\textsc{Value}(L_2)$
if $\textsc{Value}(L_2) = 0$ then return $\textsc{Value}(L_1)$
return $\lozenge$
```

Algorithm 13 VALUE function for `EXTEND`.

```
Require:
    Lqdag $L_1(\mathcal{A}')$, set $\mathcal{A} \supseteq \mathcal{A}'$.
Ensure:
    Value of the root of $(\texttt{EXTEND}, L_1, \mathcal{A})$.
return $\textsc{Value}(L_1)$
```

Algorithm 12 CHILDAT function for `OR`.

```
Require: Lqdags $L_1$ and $L_2$ in dimension $d$, integer
    $0 \leq i < 2^d$.
Ensure: An lqdag for the $i$-th child of $(\texttt{OR}, L_1, L_2)$.
if $\textsc{Value}(L_1) = 0$ then return $\textsc{ChildAt}(L_2, i)$
if $\textsc{Value}(L_2) = 0$ then return $\textsc{ChildAt}(L_1, i)$
return $(\texttt{OR}, \textsc{ChildAt}(L_1, i), \textsc{ChildAt}(L_2, i))$
```

Algorithm 14 CHILDAT function for `EXTEND`.

```
Require:
    Lqdag $L_1(\mathcal{A}')$, set $\mathcal{A} \supseteq \mathcal{A}'$, integer $0 \leq i < 2^{|\mathcal{A}|}$.
Ensure: An lqdag for the $i$-th child of $(\texttt{EXTEND}, L_1, \mathcal{A})$.
$d \leftarrow |\mathcal{A}|$, $d' \leftarrow |\mathcal{A}'|$
$m_d \leftarrow$ the $d$-bits binary representation of $i$
$m_{d'} \leftarrow$ the projection of $m_d$ to the positions in
    which the attributes of $\mathcal{A}'$ appear in $\mathcal{A}$
$i' \leftarrow$ the value in $[0, 2^{d'} - 1]$ corresponding to $m_{d'}$
return $(\texttt{EXTEND}, \textsc{ChildAt}(L_1, i'), \mathcal{A})$
```

Section 3.3) that nevertheless represents the same points of Q_F. Moreover, Q_F^+ shares with Q_F a key property: all its nodes with value 1, including the last-level leaves representing individual cells, correspond to actual tuples in the output of F.

To see how lqdags are evaluated, let us consider the query $F = R(A, B) \bowtie S(B, C) \bowtie \overline{T}(A, C)$. This corresponds to an lqdag Q_F:

$$(\texttt{AND}, (\texttt{AND}, (\texttt{EXTEND}, (\texttt{QTREE}, Q_R), \mathcal{A}), (\texttt{EXTEND}, (\texttt{QTREE}, Q_S), \mathcal{A})), (\texttt{EXTEND}, (\texttt{NOT}, Q_T), \mathcal{A})).$$

Assuming some of the trees involved have internal nodes, the super-completion Q_F^+ first produces 8 children. Suppose the grid of T is full of 1s in the first quadrant (00). Then the first child (00) of Q_T has value 1, which becomes value 0 in $(\texttt{NOT}, Q_T)$. This implies that $(\texttt{EXTEND}, (\texttt{NOT}, Q_T))$ also yields value 0 in octants 000 and 010. Thus, when function CHILDAT is called on child 000 of Q_F, our 0 is immediately propagated and CHILDAT returns 0, meaning that there are no answers for F on this octant, without ever consulting the quadtrees Q_R and Q_S (see Figure 3 for an illustration). On the other hand, if the value of the child 11 of T is 0, then $(\texttt{EXTEND}, (\texttt{NOT}, Q_T))$ will return value 1 in octants 101 and 111. This means that the result on this octant corresponds to the result of joining R and S; indeed CHILDAT towards 101 in Q_F returns

$$(\texttt{AND}, \textsc{ChildAt}((\texttt{EXTEND}, (\texttt{QTREE}, Q_R), \mathcal{A}), 101), \textsc{ChildAt}((\texttt{EXTEND}, (\texttt{QTREE}, Q_S), \mathcal{A}), 101)).$$

If $\textsc{ChildAt}((\texttt{EXTEND}, (\texttt{QTREE}, Q_R), \mathcal{A}), 101)$ and $\textsc{ChildAt}((\texttt{EXTEND}, (\texttt{QTREE}, Q_S), \mathcal{A}), 101)$ are trees with internal nodes, the resulting `AND` can be either an internal node or a leaf with value 0 (if the intersection is empty), though not a leaf with value 1. Thus, for now, the VALUE of this node is unknown, a $\lozenge$. See Figure 3 for an illustration.

Note that the running time of Algorithm 15 is $O(|Q_F^+|)$. One can then compact Q_F^+ to obtain Q_F, in time $O(|Q_F^+|)$ as well, with a simple bottom-up traversal. Thus, bounding $|Q_F^+|$ yields a bound for the running time of evaluating F. While $|Q_F^+|$ can be considerably larger than the actual size $|Q_F|$ of the output, we show that $|Q_F^+|$ is bounded by the worst-case output size of formula F for a database with relations of approximately the same size. To prove this, the introduction of values $\lozenge$ plays a key role.[4]

[4] In an implementation, we could simply use ½ instead of $\lozenge$, without indicating that we are not yet sure

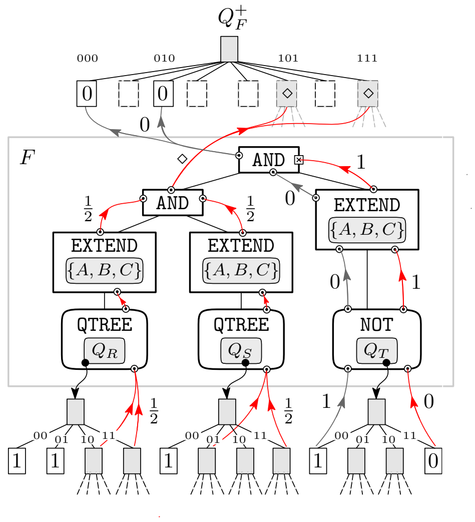


Figure 3 Illustration of the syntax tree of an lqdag for the formula $(R(A,B) \bowtie S(B,C)) \bowtie \overline{T}(A,C)$. The quadtrees Q_R, Q_S, Q_T represent the relations R, S, T, respectively. We show the top values of Q_F^+ on top and of Q_T on the bottom. The gray upward arrows show how the value 1 in the quadrant 00 of Q_T becomes 0s in octants 000 and 010 of Q_F^+ without accessing Q_R or Q_S. The red upward arrows show how the value 0 in the quadrant 11 of Q_T makes the quadrants 101 and 111 of Q_F^+ depend only on their left child (and, assuming their value is ½, becomes a ◇ in Q_F^+).

Algorithm 15 SCompletion.

```
Require: An lqdag $L_F$ whose completion represents a formula $F$ over relations with $d$ attributes.
Ensure: The super-completion $Q_F^+$ of $L_F$.
if Value($L_F$) $\in \{0,1\}$ then return a leaf with value Value($L_F$)
return an internal node with children
    $\Big($SCompletion$\big($ChildAt$(L_F, 0)\big), \ldots,$ SCompletion$\big($ChildAt$(L_F, 2^d - 1)\big)\Big)$
```

The power of the ◇ values. Consider again Algorithm 15. The lowest places in L_F where ◇ values are introduced are the Value of AND and OR lqdags where both operands have Value = ½. We must then set the Value to ◇ instead of ½ because, depending on the evaluation of the children of the operands, the Value can turn out to be actually 0 for AND or 1 for OR. Once produced, a value ◇ is inherited by the ancestors in the formula unless the other value is 0 (for AND) or 1 (for OR).

Imagine that a formula F involves n relations $R_1, \ldots, R_n$ represented as quadtrees in dimension d, including no negations. Suppose that we *trim* the quadtrees of $R_1, \ldots, R_n$ by removing all the levels at depth higher than some j (thus making the j-th level the last one) and assuming that the internal nodes at level j become leaves with value 1. We do not attempt to compact the resulting quadtrees, so their nodes at levels up to $j-1$ stay identical and with the same Value. If we now compute Q_F^+ over those (possibly non-pruned) quadtrees, the computation will be identical up to level $j-1$, and in level j every internal node in the original Q_F^+, which had value ½, will now operate over all 1s, and thus will evaluate to 1 because And and Or are monotonic.

that the value is ½: we build Q_F^+ assuming it is, and only make sure later, when we compact it into Q_F.

Thus, these nodes belong to the output of F over the relations $R'_1, \ldots, R'_n$ induced by the trimmed quadtrees (on smaller domains of size $\ell' = 2^j$), with sizes $|R_1|' \leq |R_1|, \ldots, |R_n|' \leq |R_n|$. This would imply, just as in the proof of Theorem 3.6, a bound on the maximum number of nodes in a level of Q_F^+, thus proving the worst-case optimality of the size of Q_F^+ (up to $\log min(\ell, N)$-factors, and factors depending on the query size), and thus the worst-case optimality of Algorithm 15 in data complexity.

However, this reasoning fails when one trims at the j-th level a quadtree Q that appears in an lqdag $L = (\texttt{NOT}, Q)$, because the value 1 of the nodes at the j-th level of Q after the trimming changes to 0 in L. So, to prove that our algorithm is worst-case optimal we cannot rely only on relations obtained by trimming those that appear in the formula. We need to generate new quadtrees for those relations under a NOT operation that preserve the values of the completion of NOT after the trimming. Next we formalize how to do this.

Analysis of the algorithm. Let L_F be an lqdag for a formula F. The *syntax tree* of F is the directed tree formed by the fnodes in F, with an edge from fnode L to fnode L' if L' is an operand of L. The leaves of this tree are always *atomic expressions*, that is, the fnodes, with functors QTREE and NOT, that operate on one quadtree (see Figure 3 again). We say that two atomic expressions L_1 and L_2 are equal if both their functors and operands are equal. For example, in the formula

$$F = (\texttt{OR}, (\texttt{AND}, (\texttt{QTREE}, Q_R), (\texttt{QTREE}, Q_S)), (\texttt{AND}, (\texttt{QTREE}, Q_R), (\texttt{QTREE}, Q_T)))$$

there are three different atomic expressions, $(\texttt{QTREE}, Q_R)$, $(\texttt{QTREE}, Q_S)$, and $(\texttt{QTREE}, Q_T)$, while in $F' = (\texttt{AND}, (\texttt{QTREE}, Q_R), (\texttt{NOT}, Q_R))$ there are two atomic expressions. Notice that in formulas like F', where a relation appears both negated and not negated, the two occurrences are seen as different atomic expressions. We return later to the consequences of this definition.

The following lemma is key to bound the running time of Algorithm 15 while evaluating a formula F.

▶ **Lemma 4.2.** *Let F be a relational formula represented by an lqdag L_F in dimension d, and let Q_F^+ be the super-completion of F. Let $Q_1, \ldots, Q_n$ be the quadtree operands of the different atomic expressions of F, and $R_1(\mathcal{A}_1), \ldots, R_n(\mathcal{A}_n)$ be the (not necessarily different) relations represented by these quadtrees, respectively. Let M be the maximum number of nodes in a level of Q_F^+. Then, there is a database with relations $R'_1(\mathcal{A}_1), \ldots, R'_n(\mathcal{A}_n)$ of respective sizes $O(2^d|Q_1|), \ldots, O(2^d|Q_n|)$, such that the output of F evaluated over it has size $\Omega(M/2^d)$.*

Proof. Let m_l be the number of nodes in level l of Q_F^+ and j be a level where $M = m_j$ is maximum. We assume that $j > 1$, otherwise $M = O(1)$ and the result is trivial. We first bound the number of nodes with value ½ at the $(j-1)$-th level. By hypothesis, $m_j \geq m_{j-1}$, and since a node in Q_F^+ is present at level j only if its parent at level $j-1$ has value ½, in the $(j-1)$-th level there are at least $m_j/2^d$ nodes with value ½.

Now, let $A_1, \ldots, A_n$ be the atomic expressions of F, and let $Q'_1, \ldots, Q'_n$ be the quadtrees that result from trimming the levels at depths higher than $j-1$ from $Q_1, \ldots, Q_n$, respectively. Consider the completion A_i^* of A_i evaluated over Q_i, and the completion $A_i^{*\prime}$ of A_i evaluated over (the possibly non-pruned) Q'_i, for all $1 \leq i \leq n$. If it is always the case that the first $j-1$ levels of A_i^* are respectively equal to the $j-1$ levels of $A_i^{*\prime}$ then we are done. To see why, let $Q_F^{+\prime}$ be the super-completion of F when evaluated over $Q'_1, \ldots, Q'_n$. The first $j-2$ levels of Q_F^+ will be the same as those of $Q_F^{+\prime}$ because the same results of the operations are propagated up from the leaves of the syntax tree of F before and after the trimming.

Moreover, in the $(j-1)$-th level $Q_F^{+\prime}$ (its last level) the nodes with value 1 are precisely the nodes with value 1 or ½ in Q_F^+, where we note that: (i) there are at least $m_j/2^d$ of them; and (2) they belong to the output of F over the relations $R_1', \ldots, R_n'$ represented by $Q_1', \ldots, Q_n'$.

We know that $|R_1'| \leq |R_1|, \ldots, |R_n'| \leq |R_n|$. However, the values of $R_1', \ldots, R_n'$ correspond to a smaller universe. This can be remedied by simply appending $(\log \ell - j)$ 0's at the beginning of the binary representation of these values. This would yield the desired result: we have n relations over the same set of attributes as the original ones, with same respective cardinality, and such that when F is evaluated over them the output size is $\Omega(m_j/2^d)$.

However, for atomic expressions of the type $A_i = (\mathtt{NOT}, Q_i)$ it is not the case that the first $j-1$ levels of A_i^* coincide with the $j-1$ levels of $A_i^{*\prime}$. Anyway, we can deal with this case: their first $j-2$ levels will coincide, and in the last level, the value of a node present in A_i^* is the negation of the value of the homologous node in $A_i^{*\prime}$. Thus, instead of choosing the quadtree Q_i' that results from trimming Q_i, we choose the quadtree Q_i'' in which the first $j-2$ levels are the same as Q_i', and the $(j-1)$-th level results from negating the value of every node in Q_i'. Note that if we let now $A_i^{*\prime}$ be the completion of A_i evaluated over Q_i'', then the first $j-1$ levels of A_i^* will be exactly same as the $j-1$ levels of $A_i^{*\prime}$. Finally, note that the size of the relation represented by Q_i'' cannot be larger than $2^d|Q_i|$. The result of the lemma follows. ◀

Using the same reasoning as before we can bound the time needed to compute the super-completion Q_F^+ of an lqdag L_F in dimension d involving quadtrees representing $R_1, \ldots, R_n$. Since M is the maximum number of nodes in a level of Q_F^+, the number of nodes in Q_F^+ is at most $M \log \ell$. Now, each node in Q_F^+ results from the application of $|F|$ operations on each of the 2^d children being generated, all of which take constant time. Thus the super-completion can be computed in time $O(M \cdot 2^d |F| \log \ell)$. If we use $F(D)^*$ to denote the size of the maximum output of the query F over instances with relations $R_1, \ldots, R_n$ of respective sizes $O(2^d|Q_1|), \ldots, O(2^d|Q_n|)$, then by Lemma 4.2 the query F can be computed in time $O(F(D)^* \cdot 2^{2d}|F| \log \ell)$. This means that the algorithm is indeed worst-case optimal.

The requirement of different atomic expressions is because we need to consider R and Not R as different relations. To see this, consider again our example formula $F' = (\mathtt{AND}, (\mathtt{QTREE}, Q_R), (\mathtt{NOT}, Q_R))$. We clearly have that the answer of this query is always empty, and therefore $|Q_{F'}| = 0$. However, here $|Q_{F'}^+| = \Theta(|R|)$ for every R, and thus our algorithm is worst-case optimal only if we consider the possible output size of a more general formula, $F'' = (\mathtt{AND}, (\mathtt{QTREE}, Q_R), (\mathtt{NOT}, Q_R'))$. This impacts in other operations of the relational algebra. We can write all of them as lqdags, but for some of them we will not ensure their optimal evaluation. For instance, the expression Q_R And (Not (Q_R And Q_S)), which expresses the antijoin between R and S, is not optimal because both Q_R and Not Q_R appear in the formula. A way to ensure that our result applies is to require that the atomic expressions (once the NOT operations are pushed down) refer all to different relations.

▶ **Theorem 4.3.** *Let F be a relational formula represented by an lqdag L_F. If the number of different relations involved in F equals the number of different atomic expressions, then Algorithm 15 evaluates F in worst-case optimal time in data complexity.*

Note that this result generalizes Theorem 3.6. Moreover, it does not matter how we write our formula F to achieve worst-case optimal evaluation. For example, our algorithms behave identically on $((R \bowtie S) \bowtie T)$ and on $(R \bowtie (S \bowtie T))$.

5 Final Remarks

The evaluation of join queries using qdags provides a competitive alternative to current worst-case optimal algorithms [9, 11, 15, 17, 21]. When compared to them, we find the following time-space tradeoffs.

Regarding space, qdags require only just a few extra words per tuple, which is generally much less than what standard database indexes require, and definitely less than those required by current worst-case optimal algorithms (e.g. [9, 17, 21]). Moreover, in both NPRR [17] and leapfrog [21], the required index structure only works for a specific ordering of the attributes. Thus, in order to efficiently evaluate any possible query using these two algorithms, a separate index is required for every possible attribute order (i.e., $d!$ indexes). In contrast, all we need to store is one quadtree per relation, and that works for any query. Even if we resort to the (simpler) k^d-tree representation by Brisaboa et al. [4], the extra space increases by a factor of 2^d bits, which is still considerably less than the alternative of $d!$ standard indexes for any order of variables (e.g., for $d = 10$, $2^d = 1024$, while $d! = 3628800$, i.e., ≈ 3500 times bigger).

Regarding time, the first comparison that stands aside is the $\log(N)$ factor, present in our solution but not in others like NPRR [17] and leapfrog [21]. Note, however, that NPRR assumes to be able to compute a join of two relations R and S in time $O(|R|+|S|+|R \bowtie S|)$, which is only possible when using a hash table and when time is computed in an amortized way or in expectation [17, footnote 3]. This was also noted for leapfrog [21, Section 5], where they state that their own $\log(N)$ factor can be avoided by using hashes instead of tries, but they leave open whether this is actually better in practice. More involved algorithms such as PANDA [11] build upon algorithms to compute joins of two relations, and therefore the same $\log(N)$ factor appears if one avoids hashes or amortized running time bounds. Our algorithm incurs in an additional 2^d factor in time when compared to NPRR or leapfrog, similarly to other worst-case optimal solutions based on geometric data structures [9, 15]. This is, as far as we are aware, unavoidable: it is the price to pay for using so little space. Note, however, that this factor does not depend on the data, and that it can be compensated by the fact that our native indexes are compressed, and thus might fit entirely in faster memory.

One important benefit of our framework is that answers to queries can be delivered in their compact representation. As such, we can iterate over them, or store them, or use them as materialized views, either built eagerly, as quadtrees, or in lazy form, as lqdags. One could even cache the top half of the (uncompacted) tree containing the answer, and leave the bottom half in the form of lqdags. The upper half, which is used the most, is cached, and the bottom half is computed on demand. Our framework also permits sharing lqdags as common subexpressions that are computed only once. Additionally, we envision two main uses for the techniques presented in this paper. On one hand, one could take advantage of the low storage cost of these indexes, and add them as a companion to a more traditional database setting. Smaller joins and selections could be handled by the database, while multijoins could be processed faster because they would be computed over the quadtrees. On the other hand, we could use lqdags instead, so as to evaluate more expressive queries over quadtrees. Even if some operations are not optimal, what is lost in optimality may be gained again because these data structures allow operating in faster memory levels.

There are several directions for future work. For instance, we are trying to improve our structures to achieve good bounds for acyclic queries (see Appendix A), and we see an opportunity to apply quadtrees in the setting of parallel computation (see, e.g., Suciu [20]). We also comment in Appendix A on bounds for clustered databases, another topic deserving further study.

References

1 S. Álvarez-García, N. Brisaboa, J. Fernández, M. Martínez-Prieto, and G. Navarro. Compressed vertical partitioning for efficient RDF management. *Knowledge and Information Systems*, 44(2):439–474, 2015.

2 A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

3 D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing Trees of Higher Degree. *Algorithmica*, 43(4):275–292, 2005.

4 N. R. Brisaboa, S. Ladra, and G. Navarro. Compact representation of Web graphs with extended functionality. *Information Systems*, 39(1):152–174, 2014.

5 G. de Bernardo, S. Alvarez-García, N. Brisaboa, G. Navarro, and O. Pedreira. Compact Querieable Representations of Raster Data. In *Proc. 20th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 96–108, 2013", !SERIES = "LNCS 8214.

6 R. A. Finkel and J. L. Bentley. Quad Trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

7 T. Gagie, J. González-Nova, S. Ladra, G. Navarro, and D. Seco. Faster compressed quadtrees. In *Proc. 25th Data Compression Conference (DCC)*, pages 93–102, 2015.

8 A. Hogan, C. Riveros, C. Rojas, and A. Soto. Extending SPARQL engines with multiway joins. In *Proc. 18th International Semantic Web Conference (ISWC)*, 2019. To appear.

9 M. A. Khamis, H. Q. Ngo, C. Ré, and A. Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems*, 41(4):22, 2016.

10 M. A. Khamis, H. Q. Ngo, and A. Rudra. FAQ: Questions asked frequently. In *Proc. 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 13–28, 2016.

11 M. A. Khamis, H. Q. Ngo, and D. Suciu. What do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog have to do with one another? In *Proc. 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 429–444, 2017.

12 G. M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. Technical report, IBM Ltd., 1966.

13 G. Navarro. *Compact Data Structures – A practical approach.* Cambridge University Press, 2016.

14 H. Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proc. 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 111–124, 2018.

15 H. Q. Ngo, D. T. Nguyen, C. Re, and A. Rudra. Beyond worst-case analysis for joins with Minesweeper. In *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 234–245, 2014.

16 H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. *ACM SIGMOD Record*, 42(4):5–16, 2014.

17 Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 37–48. ACM, 2012.

18 D. Nguyen, M. Aref, M. Bravenboer, G. Kollias, H. Q. Ngo, C. Ré, and A. Rudra. Join processing for graph patterns: An old dog with new tricks. In *Proc. 3rd International Workshop on Graph Data Management Experiences and Systems (GRADES)*, pages 2:1–2:8, 2015.

19 H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann, 2006.

20 D. Suciu. Communication cost in parallel query evaluation: A tutorial. In *Proc. 36th ACM Symposium on Principles of Database Systems (PODS)*, pages 319–319, 2017.

21 T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proc. 17th International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

22 D. S. Wise and J. Franco. Costs of quadtree representation of nondense matrices. *Journal of Parallel and Distributed Computing*, 9(3):282–296, 1990.

23 M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. 7th International Conference on Very Large Databases (VLDB)*, pages 82–94, 1981.

A Appendix

A.1 Additional comments on Projections

Including projection in our framework is not difficult: in a quadtree Q storing a relation R with attributes $\mathcal{A}$, one can compute the projection $\pi_{\mathcal{A}'}(R)$, for $\mathcal{A}' \subseteq \mathcal{A}$ as follows. Assume that $|\mathcal{A}| = d$ and $|\mathcal{A}'| = d'$. Then the projection is the quadtree defined inductively as follows. If $\textsc{Value}(Q)$ is 0 or 1 then the projection is a leaf with the same value. Otherwise Q has 2^d children. The quadtree for $\pi_{\mathcal{A}'}(R)$ has instead $2^{d'}$ children, where the i-th child is defined as the $\textsc{Or}$ of all children j of Q such that the projection of the d-bit representation of j to the positions in which attributes in $\mathcal{A}'$ appear in $\mathcal{A}$ is precisely the d'-bit representation of i. For example, computing $\pi_{A_1,A_2}R(A_1, A_2, A_3)$ means creating a tree with four children, resulting of the $\textsc{Or}$ of children 0 and 1, 2 and 3, 4 and 5 and 6 and 7, respectively.

Having defined the projection, a natural question is whether one can use it to obtain finer bounds for acyclic queries or for queries with bounded treewidth. For example, even though the AGM bound for $R(A, B) \bowtie S(B, C)$ is quadratic, one can use Yannakakis' algorithm [23] to compute it in time $O(|R| + |S| + |R \bowtie S|)$. This is commonly achieved by first computing $\pi_B(R)$ and $\pi_B(S)$, joining them, and then using this join to filter out R and S. Unfortunately, adopting this strategy in our lqdag framework would still give us a quadratic algorithm, even for queries with small output, because after the projection we would need to extend the result again. The same holds for the general Yannakakis' algorithm when computing the final join after performing all necessary semijoins.

More generally, this also rules out the possibility to achieve optimal bounds for queries with bounded treewidth or similar measures. Of course, this is not much of a limitation because one can always compute the most complex queries with our compact representation and then carry out Yannakakis' algorithms on top of these results with standard database techniques, but it would be better to resolve all within our framework. We are currently looking at improving our data structures in this regard.

A.2 Geometric representation and finer analysis

As quadtrees have a direct geometric interpretation, it is natural to compare them to the algorithm based on *gap boxes* proposed by Khamis et al. [9]. In a nutshell, this algorithm uses a data structure that stores relations as a set of multidimensional cubes that contain no data points, which the authors call gap boxes. Under this framework, a data point is in the answer of the join query $R_1 \bowtie \cdots \bowtie R_n$ if the point is not part of a gap box in any of the relations R_i. The authors then compute the answers of these queries using an algorithm that finds and merges appropriate gap boxes covering all cells not in the answer of the query, until no more gap boxes can be found and we are left with a covering that misses exactly those points in the answer of the query. Perhaps more interestingly, the algorithm is subject of a finer analysis: the runtime of queries can be shown to be bounded by a function of the size of a *certificate* of the instance (and not its size). The certificate in their case is simply the minimum amount of gap boxes from the input relations that is needed to cover all the gaps in the answer of the query. Finding such a minimal cover is NP-hard, but a slightly restricted notion of gap boxes maintains the bounds within an $O(\log^d \ell)$ approximation factor.

While any index structure can be thought of as providing a set of gap boxes [9], quadtrees provide a particularly natural and compact representation. Each node valued 0 in a quadtree signals that there are no points in its subgrid, and can therefore be understood as a d-dimensional gap box. We can understand qdags as a set of gap boxes as well: precisely those in its completion. Now let $J = R_1 \bowtie \cdots \bowtie R_n$ be a join query over d attributes, and let $R_1^*, \ldots, R_n^*$ denote the extension of each R_i with the attributes of J that are not in R_i. As in Khamis et al. [9], a *quadtree certificate* for J is a set of gap boxes (i.e., empty d-dimensional grids obtained from any of the R_i^*s) such that every coordinate not in the answer of J is covered by at least one of these boxes. Let $C_{J,D}$ denote a certificate for J of minimum size.

▶ **Proposition A.1.** *Given query J and database D, Algorithm 4 runs in time $O((|C_{J,D}| + |J(D)|) \cdot 2^d n \log \ell)$, where $J(D)$ is the output of the query J over D.*

Now, one can easily construct instances and queries such that the minimal certificate $C_{J,D}$ is comparable to $2^{\rho^*(J,D)}$. So this will not give us optimality results, as discovered [9, 15] for acyclic queries or queries with bounded treewidth. This is a consequence of increasing the dimensionality of the relations. Nevertheless, the bound does yield a good running time when we know that $C_{J,D}$ is small. It is also worth mentioning that our algorithms directly computes the only possible representation of the output as gap boxes (because its boxes come directly from the representation of the relations). This means that there is a direct connection between instances that give small certificates and instances for which the representation of the output is small.

A.3 Better runtime on clustered databases

Quadtrees have been shown to work well in applications such as RDF stores or web graphs, where data points are distributed in clusters [4, 1]. It turns out that combining the analysis described in Section 2 for clustered grids with the technique we used to show that joins are worst-case optimal, results in a better bound for the running time of our algorithms, and a small refinement of the AGM bound itself.

Consider again the triangle query $R(A, B) \bowtie S(B, C) \bowtie T(A, C)$, and assume the points in each relation are distributed in c clusters, each of them of size at most $s \times s$, and with p points in total. Then, at depth $\log(\ell/s)$, the quadtrees of T, R, and S have at most 2^d internal nodes per cluster (where we are in dimension $d = 3$): at this level one can think of the trimmed quadtree as representing a coarser grid of cells of size s^d, and therefore each cluster can intersect at most two of these coarser cells per dimension. Thus, letting Q'_R, Q'_S, and Q'_T be the quadtrees for R, S and T trimmed up to level $\log(\ell/s)$ (and where internal nodes take value 1), then the proof of Theorem 3.6 yields a bound for the number of internal nodes at level $\log(\ell/s)$ of the quadtree Q^+ of the output before the compaction step (or, equivalently, of the super-completion of the lqdag of the triangle query): this number must be bounded by the AGM bound of the instances given by Q'_R, Q'_S and Q'_T, which is at most $(c \cdot 2^d)^{3/2}$. Going back to the data for the quadtree Q^+, the bound on the number of internal nodes means that the points of the output are distributed in at most $(c \cdot 2^d)^{3/2}$ clusters of size at most s^d. In turn, the maximal number of 1s in the answer is bounded by the AGM bound itself, which here is $p^{3/2}$. This means that the size of Q^+ is bounded by $O((c \cdot 2^d)^{3/2} \log \ell + p^{3/2} \log s)$, and therefore the running time of the algorithm is $O\big(((c \cdot 2^d)^{3/2} \log \ell + p^{3/2} \log s) \cdot 2^d\big)$. This is an important reduction in running time if the number c of clusters and their width s are small, as we now multiply the number of answers by $\log s$ instead of $\log \ell$.

To generalize, let us use $D^{c,d}$ as the database "trimmed" to $c \cdot 2^d$ points. The discussion above can be extended to prove the following.

▶ **Proposition A.2.** *Let $J = R_1 \bowtie \cdots \bowtie R_n$ be a full join query, and D a database over schema $\{R_1, \ldots, R_n\}$, with d attributes in total, where the domains of the relations are in $[0, \ell - 1]$, and where the points in each relation are distributed in c clusters of width s. Then Algorithm 4 works in time $O\big((2^{\rho^*(J,D^{c,d})} \log \ell + 2^{\rho^*(J,D)} \log s) \cdot 2^d n\big)$.*

The Space Complexity of Inner Product Filters

Rasmus Pagh
IT University of Copenhagen, Denmark
BARC, Copenhagen, Denmark
pagh@itu.dk

Johan Sivertsen
IT University of Copenhagen, Denmark
johanvts@gmail.com

Abstract

Motivated by the problem of filtering candidate pairs in inner product similarity joins we study the following *inner product estimation* problem: Given parameters $d \in \mathbb{N}$, $\alpha > \beta \geq 0$ and unit vectors $x, y \in \mathbb{R}^d$ consider the task of distinguishing between the cases $\langle x, y\rangle \leq \beta$ and $\langle x, y\rangle \geq \alpha$ where $\langle x, y\rangle = \sum_{i=1}^{d} x_i y_i$ is the inner product of vectors x and y. The goal is to distinguish these cases based on information on each vector encoded independently in a bit string of the shortest length possible. In contrast to much work on compressing vectors using randomized dimensionality reduction, we seek to solve the problem *deterministically*, with no probability of error. Inner product estimation can be solved in general via estimating $\langle x, y\rangle$ with an additive error bounded by $\varepsilon = \alpha - \beta$. We show that $d \log_2\left(\frac{\sqrt{1-\beta}}{\varepsilon}\right) \pm \Theta(d)$ bits of information about each vector is necessary and sufficient. Our upper bound is constructive and improves a known upper bound of $d \log_2(1/\varepsilon) + O(d)$ by up to a factor of 2 when β is close to 1. The lower bound holds even in a stronger model where one of the vectors is known exactly, and an arbitrary estimation function is allowed.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Similarity, estimation, dot product, filtering

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.22

Funding The work leading to this research has received funding from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.
Rasmus Pagh: Supported by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

Acknowledgements We would like to thank Thomas Dybdahl Ahle for suggesting a simple proof for an inequality in section 4, and the anonymous reviewers for many constructive suggestions, improving the presentation. Part of this work was done while the first author was visiting Google Research.

1 Introduction

Modern data sets increasingly consist of noisy or incomplete information, which means that traditional ways of matching database records often fall short. One approach to dealing with this in database systems is to provide *similarity join* operators that find pairs of tuples satisfying a similarity predicate. We refer to the book of Augsten and Böhlen [5] for a survey of similarity joins in relational database systems, and to [30] for an overview of theoretical results in the area. Note that joins can be implemented using similarity search *indexes* that allow searching a relation for tuples that satisfy a similarity predicate with a given query q. Thus we include works on similarity search indexes in discussion of previous work on similarity join.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 22; pp. 22:1–22:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we consider *inner product similarity* predicates of the form $\langle x, y\rangle /geq\alpha$, where $x, y \in \mathbb{R}^d$ are real-valued vectors, i.e., the predicate is true for vectors whose inner product $\sum_{i=1}^{d} x_i y_i$ exceeds a user-specified threshold α. This notion of similarity is important, for example, in neural network training and inference, (see [38]). In that context, an inner product similarity join can be used for classifying a collection of vectors according to a neural network model.

Inner product similarity join is a special case of similarity join under Euclidean distance, implemented for example in the Apache Spark framework[1]. Conversely, it generalizes *cosine similarity*, which has been studied for more than a decade (see, e.g., the influential papers [8, 11] and more recent works such as [4, 7, 13, 35]). In recent years the general inner product similarity join problem has attracted increasing attention (see e.g. [2, 34, 37, 39, 43]). Recently proposed practical inner product similarity join algorithms work by reducing the general problem to a number of instances with unit length vectors, which is equivalent to join under cosine similarity [43].

Candidate generation approach. State-of-the-art algorithms computing similarity joins on high-dimensional vectors use a two-phase approach:

1. Generate a set of *candidate pairs* (x, y) that contains all pairs satisfying the predicate (keeping track of the corresponding tuples in the relations).
2. Iterate over the candidate pairs to *check* which ones satisfy the predicate.

Suppose for simplicity that both relations in the similarity join contain n tuples. A naïve candidate generation phase would output all n^2 corresponding pairs of vectors. For many data sets it is possible to reduce the number of candidate pairs significantly below n^2, but the check phase remains a bottleneck. A direct implementation of the check phase would require full information about the vectors (x, y), in practice d floating point numbers per vector. Though the inner product computation is trivial, for high-dimensional vectors the cost of transferring data from memory can be a bottleneck.

Filtering candidate pairs using approximation. An approach to reducing communication is to *approximate* inner products, which is enough to handle those candidate pairs that do not have inner product close to the threshold α. The exact inner product is computed only for the remaining pairs, often a small fraction of the set of all candidates. We stress that globally, the join computation we consider is not approximate, but approximations are used to speed up parts of the algorithm. (Note that under common assumptions in fine-grained complexity, the inner product similarity join problem is difficult in the worst case, even with approximation [1, 2].)

Such additional filtering of candidate pairs has been successfully used in "Monte Carlo" style randomized algorithms that allow the algorithm to sometimes fail to identify a pair satisfying the predicate, e.g. [35, 6]. While the error in Monte Carlo algorithms can usually be made very small at a reasonable computation cost, such algorithms are not suitable in all settings. For example:

- Firm guarantees may be needed to comply with regulation, or to ensure a clear and consistent semantics of a system (such as a DBMS) in which the similarity estimation algorithm is part.
- Guarantees on accuracy are shown under the assumption that the input data is independent of the random choices made by the algorithm. Technically this assumption may not hold

[1] https://spark.apache.org/

if output from the algorithm can affect future inputs. Maybe more seriously, if vectors can be chosen adversarially based on (partial) knowledge of the randomness of the algorithm, for example obtained by timing attacks, the guarantees cease to hold (see e.g. [14] for more discussion of adversarial settings).

In this paper we study what kind of approximation is possible without randomization, targeting settings where false negatives are not permitted, or where we cannot ensure that inputs are independent of any randomization used by the algorithm.

We seek to efficiently eliminate all candidate pairs that have inner product less than β, for some β smaller than the threshold α, so that the number of remaining candidate pairs (for which an expensive inner product computation must be done) may be significantly reduced. In order to not eliminate any candidate pair passing the threshold it is necessary and sufficient that the approximation is strong enough to distinguish the cases $\langle x, y\rangle \leq \beta$ and $\langle x, y\rangle \geq \alpha$.

Similarity join memory bottlenecks. The complexity of similarity joins in the I/O model was studied in [31], which assumes that a block transfer moves B vectors from or to external memory, and that internal memory can hold M vectors. Reducing the amount of data that needs to be transferred to evaluate a similarity predicate leads to a larger capacity of blocks as well as internal memory, in turn leading to a reduction in I/O complexity that is roughly proportional to the reduction in size. The exact improvement is a bit more complicated because additional I/Os are needed to evaluate the exact inner products of pairs with similarity above β. McCauley and Silvestri [29] studied the related problem of similarity joins in MapReduce where considerations similar to the I/O model can be made.

1.1 Our results

Without loss of generality we can consider *unit vectors*, since the general estimation problem can be reduced to this case by storing an (approximate) norm of each vector in space independent of the number of dimensions. Similarly, lower bounds shown for unit vectors imply lower bounds for arbitrary vector lengths by a scaling argument

We study the following version of the inner product estimation problem for unit vectors: Distinguish inner products smaller than β from inner products larger than α, for threshold parameters α and β. This problem can of course be solved by estimating the inner product with additive error less than $\alpha - \beta$. However, we will see that the number of bits needed is not a function of $\alpha - \beta$, and that guarantees can be improved when these parameters have values close to 1.

Let x and y be vectors from the d-dimensional Euclidean unit sphere $\mathbb{S}^{d-1}$. When represented in a computer with limited precision floating or fixed-point numbers, the precision we can obtain when computing the inner product $\langle x, y\rangle$ will of course depend on the precision of the representation of x and y. Suppose we round coordinates x and y to the nearest integer multiple of ε/d, for some parameter $\varepsilon > 0$, to produce "uniformly quantized" vectors x' and y'. Then it is easy to see that the difference between $\langle x, y\rangle$ and $\langle x', y'\rangle$ is at most ε. The space required to store each coordinate $x'_i, y'_i \in [-1, +1]$ is $\lceil \log_2(2d/\varepsilon)\rceil$ bits, so we get $d \log_2(d/\varepsilon) + O(d)$ bits in total using standard, uniform quantization. On the upper bound side we know that the number of bits per dimension can be made independent of d. The following lemma appears to be folklore – a proof can be found in [3, Theorem 4.1].

▶ **Lemma 1.** *For every $\varepsilon > 0$ there exists a mapping $\mathcal{E} : S^{d-1} \to \{0,1\}^\ell$, where $\ell = d\log_2(1/\varepsilon) + O(d)$ such that $\langle x, y\rangle$ can be estimated from $\mathcal{E}(x)$ and $\mathcal{E}(y)$ with additive error at most ε.*

In this paper we ask if this space usage is optimal for the problem of distinguishing between two specific inner product values. Our methods will work through a decoding function that produces a unit vector from a bit representation (i.e., the approximation is a result of quantizing the input vectors). Specifically, we consider the following problem:

▶ **Definition 2.** *For positive integers d and ℓ, and $\alpha, \beta \in [0,1]$ with $\alpha > \beta$ the (α, β, d, ℓ)-InnerProduct problem is to construct mappings $\mathcal{E} : \mathbb{S}^{d-1} \to \{0,1\}^\ell$ and $\mathcal{D} : \{0,1\}^\ell \to \mathbb{S}^{d-1}$ and choose $t \in \mathbb{R}$, such that for every choice of unit vectors $x, y \in \mathbb{S}^{d-1}$ we have:*

$$\begin{aligned} \langle x, y\rangle \geq \alpha &\implies \langle \mathcal{D}(\mathcal{E}(x)), \mathcal{D}(\mathcal{E}(y))\rangle \geq t \text{ and} \\ \langle x, y\rangle \leq \beta &\implies \langle \mathcal{D}(\mathcal{E}(x)), \mathcal{D}(\mathcal{E}(y))\rangle < t \ . \end{aligned}$$

We refer to the parameter ℓ as the space usage *of a construction. Whenever d and ℓ are understood from the context we omit them and talk about the (α, β)-InnerProduct problem.*

On the upper bound side our main technical lemma is the following:

▶ **Theorem 3.** *(α, β)-InnerProduct can be solved using space $\ell = d\log_2\left(\frac{\sqrt{1-\beta}}{\alpha-\beta}\right) + O(d)$.*

Theorem 3 upper bounds the space needed to approximate inner products between unit vectors. For example we can distinguish pairs with inner product $\alpha = 1 - \varepsilon$ from pairs with inner product less than $\beta = 1 - 2\varepsilon$ using space $\frac{d}{2}\log_2\frac{1}{\varepsilon} + O(d)$. The problem is closely linked to estimation, so it is unsurprising that it matches the bound in Lemma 1 for $\alpha - \beta = \varepsilon$ in the worst case of $\beta = 0$. What is interesting is that for β close to 1 we get improved constants, saving up to a factor 2 on the space when α approaches 1.

Our proof uses a variant of *pyramid vector quantization* [15] and the technique is essentially an implementation of a grid-based ε-net as described in [3], though the analysis is different. The exposition is supposed to be self-contained, and in particular we do not assume that the reader is familiar with pyramid vector quantization or ε-nets.

Finally, we show a tight lower bound. Consider a communication protocol where Alice is given $x \in \mathbb{S}^{d-1}$ and Bob is given $y \in \mathbb{S}^{d-1}$. For parameters $\alpha, \beta \in (0,1)$, with $\alpha = \beta + \varepsilon$, known to both parties, how many bits of information does Alice need to send to Bob in order for Bob to be able to distinguish the cases $\langle x, y\rangle \geq \alpha$ and $\langle x, y\rangle \leq \beta$? Specifically, how many bits must Alice send, in the worst case over all vectors x, to allow Bob to answer correctly for every vector y? We note that a solution for the (α, β)-InnerProduct problem implies a communication protocol using ℓ bits, but our lower bound applies more generally to any one-way communication protocol, not necessarily based on quantization.

▶ **Theorem 4.** *For each choice of $\alpha, \beta \in (0,1)$ with $\alpha > \beta$, suppose that there exists a mapping $\mathcal{E} : S^{d-1} \to \{0,1\}^\ell$ such that for all $x, y \in S^{d-1}$ we can determine from $\mathcal{E}(x)$ and $\mathcal{E}(y)$ whether $\langle x, y\rangle \leq \beta$ or $\langle x, y\rangle \geq \alpha$ (or output anything if $\langle x, y\rangle \in (\beta, \alpha)$). Then $\ell \geq d\log_2\left(\frac{\sqrt{1-\beta}}{\alpha-\beta}\right) - O(d)$.*

This matches the upper bound up to the additive term of $O(d)$ bits.

2 Further related work

Motivating applications. Calculating the inner product of two vectors is a frequently used sub-routine in linear algebra, and many machine learning algorithms rely heavily on inner product calculation. For example, the inner loop of algorithms for training of complex neural networks uses millions and millions of inner product computations. Often what is ultimately learned is an embedding onto a high dimensional unit sphere where the inner product can be used directly as a similarity measure.

In such large scale computations the bottleneck is often the limited bandwidth of the hardware in question, and having slightly smaller vector representations can massively improve the execution time. This gives rise to the idea of computing inner products with reduced precision. Recently, several studies showed that deep neural networks can be trained using low precision arithmetic, see e.g. [12, 18, 19]. This has led to a new generation of software and reduced-precision hardware for machine learning algorithms:

- NVIDIA's TensorRT GPU framework and Google's TensorFlow and Tensor Processing Unit, that both operate with 8- or 16-bit fixed point number representations, and
- Intel's Nervana processor that uses the so-called *FlexPoint* vector representation [28], combining 16-bit uniform quantization with a shared exponent that allows representing vectors in a large dynamic range of magnitudes.

From a theoretical point of view these hardware representations use at least $\log_2 d \pm O(1)$ more bits per dimension than what is required to ensure a given additive error.

Reduced precision inner products have also been employed in knowledge discovery [9] and similarity search [17, 25].

Dimensionality reduction. There is a large literature on the space complexity of estimating Euclidean distances, usually studied in the setting where a certain failure probability $\delta > 0$ is allowed, and with number of *dimensions* (rather than bits) as the measure of space. For certain "random projection" mappings $f : \mathbb{R}^d \to \mathbb{R}^D$ one can estimate the Euclidean distance $||x-y||_2$ from $f(x)$ and $f(y)$ up to a multiplicative error of $1+\varepsilon$, with failure probability δ. It is known that using $D = O(\log(\delta^{-1})\varepsilon^{-2})$ dimensions is necessary [24, 27] and sufficient [26]. For unit vectors this implies an approximation of inner products with $O(\varepsilon)$ additive error through the identity

$$\langle x, y\rangle = \tfrac{1}{2}(||x||_2^2 + ||y||_2^2 - ||x-y||_2^2) \ . \tag{1}$$

Using (a specific type of) random projections to estimate inner products, with an additive error guarantee, is known as "feature hashing" [42].

Indyk et al. [20, 21] considered the bit complexity of representing all distances, up to a given relative error $1+\varepsilon$, within a given set S of n vectors in $\mathbb{R}^d$. For this problem one can assume without loss of generality that $d = O(\varepsilon^{-2}\log n)$, using dimension reduction. Suppose that we only need to preserve distances of unit vectors up to an additive ε, which implies that inner products are preserved up to $O(\varepsilon)$. Then for $d = \varepsilon^{-2}\log n$ the space usage per point of the method described in [21] is $O(d\log(1/\varepsilon))$. This is within a constant factor of our upper bound, but not directly comparable to our result which works for all unit vectors. Recently, Indyk and Wagner [22] studied the space required to solve the d-dimensional Euclidean $(1+\varepsilon)$-approximate nearest neighbor problem in the setting where vector coordinates are integers in a bounded range (e.g. of size $n^{O(1)}$). While this method gives guarantees for new vectors outside of S their method is randomized and can fail to correctly determine an approximate nearest neighbor, while our method is deterministic.

Vector quantization. In a nutshell, vector quantization [16] is the process of mapping vectors in a space (usually a bounded subset of Euclidean space) to the nearest in a finite set of vectors Q. The goal is to minimize the size of Q and the distance between vectors and their quantized versions, often with respect to a certain distribution of source (or input) vectors. Fischer first described *pyramid vector quantization* [15], showing that it is near-optimal for Laplacian sources. Since high-dimensional Laplacian vectors have lengths that are tightly concentrated around the expectation, it is natural to speculate if the method is also near-optimal for fixed-length (or unit) vectors. It turns out to be easier to analyze a variant of pyramid vector quantization for which we can show that this is indeed the case. This is described in section 3.

Quantization methods have previously been used to speed up nearest neighbor search. The technique of *product quantization* [25] has been particularly successful for this application. Product quantization uses an initial random rotation of input vectors followed by application of an optimal quantization method on low-dimensional blocks. Since the size of the codebook is fixed for each block the resulting quantization error cannot be bounded with probability 1.

Quantization of the unit sphere has been studied in complexity theory as *ε-nets for spherical caps*. Rabani and Shpilka [33] give a construction in which $|Q|$ is polynomially related to the best size possible with a given quantization error. Along and Klartag [3] use such nets to achieve $|Q|$ that is within a factor $\exp(O(d))$ of optimal, improving [33] whenever the quantization error is $o(1)$, such that $|Q|$ must be superexponential in d.

In the literature on machine learning (and its application areas) there is a myriad of methods for learning a data-dependent quantization mapping that exploits structure in a data set to decrease the quantization error. We refer to the survey of Wang et al. [41] for details. In contrast to such methods, we seek guarantees for all vectors, not just vectors from a given data set or distribution.

Communication complexity. Consider a communication protocol in which Alice and Bob are given unit vectors $x, y \in \left\{\pm\frac{1}{\sqrt{d}}\right\}^d$ and need to approximate $\langle x, y\rangle$. The *gap hamming* problem is the special case where the task is to distinguish between cases of weak positive and negative correlation: Is $\langle x, y\rangle > 1/\sqrt{d}$ or is $\langle x, y\rangle < -1/\sqrt{d}$? This problem is known to require $\Omega(d)$ bits of communication [10, 23, 36], even for randomized protocols with error probability, say, $1/3$. In turn, this implies a lower bound for the space complexity of estimating inner products, since a space complexity of ℓ bits implies a (one-way) communication protocol using ℓ bits of communication. The lower bound extends to arbitrary thresholds α and β with $\alpha - \beta = \Theta(1/\sqrt{d})$ by translation. In this paper we consider general unit vectors $x, y \in \mathbb{R}^d$ and are able to show a higher lower bound of $\frac{1}{2}d\log_2 d - O(d)$ bits for distinguishing inner products of distance $\varepsilon = \Theta(1/\sqrt{d})$.

3 Upper bound

We use a well-known grid-based rounding method to construct our representation, see e.g. [3, 15]. For completeness we provide a simple, self-contained description of a representation and show that it has the properties described in Theorem 3. The grid resolution is controlled by a parameter $\delta \in [0, 1]$. For every vector $x \in \mathbb{R}^d$ let $f(x) \in \mathbb{R}^d$ be defined by

$$f(x) = x'/||x'||_2, \text{ where } x'_i = \left\lfloor x_i \tfrac{\sqrt{d}}{\delta} + \tfrac{1}{2} \right\rfloor \frac{\delta}{\sqrt{d}} \text{ for } i = 1, \ldots, d \ . \tag{2}$$

It is clear that the number of bits for storing a single integer coordinate x'_i can be large in high dimensions, up to $\log_2(2\sqrt{d}/\delta)$ bits, but we can give a much better bound on the *average* number of bits per coordinate. If $\|x\| \leq 1$ we can store $f(x)$ using $\ell = d\log_2(1/\delta) + O(d)$ bits of space. To compute x' it suffices to know the integers $z_i = \lfloor x_i \frac{\sqrt{d}}{\delta} + \frac{1}{2} \rfloor$. We first allocate d bits to store the set $\{i \mid z_i < 0\}$, such that it only remains to store the sequence of absolute values $|z_1|, \dots, |z_d|$. Next, using $\|x\|_2 \leq 1$ we observe that

$$\sum_{i=1}^{d} |z_i| \leq \|x\|_1 \frac{\sqrt{d}}{\delta} + d/2 \leq \sqrt{d}\, \|x\|_2 \frac{\sqrt{d}}{\delta} + d/2 \leq d/\delta + d/2 \ .$$

Thus if we set $s = \lfloor d/\delta + d/2 \rfloor$ we can encode $|z_1|, \dots, |z_d|$ by specifying a partitioning of s elements into $d+1$ parts. There are $\binom{s+d}{d}$ such partitionings so we can assign each vector a unique representation of $\ell = \left\lceil \log_2 \binom{s+d}{d} \right\rceil + d = d\log_2(1/\delta) + O(d)$ bits.

Before proving Theorem 3 we give a simple space bound for distance distortion which is useful in it own right.

▶ **Lemma 5.** *For $\delta \leq 1$ and every choice of $x, y \in \mathbb{R}^d$, defining f according to (2) we have:*

$$\|x - y\|_2 - \delta \leq \|f(x) - f(y)\|_2 \leq \|x - y\|_2 + \delta \ .$$

Proof. Observe that $|x_i - x'_i| \leq \frac{\delta}{2\sqrt{d}}$. This means that:

$$\|x - x'\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2} \leq \sqrt{\sum_{i=1}^{d} \frac{\delta^2}{4d}} = \frac{\delta}{2} \ .$$

Since $\delta \leq 1$ the angle between x and x' is bounded by $\pi/3$, and hence $\|x - f(x)\|_2 \leq 1$. This implies that $\|x - f(x)\|_2^2 = 2 - 2\langle x, x'/\|x'\|_2 \rangle \leq 1 + \|x'\|_2 - 2\langle x, x' \rangle = \|x - x'\|_2^2$, and in particular we get $\|x - f(x)\|_2 \leq \|x - x'\|_2 \leq \delta/2$. Finally, using the triangle inequality:

$$\begin{aligned} \|f(x) - f(y)\|_2 &\leq \|x - f(x)\|_2 + \|x - y\|_2 + \|y - f(y)\|_2 \leq \|x - y\|_2 + \delta, \text{ and} \\ \|f(x) - f(y)\|_2 &\geq \|x - y\|_2 - \|x - f(x)\|_2 - \|y - f(y)\|_2 \geq \|x - y\|_2 - \delta \ . \end{aligned}$$ ◀

We are now ready to prove Theorem 3.

Proof of Theorem 3. Let the encoding function $\mathcal{E}(\cdot)$ map a vector x to the ℓ-bit representation of x' as defined in (2). The decoding function $\mathcal{D}(\cdot)$ is defined such that $\mathcal{D}(\mathcal{E}(x)) = f(x)$.

By Lemma 5 we have:

$$\max\{0, \|x - y\| - \delta\} \leq \|f(x) - f(y)\| \leq \|x - y\| + \delta \ .$$

Using the distance bounds and the identity (1) several times we get:

$$\begin{aligned} \langle f(x), f(y) \rangle &= \tfrac{1}{2}(\|f(x)\|^2 + \|f(y)\|^2 - \|f(x) - f(y)\|^2) \\ &\leq \min\{1, \tfrac{1}{2}(2 + 2\|x - y\|\delta - \|x - y\|^2 - \delta^2)\} \\ &= \min\{1, \langle x, y \rangle + \|x - y\|\delta - \delta^2/2\}, \text{ and} \end{aligned}$$

$$\begin{aligned} \langle f(x), f(y) \rangle &\geq \tfrac{1}{2}(2 - 2\|x - y\|\delta - \|x - y\|^2 - \delta^2) \\ &= \langle x, y \rangle - \|x - y\|\delta - \delta^2/2 \ . \end{aligned}$$

We can then see

$$\langle x,y\rangle \geq \alpha \implies \langle \mathcal{D}(\mathcal{E}(x)), \mathcal{D}(\mathcal{E}(y))\rangle \geq \alpha - \delta\sqrt{2-2\alpha} - \delta^2/2 \text{ and}$$
$$\langle x,y\rangle \leq \beta \implies \langle \mathcal{D}(\mathcal{E}(x)), \mathcal{D}(\mathcal{E}(y))\rangle \leq \min\{1, \beta + \delta\sqrt{2-2\beta} - \delta^2/2\}$$

Setting $\delta = \frac{\alpha-\beta}{2\sqrt{2-2\beta}}$ and $t = \alpha - \delta\sqrt{2-2\alpha} - \delta^2/2$ we get a valid solution to the (α,β)-InnerProduct problem. ◀

The same grid, as specified by δ, works for every (α',β')-InnerProduct instance (with a suitable choice of threshold t) as long as

$$\delta < \frac{\alpha'-\beta'}{\sqrt{2-2\alpha'} + \sqrt{2-2\beta'}} \ . \tag{3}$$

Note that Lemma 1 also follows from Theorem 3: For a given inner product value p consider $\alpha' = p + \varepsilon/2$ and $\beta' = p - \varepsilon/2$. Setting $\delta = \varepsilon/4$, we satisfy (3) for all p and get that for every choice of unit vectors $x, y \in \mathbb{S}^{d-1}$:

$$\langle x,y\rangle - \varepsilon \leq \langle \mathcal{D}(\mathcal{E}(x)), \mathcal{D}(\mathcal{E}(x))\rangle \leq \langle x,y\rangle + \varepsilon \ .$$

4 Lower bound

Consider a one-way communication protocol solving the (α,β)-InnerProduct problem where Alice sends a string $\mathcal{E}(x)$ to Bob, and Bob must be able to output a real number $p(\mathcal{E}(x), y)$ and a threshold $t \in \mathbb{R}$ such that

$$\langle x,y\rangle \geq \alpha \implies p(\mathcal{E}(x),y) \geq t, \text{ and}$$

$$\langle x,y\rangle \leq \beta \implies p(\mathcal{E}(x),y) < t \ .$$

Note that there is no requirement on $p(\mathcal{E}(x),y)$ whenever $\langle x,y\rangle \in (\beta,\alpha)$. We wish to answer the following question: How many bits must Alice send, in the worst case over all vectors x, to allow Bob to answer correctly for every vector y?

Let $d > 1$ be an integer. For $\varepsilon > 0$ and $x \in \mathbb{R}^d$ let $B_\varepsilon(x) = \{y \in \mathbb{R}^d \mid ||x-y||_2 \leq \varepsilon\}$ be the ball of radius ε centered at x, let $B_1 = B_1(0)$ be the unit ball centered at the origin, and denote by $\text{cap}_\Theta(x) = \{y \in \mathbb{S}^{d-1} \mid \langle x,y\rangle \geq \cos\Theta\}$ the unit spherical cap around x with polar angle Θ.

4.1 Preliminaries

The gamma function is an extension of the factorial function to complex numbers. We will need the following formulas for the value of the gamma function on integers and half-integers (see e.g. [40]). For integer $n > 0$:

$$\Gamma(n+\tfrac{1}{2}) = \frac{(2n)!\sqrt{\pi}}{2^{2n}n!}$$
$$\Gamma(n+1) = n! \tag{4}$$

▶ **Lemma 6.** *For integer $d > 1$, $\Gamma(d/2+\frac{1}{2})/\Gamma(d/2+1) > 1/(3\sqrt{d})$.*

Proof. We use Stirling's approximation to the factorial:

$$\sqrt{2\pi} n^{n+1/2} e^{-n} \leq n! \leq e\, n^{n+1/2} e^{-n}$$

Together with (4) we get, for even d:

$$\Gamma(d/2+\tfrac{1}{2})/\Gamma(d/2+1) = \frac{d!\sqrt{\pi}}{2^d((d/2)!)^2} \geq \frac{\sqrt{2\pi}\, d^{d+1/2} e^{-d}}{2^d e^2 (d/2)^{d+1} e^{-d}} = \frac{2\sqrt{2\pi}}{e^2\sqrt{d}} > 1/(3\sqrt{d}) \ .$$

The case of odd d is similar. ◀

▶ **Lemma 7.** *Let $c_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$, $r > 0$, and $\delta \in (0,1)$. Then:*

1. *The volume of B_r^d, the d-dimensional ball of radius r, is $c_d\, r^d$.*
2. *The surface area of $\mathbb{S}_r^{d-1}$, the d-dimensional sphere of radius r, is $c_d\, d\, r^{d-1}$.*
3. *The surface area of cap$_\Theta^d(x)$, a unit spherical cap with polar angle Θ, is at most*

$$c_{d-1}\, d\, (2(1-\cos\Theta))^{(d-1)/2} \ .$$

Proof. Volume bound 1. is standard, see e.g. [32, page 11]. Differentiating with respect to r gives the surface area in line 2. For the upper bound 3. we express the surface area as an integral. Let $r(h) = \sqrt{h(2-h)} < \sqrt{2h}$ be the radius of the $d-1$-dimensional sphere at the base of the cap of height h. Note that the sphere has surface area $c_{d-1}\, d\, r(x)^{d-1}$. Integrating over cap heights from 0 to $1-\cos\Theta$ we bound the surface area:

$$\int_0^{1-\cos\Theta} c_{d-1}\, d\, r(x)^{d-1} dx \leq (1-\cos\Theta) c_{d-1}\, d\, (2(1-\cos\Theta))^{(d-1)/2} \ .$$

The inequality uses that $r(x) \leq r(1-\cos\Theta) \leq \sqrt{2(1-\cos\Theta)}$ for $x \in [0, 1-\cos\Theta]$. ◀

▶ **Lemma 8.** *For every $\Theta \in (0,\pi/2)$ there exists a code $\mathcal{C}_\Theta \subset S^{d-1}$ of size*

$$|\mathcal{C}_\Theta| \geq (2(1-\cos\Theta))^{(d-1)/2}/(3\sqrt{d})$$

such that for all $x, y \in \mathcal{C}_\Theta$ with $x \neq y$ we have $\langle x, y\rangle \geq \cos\Theta$.

Proof. We follow the outline of the standard non-constructive proof of the Gilbert-Varshamov bound. That is, we argue that $\mathcal{C}_\Theta$ can be constructed in a greedy manner by adding an additional point from

$$\mathbb{S}^{d-1} \backslash \bigcup_{x \in \mathcal{C}_\Theta} \text{cap}_\Theta^d(x)$$

until this set is empty. Clearly this construction produces a set $\mathcal{C}_\Theta$ with the property that every pair of points have angle greater than Θ. We observe that the procedure can stop only when the area of $\bigcup_{x\in\mathcal{C}_\Theta} \text{cap}_\Theta(x)$ exceeds that of $\mathbb{S}^{d-1}$. The number of iterations, and thus the size of $\mathcal{C}_\Theta$ is at least the ratio between the surface area of the unit sphere and a spherical cap, cap$_\Theta(\cdot)$. In turn, this is lower bounded by the ratio of bound number 2 (with $r = 1$) and 3 from Lemma 7. Using Lemma 6 we get:

$$\frac{c_d\, d}{c_{d-1}\, d\, (2(1-\cos\Theta))^{(d-1)/2}} \geq (2(1-\cos\Theta))^{-(d-1)/2}/(3\sqrt{d}) \ .$$ ◀

4.2 Space complexity

Define $\Theta = \arccos \beta - \arccos \alpha$. We claim that for every pair of vectors $x_1, x_2 \in \mathbb{S}^{d-1}$ with angle $\theta = \arccos \langle x_1, x_2 \rangle \geq \Theta$ there exists a vector $y \in \mathbb{S}^{d-1}$ such that $\langle x_1, y \rangle = \beta$ and $\langle x_2, y \rangle \geq \alpha$. Specifically, let

$$y = y(x_1, x_2) = \left(\beta - \sqrt{1-\beta^2}\,\tfrac{\cos\theta}{\sin\theta}\right) x_1 + \tfrac{\sqrt{1-\beta^2}}{\sin\theta}\, x_2 \ . \tag{5}$$

To see that y is indeed a unit vector we compute $||y||_2^2 = \langle y, y \rangle$. Since $\langle x_1, x_2 \rangle = \cos\theta$ (by definition of θ) and $\langle x_1, x_1 \rangle = \langle x_2, x_2 \rangle = 1$ (since x_1 and x_2 are unit vectors) we get:

$$\begin{aligned}\langle y, y \rangle &= \left(\beta - \sqrt{1-\beta^2}\,\tfrac{\cos\theta}{\sin\theta}\right)^2 + \left(\tfrac{\sqrt{1-\beta^2}}{\sin\theta}\right)^2 + 2\left(\beta - \sqrt{1-\beta^2}\,\tfrac{\cos\theta}{\sin\theta}\right)\left(\tfrac{\sqrt{1-\beta^2}}{\sin\theta}\right)\cos\theta \\ &= \beta^2 + \tfrac{1-\beta^2}{\sin^2\theta}(1-\cos^2\theta) \\ &= \beta^2 + (1-\beta^2) = 1 \ .\end{aligned}$$

The third equality uses the Pythagorean identity $\cos^2\theta + \sin^2\theta = 1$. Next, we check that y has the claimed inner products with x_1 and x_2:

$$\begin{aligned}\langle x_1, y \rangle &= \left(\beta - \sqrt{1-\beta^2}\,\tfrac{\cos\theta}{\sin\theta}\right) + \sqrt{1-\beta^2}\,\tfrac{\cos\theta}{\sin\theta} = \beta, \\ \langle x_2, y \rangle &= \left(\beta - \sqrt{1-\beta^2}\,\tfrac{\cos\theta}{\sin\theta}\right)\cos\theta + \tfrac{\sqrt{1-\beta^2}}{\sin\theta} \\ &= \langle (\beta, \sqrt{1-\beta^2}), (\cos\theta, \tfrac{1-\cos^2\theta}{\sin\theta}) \rangle \\ &= \langle (\beta, \sqrt{1-\beta^2}), (\cos\theta, \sin\theta) \rangle \geq \alpha \ .\end{aligned}$$

The final inequality follows since the angle between the vectors $(\beta, \sqrt{1-\beta^2})$ and $(\cos\theta, \sin\theta)$ is at most $\arccos(\beta) - \theta \leq \arccos(\beta) - \Theta = \arccos\alpha$.

Now consider the code $\mathcal{C}_\Theta$. For distinct $x_1, x_2 \in \mathcal{C}_\Theta$ we must have, for $y = y(x_1, x_2)$ as defined in (5), $p(e(x_1), y) < t \leq p(e(x_2), y)$, which means that $e(x_1) \neq e(x_2)$. Hence $R = \{\mathcal{E}(x) \mid x \in \mathbb{S}^{d-1}\}$ must contain at least $|\mathcal{C}_\Theta|$ binary strings, and in particular

$$\begin{aligned}\ell \geq \log_2 |\mathcal{C}_\Theta| &\geq \log_2 \left((2(1-\cos\Theta))^{-(d-1)/2}/(3\sqrt{d})\right) \\ &\geq \tfrac{d}{2}\log_2\left(\frac{1}{1-\cos\Theta}\right) - O(d) \\ &\geq d\log_2(1/\Theta) - O(d) \ .\end{aligned}$$

For the final inequality we use that $1 - \cos\Theta \leq \Theta^2$, which holds when $\Theta \in (0, \pi/2)$. To finish the proof we will show that whenever $0 \leq \beta \leq \alpha \leq 1$:

$$\Theta = \arccos\beta - \arccos\alpha \leq \tfrac{\pi}{2}\tfrac{\alpha-\beta}{\sqrt{1-\beta}} \ . \tag{6}$$

For each fixed $\beta \in [0, 1]$ we must show that $\arccos\alpha \geq \arccos\beta - \frac{\pi}{2}\frac{\alpha-\beta}{\sqrt{1-\beta}}$ for all $\alpha \in [\beta, 1]$. Since $\alpha \mapsto \arccos\alpha$ is concave for $\alpha \in [0, 1]$, and the function $\alpha \mapsto \arccos\beta - \frac{\pi}{2}\frac{\alpha-\beta}{\sqrt{1-\beta}}$ is linear, it suffices to check the inequality at the endpoints where $\alpha = \beta$ and $\alpha = 1$. In the former case we clearly get equality. In the latter we use the fact $\arccos\beta - \frac{\pi}{2}\sqrt{1-\beta} \leq 0$ for $\beta \in [0, 1]$ to see that the inequality (6) holds.

This proves that $\ell \geq d\log_2\left(\frac{\sqrt{1-\beta}}{\alpha-\beta}\right) - O(d)$ bits are needed, establishing Theorem 4.

Table 1 Space usage and absolute inner product error using the method of Section 3 on pairs of vectors from various real-life data sets. The space usage is measured against a baseline of using a 32-bit floating point numbers to represent each of the d dimensions. Observed errors are smaller than our worst-case bound of 4δ, probably due to cancellation effects.

Dataset	d	δ	space	median	90th pct.	max
MNIST	784	0.1	16%	0.0019	0.0052	0.0165
SIFT	128	0.1	16%	0.0034	0.0084	0.0181
DLIB	128	0.1	16%	0.0028	0.0071	0.0135
MNIST	784	0.05	19%	0.0008	0.0027	0.0103
SIFT	128	0.05	19%	0.0019	0.0044	0.0091
DLIB	128	0.05	19%	0.0012	0.0033	0.0075
MNIST	784	0.01	26%	0.0002	0.0007	0.0027
SIFT	128	0.01	26%	0.0004	0.0008	0.0100
DLIB	128	0.01	26%	0.0002	0.0005	0.0011

5 Experiments

Since the encoding in our upper bound is potentially practical, we evaluated the accuracy of the method experimentally on several data sets. We also computed the space usage of each set of vectors, based on an optimal encoding of the method of Section 3 with various values of parameter δ, and compared it to a baseline of using 32-bit floating point numbers.

We considered three data sets, MNIST (handwritten digits), SIFT (image features), and DLIB-FACES (neural net embeddings of faces on a 128-dimensional unit sphere). The data sets are chosen to span different distributions of entry magnitude. Whereas MNIST vectors are approximately sparse, DLIB vectors have smoothly distributed magnitudes. MNIST and SIFT are not natively unit vectors, so we normalized the vectors prior to encoding.

We computed the inner product error for 2000 vector pairs in each data set. Table 1 shows the maximum absolute error observed when calculating inner products using the decoded vectors compared to using the original vectors. It also shows the median absolute error and the error at the 90th percentile.

In all cases the observed errors are well below the worst case bound of $\varepsilon = 4\delta$. This can partly be explained by sparsity of vectors, since our method represents zero entries in vectors with no error. Also, while the effect of d rounding errors is d times the individual error in the worst case (which is what our theoretical results bound), in the typical case errors will tend to cancel since not all errors go in the same direction. Heuristically we could imagine that errors are independent and random, in which case we would expect the sum of all errors to be proportional to $\sqrt{d}$ rather than d.

6 Conclusion

We have established tight upper and lower bounds for the problem of representing unit vectors such that inner products can be estimated within a given additive error (with probability 1). An interesting possibility would be to consider relative error estimates of Euclidean distances (as in the recent work [22]) while not allowing any failure probability. Another potential direction would be to achieve provably smaller *expected* error, while preserving the worst-case guarantees, by applying an initial random rotation to all data vectors.

References

1 Amir Abboud, Aviad Rubinstein, and R. Ryan Williams. Distributed PCP Theorems for Hardness of Approximation in P. In *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 25–36, 2017. doi:10.1109/FOCS.2017.12.

2 Thomas Dybdahl Ahle, Rasmus Pagh, Ilya Razenshteyn, and Francesco Silvestri. On the complexity of inner product similarity join. In *Proceedings of the 35th ACM Symposium on Principles of Database Systems (PODS)*, pages 151–164. ACM, 2016.

3 Noga Alon and Bo'az Klartag. Optimal compression of approximate inner products and dimension reduction. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 639–650. IEEE, 2017.

4 David C. Anastasiu and George Karypis. L2AP: Fast cosine similarity search with prefix l-2 norm bounds. In *30th IEEE International Conference on Data Engineering (ICDE)*, pages 784–795. IEEE, 2014.

5 Nikolaus Augsten and Michael H. Böhlen. *Similarity Joins in Relational Database Systems.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013. doi:10.2200/S00544ED1V01Y201310DTM038.

6 Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Michael Vesterli. PUFFINN: parameterless and universally fast finding of nearest neighbors. In *27th Annual European Symposium on Algorithms (ESA)*, pages 10:1–10:16, 2019. doi:10.4230/LIPIcs.ESA.2019.10.

7 Raghavendran Balu, Teddy Furon, and Hervé Jégou. Beyond "project and sign" for cosine estimation with binary codes. In *IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP)*, pages 6884–6888, 2014. doi:10.1109/ICASSP.2014.6854934.

8 Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proceedings of 16th international conference on World Wide Web (WWW)*, pages 131–140. ACM, 2007.

9 Davis W. Blalock and John V. Guttag. Bolt: Accelerated data mining with fast vector compression. In *Proceedings of 23rd ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 727–735. ACM, 2017.

10 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM Journal on Computing*, 41(5):1299–1317, 2012.

11 Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002. doi:10.1145/509907.509965.

12 Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, pages 2285–2294, 2015.

13 Tobias Christiani, Rasmus Pagh, and Johan Sivertsen. Scalable and Robust Set Similarity Join. In *34th IEEE International Conference on Data Engineering (ICDE)*, pages 1240–1243, 2018. doi:10.1109/ICDE.2018.00120.

14 David Clayton, Christopher Patton, and Thomas Shrimpton. Probabilistic Data Structures in Adversarial Environments. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 1317–1334, 2019. doi:10.1145/3319535.3354235.

15 Thomas Fischer. A pyramid vector quantizer. *IEEE transactions on information theory*, 32(4):568–583, 1986.

16 Allen Gersho and Robert M. Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012.

17 Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. Quantization based fast inner product search. In *Artificial Intelligence and Statistics*, pages 482–490, 2016.

18 Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning (ICML)*, pages 1737–1746, 2015.

19 Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

20 Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Practical Data-Dependent Metric Compression with Provable Guarantees. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2614–2623, 2017.

21 Piotr Indyk and Tal Wagner. Near-optimal (euclidean) metric compression. In *Proceedings of 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 710–723. SIAM, 2017.

22 Piotr Indyk and Tal Wagner. Approximate Nearest Neighbors in Limited Space. In *Proceedings of 31st Conference On Learning Theory (COLT)*, volume 75 of *Proceedings of Machine Learning Research*, pages 2012–2036. PMLR, 2018.

23 Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 283–288. IEEE, 2003.

24 Thathachar S. Jayram and David P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms (TALG)*, 9(3):26, 2013.

25 Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

26 William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

27 Daniel Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit Johnson-Lindenstrauss families. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 628–639. Springer, 2011.

28 Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William Constable, Oguz Elibol, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J. Pai, and Naveen Rao. Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1742–1752, 2017. URL: http://papers.nips.cc/paper/6771-flexpoint-an-adaptive-numerical-format-for-efficient-training-of-deep-neural-networks.

29 Samuel McCauley and Francesco Silvestri. Adaptive MapReduce Similarity Joins. In *Proceedings of the 5th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, page 4. ACM, 2018.

30 Rasmus Pagh. Large-Scale Similarity Joins With Guarantees (Invited Talk). In *18th International Conference on Database Theory (ICDT)*, pages 15–24, 2015. doi:10.4230/LIPIcs.ICDT.2015.15.

31 Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. I/O-efficient similarity join. In *23rd Annual European Symposium on Algorithms (ESA)*, pages 941–952, 2015. doi:10.1007/978-3-662-48350-3_78.

32 Gilles Pisier. *The volume of convex bodies and Banach space geometry*, volume 94 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 1999.

33 Yuval Rabani and Amir Shpilka. Explicit construction of a small ε-net for linear threshold functions. *SIAM Journal on Computing*, 39(8):3501–3520, 2010.

34 Parikshit Ram and Alexander G. Gray. Maximum inner-product search using cone trees. In *18th ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 931–939, 2012. doi:10.1145/2339530.2339677.

35 Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *Proceedings of the VLDB Endowment*, 5(5):430–441, 2012.

36 Alexander A. Sherstov. The Communication Complexity of Gap Hamming Distance. *Theory of Computing*, 8(1):197–208, 2012.

37 Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *Advances in Neural Information Processing Systems (NIPS)*,

pages 2321–2329, 2014. URL: http://papers.nips.cc/paper/5329-asymmetric-lsh-alsh-for-sublinear-time-maximum-inner-product-search-mips.

38 Ryan Spring and Anshumali Shrivastava. Scalable and Sustainable Deep Learning via Randomized Hashing. In *Proceedings of 23rd ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 445–454, 2017. doi:10.1145/3097983.3098035.

39 Christina Teflioudi and Rainer Gemulla. Exact and approximate maximum inner product search with LEMP. *ACM Transactions on Database Systems (TODS)*, 42(1):5, 2017.

40 Raimundas Vidunas. EXPRESSIONS FOR VALUES OF THE GAMMA FUNCTION. *Kyushu Journal of Mathematics*, 59(2):267–283, 2005.

41 Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data—a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

42 Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of 26th International Conference on Machine Learning (ICML)*, pages 1113–1120. ACM, 2009.

43 Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. Norm-Ranging LSH for Maximum Inner Product Search. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2956–2965, 2018. URL: http://papers.nips.cc/paper/7559-norm-ranging-lsh-for-maximum-inner-product-search.

A Family of Centrality Measures for Graph Data Based on Subgraphs

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
cristian.riveros@uc.cl

Jorge Salas
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
jusalas@uc.cl

Abstract

We present the theoretical foundations of a new approach in centrality measures for graph data. The main principle of our approach is very simple: the more relevant subgraphs around a vertex, the more central it is in the network. We formalize the notion of "relevant subgraphs" by choosing a family of subgraphs that, give a graph G and a vertex v in G, it assigns a subset of connected subgraphs of G that contains v. Any of such families defines a measure of centrality by counting the number of subgraphs assigned to the vertex, i.e., a vertex will be more important for the network if it belongs to more subgraphs in the family. We show many examples of this approach and, in particular, we propose the all-subgraphs centrality, a centrality measure that takes every subgraph into account. We study fundamental properties over families of subgraphs that guarantee desirable properties over the corresponding centrality measure. Interestingly, all-subgraphs centrality satisfies all these properties, showing its robustness as a notion for centrality. Finally, we study the computational complexity of counting certain families of subgraphs and show a polynomial time algorithm to compute the all-subgraphs centrality for graphs with bounded tree width.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Information systems → Graph-based database models

Keywords and phrases Graph data, graph centrality, centrality measures

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.23

Funding C. Riveros and J. Salas were partially funded by the Millennium Institute for Foundational Research on Data.

1 Introduction

Which are the most important or "central" nodes in a network? This is a crucial question that has been asked in several areas like social science [21], biology [19], computer science [9] and essentially every area where graph data is relevant [25]. Given the graph structure of data one expects that more central nodes are more important for the network and they will be relevant in understanding its underlying structure. Several centrality measures has been proposed like closeness [4], betweenness [15], Page Rank [9], Katz index [20], among others [25], trying to give an answer or explanation to our first question.

Which centrality measure is the most meaningful for network analysis? This has been behind all proposals of centrality measures and it is an old question that has been discussed from the beginning of network analysis [7, 16, 27]. Over the years, some axioms or properties have been risen as crucial for a centrality measure and several centrality measures has been axiomatized [28, 29, 31]. However, as it was shown in [6] many commonly used centrality

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 23; pp. 23:1–23:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

measures do not satisfy even a simple set of "desirable" axioms (i.e. properties). The question above then remains unanswered: how to naturally and formally define a centrality measure that has reasonable properties?

To motivate our approach that aims both questions, consider the following setting from graph data management. Suppose a graph database G and a query language $\mathcal{L}$ for extracting patterns from G. Further, suppose Q is a query in $\mathcal{L}$ such that the evaluation of Q over G, denoted by $Q(G)$, retrieves a set of nodes in G. How should we rank $Q(G)$ in order to output the most meaningful outputs first? More specifically, suppose that G is a property graph and $\mathcal{L}$ is a language of basic graph patterns [1]. Given that queries dynamically change over time [10, 22], one would expect that, if $v \in Q(G)$ satisfies more patterns from $\mathcal{L}$, it will have more chances to appear latter as an extension of Q. More general, one would expect that the more queries from $\mathcal{L}$ where v is included, the more important is v on G with respect to $\mathcal{L}$. Furthermore, depending whether $\mathcal{L}$ is designed to look for paths, trees, or maybe triangles [1] on G, maybe the user would like its measure of centrality to focus more on these patterns than in all basic graph patterns.

In this paper, we tackle the first question following the simple idea motivated from the graph data management setting: the more relevant subpatterns around a node, the more central it is in the network. Several proposals in the literature (e.g degree, betweenness [15], cross-clique [14]) already have considered relevant subpatterns like edges, paths, or cliques to define meaningful centrality measures. We generalize this approach by defining centrality measures based on families of subgraphs. Specifically, we formalize the notion of "relevant subgraphs" by choosing any family of subgraphs that, given a graph G and a vertex v in G, it assigns a subset of connected subgraphs from G that contains v. Any of such families defines a measure of centrality by counting the number of subgraphs assigned to the vertex, i.e., a vertex will be more important for the network if it belongs to more subgraphs in the family. We show several examples that can be derived by following this approach. In particular, a natural family of subgraphs is to consider all connected subgraphs around a vertex, that we called *all-subgraphs centrality*, and we show that it defines a well-behaved notion of centrality.

With a family of centrality measures at hand we embark on answering the second question. Generally speaking, we can consider any property on the family of subgraphs and see what "axiom" it implies in the respectively centrality notion that it defines. With this strategy, we no longer depend on comparing centrality measures of different nature (e.g. Page Rank vs Betweenness). Instead, we can understand all centrality notions proposed in this paper by just understanding the properties that satisfy the families of subgraphs. We consider simple axioms that has been proposed in the literature (e.g. monotonicity [27] or isolated vertex [16]). Then, look for simple properties in the family of subgraph that imply them. Interestingly, we can show natural examples of families of subgraphs that do not satisfy these properties and whose corresponding centrality notions do not satisfy the axioms. This allows to have a more deep understanding of why a centrality measure does not behave as expected and, moreover, to look into ways on how to "fix" it. Finally, the all-subgraphs centrality proposed in this paper satisfies all these properties and axioms, showing its robustness as a measure for centrality.

The general definition of centrality based on subgraphs allows us to easily extend the idea from vertices to sets of vertices, also called group centrality. We propose an approach to extend every centrality measure to groups, and prove a natural way to reduce the computation of all-subgraph centrality from groups to vertices. We show that this extension over sets allows to answer simple questions on the dynamic of graphs, like how to maximize the centrality of a vertex when an edge is added.

Towards the end of the paper, we study the computational complexity of counting certain families of subgraphs. Unfortunately, we show that the centrality measures defined from families of subgraphs like all subgraphs or trees lead to intractability. In terms of good news, we show that these centralities can be efficiently computed in acyclic graphs (i.e. trees). Moreover, we show that this result can be extended to more classes of graphs, by showing a polynomial time algorithm to compute the all-subgraphs centrality over all classes of graphs with bounded tree width.

Related work. Centrality measures have been extensively studied since the 50's [4, 21] and the subject is spread in different research areas. Moreover, the literature contains several alternative proposals that, given space restrictions, it will be impossible to cover all of them here (see [25]). Instead, we review here the work that is more closed in spirit to our proposal by stressing the main differences.

Centrality measures based on some relevant subgraphs have been studied before (e.g. betweenness [15], cross-clique [14]). The difference with our approach is that we take a step further and studied families of subgraphs in a more general setting. In particular, to the best of our knowledge all-subgraphs centrality and trees centrality (see Section 3 and 4) are new measures and have not been studied before.

There are several papers that have studied centrality measures in terms of properties [7, 16, 27]. Furthermore, in the last years there are several proposals to axiomatize standard centrality measures [5, 6, 28, 29, 31]. In this paper, we study properties and axioms in terms of families of subgraphs, which is a different goal compared to previous approaches.

Finally, a centrality measure called subgraph centrality was proposed in [13]. Although the name resemble our approach and the paper also motivates the use of subgraphs, subgraph centrality sums the number of closed-walks weighted by its length and not all the connected subgraphs that contains a nodes, as in our case.

2 Preliminaries

For a finite set V, we denote by $\text{edges}(V) = \{\{u, v\} \subseteq V \mid u \neq v\}$ all subsets of V of size two. Sometimes, we consider a function f as a relation and write $f' \subseteq f$ when f' is a (partial) function resulting to take a subset of the order pairs from f. In the sequel, all logarithms are in base 2 unless it is stated differently.

Undirected graphs. We consider finite undirected graphs of the form $G = (V, E)$ where V is a finite non-empty set and $E \subseteq \text{edges}(V)$. Given a graph G, we will denote by $V(G)$ and $E(G)$ the set of vertices and edges, respectively. We will usually use u and v for denoting vertices and e and f for edges. Furthermore, we will use edges as sets and write $v \in e$ when e is an edge incident to v. We denote by $N(v, G) = \{u \mid \{u, v\} \in E(G)\}$ the neighborhood of v in G. We say that a graph G' is a subgraph of G, denoted $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. If two graphs G_1 and G_2 are isomorphic, we write $G_1 \cong G_2$. Furthermore, we write $G_1, v_1 \cong G_2, v_2$ for $v_1 \in V(G_1)$ and $v_2 \in V(G_2)$ if $G_1 \cong G_2$ and v_1 is equivalent to v_2 under the bijective function between G_1 and G_2.

Multigraphs. We also work with graphs with multiple edges between vertices, called multigraphs. A multigraph M is a triple $M = (V, E, r)$ such that V is a finite non-empty set, E is a finite set, and $r : E \to \text{edges}(V)$ (i.e. the edge-assignment function). Intuitively, E is a set of identifiers for edges and r assigns identifiers to edges (i.e. there could be multiple edges between two pair of vertices). Similar than for graphs, we denote by $V(M)$, $E(M)$, and $r(M)$ the corresponding set of vertices, edges, and edge-assignment of M, respectively. We say that a multigraph M' is a sub-multigraph of M, denoted by $M' \subseteq M$, if $V(M') \subseteq V(M)$, $E(M') \subseteq E(M)$, and $r(M') \subseteq r(M)$.

Note that a simple graph is a multigraph M where $r(M)$ is an injective function. For this reason, in the future we will not make distinction between graphs and multigraphs. Furthermore, we will usually work with graphs but all definitions and results also extend to multigraphs. When this is not the case, we will make the distinction explicitly.

Connected graphs. A path in a graph G is a sequence of nodes $\pi = v_0, \ldots, v_n$ such that $\{v_i, v_{i+1}\} \in E(G)$ for every $i < n$ and we say that the length of π is n. Note that v_0 is the trivial path from v_0 to itself of length 0. We say that G is connected if there exists a path between any pair of vertices. Furthermore, we say that $G' \subseteq G$ is a connected component of G if G' is connected and its maximal element over all subgraphs of G under $\subseteq$. We denote by $\text{ConnComp}(G)$ the set of all connected components of G. For $u, v \in V(G)$ we say that u is at distance d of v if there exists a path from u to v of length d and every path from u to v is of length at least d. We denote the distance d from u and v in G by $\text{dist}_G(u, v)$. Given this distance, the diameter of G is defined as $\max_{u,v \in V(G)} \text{dist}_G(u, v)$.

Families. We consider several families of graphs through the paper to give examples or show some properties of our centrality measures. Given a vertex v, we denote by G_v the graph with one vertex v (i.e. $V(G_v) = \{v\}$) and no-edges (i.e. $E(G_v) = \emptyset$). Given an edge $e = \{u, v\}$, we denote by G_e the graph only containing e (i.e. $V(G_e) = \{u, v\}$ and $E(G_e) = \{e\}$). For any $n \geq 1$, we write S_n for the star with $n + 1$ vertices such that $V(S_n) = \{0, 1, \ldots, n\}$ and all nodes are connected to 0, namely, $E(S_n) = \{\{0, i\} \mid 0 < i \leq n\}$. Similarly, we write L_n for the line with n vertices where $V(L_n) = \{0, \ldots, n-1\}$ and $E(L_n) = \{\{i, i+1\} \mid 0 \leq i < n-1\}$. The circuit with n vertices is denoted by C_n with $V(C_n) = \{0, \ldots, n-1\}$ and $E(C_n) = \{\{i, (i+1) \mod n\} \mid 0 \leq i \leq n-1\}$. Finally, the clique of size n is denoted by K_n where $V(K_n) = \{0, \ldots, n-1\}$ and $E(K_n) = \text{edges}(V(K_n))$.

Operations. Through the paper, we use several operations to create, modify, or combine graphs. Given $v \in V(G)$, we denote by $G - v$ the result of removing v from G and all its incident edges, namely, $V(G - v) = V(G) \setminus \{v\}$ and $E(G - v) = \{e \in E(G) \mid v \notin e\}$. Given $e = \{u, v\}$, we write $G + e$ for the result of adding e into G, formally, $V(G + e) = V(G) \cup e$ and $E(G + e) = E(G) \cup \{e\}$ (i.e. if u or v are not in G, then they are included as new vertices). Instead, we write $G - e$ for the result of removing all edges between u and v, namely, $V(G - e) = V(G)$ and $E(G - e) = E(G) \setminus \{e\}$. Note that if G is a (simple) graph, then at most one edge is removed, but if G is a multigraph then all edges between u and v are removed. For G_1 and G_2, we denote by $G_1 \cup G_2$ the union of the two graphs, namely, $V(G_1 \cup G_2) = V(G_1) \cup V(G_2)$ and $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$. In particular, $G + e = G \cup G_e$.

Let G be a graph and $U \subseteq V(G)$. We define the *set contraction* of U on G as the multigraph G/U by merging the vertices U to one vertex (called U) and keeping multi-edges into U. Formally, G/U is the multigraph M such that $V(M) = (V(G) \setminus U) \cup \{U\}$, $E(M) = \{e \in E(G) \mid e \not\subseteq U\}$ and for every $e \in E(M)$ either $r(M)(e) = e$ whenever $e \cap U = \emptyset$, or $r(M)(e) = \{v, U\}$ whenever $e = \{v, u\}$ with $v \notin U$ and $u \in U$. Note that we use U (i.e. the set) as the new vertex that represent the contraction in M/U. When G is a multigraph, the set contraction G/U easily follows from the above definition.

3 The all-subgraphs centrality

We start by introducing our first centrality measure based on all subgraphs, called the all-subgraphs centrality. In the next section, we generalize this idea to any family of subgraphs.

Fix a graph G and a vertex $v \in V(G)$. We denote by $\mathcal{A}(v, G)$ the set of all connected subgraphs of G that contains v, formally, $\mathcal{A}(v) = \{G' \subseteq G \mid G' \text{ is connected} \wedge v \in V(G')\}$. The *all-subgraphs centrality* of v in G is defined as:

$$C_{\mathcal{A}}(v, G) \;:=\; \log\big(\,|\mathcal{A}(v, G)|\,\big)$$

namely, the logarithm of the number of connected subgraphs of G that contains v. Intuitively, the all-subgraphs centrality of a node only considers connected graphs since it captures the importance of the node in the neighborhood that it belongs. We add more importance to a node if its neighborhood is richer in substructures. Furthermore, we consider connected subgraphs since there is no argument to say that a node has more centrality by counting another component that is not directly connected to it.

The function $C_{\mathcal{A}}$ naturally induces a ranking between nodes: the higher the centrality $C_{\mathcal{A}}(v, G)$, the more important is v in G. We define the ranking $<_{\mathcal{A}}$ over $V(G)$ induced by $C_{\mathcal{A}}$ (or just $\mathcal{A}$-ranking for short) such that $u <_{\mathcal{A}} v$ if, and only if, $C_{\mathcal{A}}(u, G) > C_{\mathcal{A}}(v, G)$. Strictly speaking, $<_{\mathcal{A}}$ is not an order in $V(G)$, given that there could exist vertices u and v such that $C_{\mathcal{A}}(u, G) = C_{\mathcal{A}}(v, G)$ (e.g. u and v are isomorphic in G). In this case, we write $u =_{\mathcal{A}} v$.

▶ **Example 1.** Let v be a vertex. Recall that G_v is the trivial graph with one vertex v and no edges. Then one can easily check that $C_{\mathcal{A}}(v, G_v) = 0$ given that $\mathcal{A}(v, G_v) = \{G_v\}$ and then $\log(|\mathcal{A}(v, G_v)|) = \log(1) = 0$. Note that this is the only vertex and graph (up to isomorphism) where the centrality is equal to 0. This follows the intuition that an isolated vertex must have 0 centrality since no one is connected to him.

▶ **Example 2.** Recall that S_n denotes the star graph with $n + 1$ vertices. Note that every connected subgraph of S_n corresponds to a subset of $E(S_n)$, and there are 2^n subsets of $E(S_n)$. Therefore, the centrality of the center of the star (i.e. the 0 vertex) is $C_{\mathcal{A}}(0, S_n) = n$. Interestingly, the all-subgraphs centrality of the center of a star coincides with its degree-centrality [25], following the intuition of what should be the centrality in this case. One can easily show that, for any $i \neq 0$, $C_{\mathcal{A}}(i, S_n) = n - 1 + \epsilon$ with $\epsilon \in o(1)$. Thus, in terms of ranking we have that $0 <_{\mathcal{A}} i$ and $i =_{\mathcal{A}} j$ for every $i, j > 0$.

The all-subgraphs centrality is measuring the worst-case entropy [12,24] of the set $\mathcal{A}(v, G)$, namely, the minimum number of bits that are required to represent the set $\mathcal{A}(v, G)$ with bit-codes. Of course, using the size of $|\mathcal{A}(v, G)|$ will give the same ranking of centrality over the vertex of G. Nevertheless, the log-function gives a better interpretation of the centrality in terms of information theory. Moreover, it normalizes the value $|\mathcal{A}(v, G)|$ in a scale that is in correspondence with the intuition of a centrality notion, e.g. Examples 1 and 2 above.

The next lemma is another result that validates the use of worst-case entropy and it will be useful for computing the all-subgraphs centrality over simple graphs. Recall that a vertex $v \in V(G)$ is a *cut vertex* of G if $|\operatorname{ConnComp}(G - v)| < |\operatorname{ConnComp}(G)|$, namely, whose removal increases the number of connected components of G.

▶ **Lemma 3.** *Let v be a cut-vertex of graph G and $G_1, \ldots, G_n$ are all the subgraphs that partition G and whose pairwise intersection is v, that is, $V(G) = \cup_{i=1}^{n} V(G_i)$, $E(G) = \cup_{i=1}^{n} E(G_i)$, and $V(G_i) \cap V(G_j) = \{v\}$ for $i \neq j$. Then*

$$C_{\mathcal{A}}(v, G) = \sum_{i=1}^{n} C_{\mathcal{A}}(v, G_i).$$

Namely, the centrality of v in G is the sum of its centrality in all the components G_i.

This property is usually known in the literature as cut-vertex additivity [29]. Since not every centrality measure satisfies it, this can be seen as the first distinction between all-subgraphs centrality and commonly used centrality measures (e.g. pagerank, betweenness).

▶ **Example 4.** Let G be any graph, $u \in V(G)$, and v be a new vertex not in G. For $e = \{u, v\}$, recall that G_e is the graph only containing e. Then one can easily see that $C_{\mathcal{A}}(u, G_e) = 1$. Since $G + e = G \cup G_e$ and u is a cut-vertex of $G + e$, by Lemma 3 we get:

$$C_{\mathcal{A}}(u, G + e) \;=\; C_{\mathcal{A}}(u, G) + C_{\mathcal{A}}(u, G_e) \;=\; C_{\mathcal{A}}(u, G) + 1$$

Thus, by connecting one new vertex directly to u its centrality grows exactly in one unit. This property is very appealing for a centrality measure and follows verbatim the intuition of the score-monotonicity axiom in [6] (see Section 5 for more discussion). On the other hand, one can check that the new vertex v in $G + e$ absorbs part of the centrality of u in G. Specifically, one can easily see that $|\mathcal{A}(v, G + e)| = |\mathcal{A}(u, G)| + 1$ and then $C_{\mathcal{A}}(v, G + e) = \log(|\mathcal{A}(u, G)| + 1) = C_{\mathcal{A}}(u, G) + \epsilon$, where ϵ is a negligible factor.

▶ **Example 5.** For $n \geq 1$, recall that L_n is the line with n nodes starting from 0 and ending in $n - 1$. For the 0-vertex in L_n there are n-different subgraphs, one for each vertex, and then $C_{\mathcal{A}}(0, L_n) = \log(n)$. The line graph is the most sparse graph with n vertices and 0 is the most extreme vertex in the graph. As one could expect, the centrality of 0 grows very slow, logarithmic in the number of vertex.

For the i-vertex in L_n, we can easily compute its centrality by using Lemma 3. Indeed, the centrality for i is the composition of two lines with $i + 1$ and $n - i$ vertices, respectively. Therefore, by Lemma 3:

$$C_{\mathcal{A}}(i, L_n) \;=\; C_{\mathcal{A}}(0, L_{i+1}) + C_{\mathcal{A}}(0, L_{n-i}) \;=\; \log(i + 1) + \log(n - i).$$

If n is odd, the vertex with maximum centrality is reached by the middle node $\frac{n-1}{2}$ and $C_{\mathcal{A}}(\frac{n-1}{2}, L_n) = 2(\log(n + 1) - 1)$. Thus, the middle point of a line doubles the centrality of the extreme vertices, nevertheless, the grow of its centrality is still logarithmic in n. Finally, note that the centrality is maximized in the middle node and the ranking decreases towards the extremes (i.e. $i <_{\mathcal{A}} i + 1$ for every $i < \frac{n-1}{2}$).

A natural question at this point is to think in lower and upper bounds of the centrality with respect to the number of edges of a graph. Indeed, the number of subgraphs $\mathcal{A}(v, G)$ could be exponential in G but its entropy is bounded by the number of edges as follows.

▶ **Proposition 6.** *For any connected graph G and $v \in V$, it holds that:*

$$\log(|E(G)| + 1) \;\leq\; C_{\mathcal{A}}(v, G) \;\leq\; |E(G)|.$$

From Example 2 above, we can infer that the upper bound is reached by the central vertex of a star. This follows the intuition that the central vertex of a star must be the most central vertex regarding the number of edges (i.e. all edges are pointing to him). Furthermore, in Example 5 we show that the extreme vertex of a line L_n has centrality $\log(n) = \log(|E| + 1)$. That is, the minimum centrality is reached in the extreme points of a line, agreeing with the intuition that the line graph is the most sparsest graph over all undirected graphs.

4 A family of centralities based on subgraphs

The idea of measuring the centrality of a vertex based on relevant substructures is not new [14, 15]. For example, the degree centrality counts how many edges are incident to a vertex and the betweenness centrality [15] counts how many geodesic paths passed through a vertex. In our case, all-subgraphs centrality measures all connected subgraphs including v, but maybe for an expert not all subgraphs are equally important and he will be interested in counting some of them. In this section we generalize the notion of all-subgraphs centrality to propose a framework of centrality notions based on measuring the worst-case entropy of relevant substructures surrounding a vertex.

A family of substructures is a function $\mathcal{F}$ that, given a graph G and a vertex $v \in V(G)$, it assigns a non-empty subset of connected subgraphs in G that contains v. Formally, $\mathcal{F}$ is a function such that $\mathcal{F}(v, G) \subseteq \mathcal{A}(v, G)$ and $\mathcal{F}(v, G) \neq \emptyset$. We also assume that $\mathcal{F}$ is closed under isomorphism, namely, if $G_1, v_1 \cong G_2, v_2$ then $\mathcal{F}(v_1, G_1)$ is isomorphic to $\mathcal{F}(v_2, G_2)$, by extending the isomorphism between G_1, v_1 and G_2, v_2 to subgraphs. For example, $\mathcal{A}$ is a family of substructures where $\mathcal{A}(v, G)$ contains all connected subgraphs in G containing v and is closed under isomorphism. Given a family of substructures we define the $\mathcal{F}$-subgraph centrality (denoted by $C_{\mathcal{F}}(v, G)$) as:

$$C_{\mathcal{F}}(v, G) \ := \ \log\left(|\mathcal{F}(v, G)|\right)$$

for any graph G and vertex $v \in V(G)$. In other words, following the idea of all-subgraphs centrality it measures the worst-case entropy of the substructures $\mathcal{F}(v, G)$. We could have left the framework open to any monotone positive function over $\mathcal{F}(v, G)$ instead of the logarithm, leading to the same ranking of centrality between vertices. Of course, this will derive in a more complex and enriched theory, however, for the purpose of this paper we will keep the simplicity of the logarithm as it still give place to novel results.

Note that $\mathcal{F}(v, G)$ is non-empty and, therefore, $C_{\mathcal{F}}(v, G)$ is always well-defined. Similar to all-subgraphs centrality, the centrality measures induced a ranking between nodes: we define the $\mathcal{F}$-ranking $<_{\mathcal{F}}$ over $V(G)$ such that $u <_{\mathcal{F}} v$ if, and only if, $C_{\mathcal{F}}(u, G) < C_{\mathcal{F}}(v, G)$.

▶ **Example 7.** Given a graph G and $v \in V(G)$, denote by $\mathcal{T}(v, G)$ all subgraphs $T \in \mathcal{A}(v, G)$ such that T is a tree. Note that an isolated vertex is defined as a trivial tree, so $\mathcal{T}(v, G)$ is always non-empty. Furthermore, the family $\mathcal{T}$ is closed under isomorphism. Then $C_{\mathcal{T}}$ measures the centrality of a vertex based on trees and we call it *the trees centrality*. For example, if L_n is a line graph with n vertices (see Example 5) then we have that $C_{\mathcal{A}}(v, G) = C_{\mathcal{T}}(v, G)$. Indeed, if T is a tree, then $C_{\mathcal{A}}(v, G) = C_{\mathcal{T}}(v, G)$ for every $v \in V(G)$. However, this is not always the case if G has cycles and one can find examples where the two measures give different values and ranking.

The motivation behind trees centrality is to considered substructures defined by acyclic graphs like trees or paths. For example, path queries [1] are at the core of graph queries languages and they are used to find path substructures between pair of nodes. Also, basic graph patterns that are acyclic (e.g. tree-shaped queries) forms a well-behaved core of graph query languages that can be evaluated efficiently [18]. Therefore, if the query languages mostly uses queries that are acyclic, maybe it makes sense to rank the results by a centrality notion based on trees.

The generalization of all-subgraphs centrality to any family of subgraphs opens the possibilities of defining any centrality notion based on a particular group of relevant subgraphs. In the next section, we use this framework to understand which properties in the family leads to desirable properties in the corresponding centrality measure. This will help to guide the design of a centrality notion based on subgraphs and, moreover, to have a better understanding of this framework and all-subgraphs centrality.

5 What families of subgraphs define good centrality measures?

Several attempts have been taken to define which properties a centrality measure should satisfy and how to axiomatize them [3, 28, 29]. In our framework, each family of subgraphs defines a new centrality measure, so it is not our purpose here to axiomatize them. In some sense, each family of subgraphs captures the know-how of an expert who knows what are the relevant subpatterns around a vertex. From this point of view, it does not make sense to prefer one notion of centrality over the other. Instead, we study here which properties over the family of subgraphs lead to desirable properties on the corresponding centrality notion. We hope that these properties will guide experts on the design of a centrality based on subgraphs and they will help to understand the benefits and problems of choosing one family over the other. Towards this goal, we consider several axioms of centrality that has been proposed in the literature and study which natural property on the family of subgraphs is enough to satisfy it. We also give several examples for showing what happens when a property is not satisfied.

In the sequel, a centrality measure is any function C that given a graph G and $v \in V(G)$, it outputs a non-negative value, i.e., $C(v, G) \geq 0$.

Default axioms. We start our discussion by showing three natural axioms proposed in the literature that any $C_{\mathcal{F}}$ satisfies, for any family of substructures $\mathcal{F}$. We discuss these three axioms briefly and show that they are naturally satisfied by definition.

In [28] they present the so-called *locality axiom*, which says that the centrality of a vertex should only depend on the connected component it belongs. In other words, after removing components that are not connected to a vertex v the centrality of v should not change. This natural axiom is satisfied by any centrality measure based on subgraphs because we define a family of substructures as a subset of connected subgraphs. This might be seen as an irrelevant detail but it is an important design decision of our approach. In second place, an axiom called *anonymity* is introduced in [27]. This is the same as saying that a centrality measure is closed under isomorphism. In our definition we explicitly say that any feasible substructure must be closed under isomorphism, which means that the centrality measure as defined will satisfy this axiom. Finally, in [17] the authors propose a minimum value for any centrality. More specifically, the centrality of an isolated vertex is the minimum possible and it should be 0. These two properties are called isolated minimization and isolated zero, respectively. In our case, these axioms are satisfied by definition, because the set of substructures associated to a vertex must be always non-empty, which means that $|\mathcal{F}(v, G_v)| = 1$ for any family of substructures. Therefore, the minimum possible value for any centrality measure defined in this way is 0.

Monotonicity. The monotonicity axiom is probably the property that more people [6, 27, 28] agree that any centrality notion should satisfy. In [6], the definition of this axiom says that if an edge is added to the graph, then the centrality of the vertex that is incident with the new edge should not decrease. Clearly, a vertex is more central the more edges it has and, thus, a new edge should help to increase its relevance in the graph. A more general definition of this axiom was introduced in [27] where the effect of adding any new edge in the graph should not decrease the centrality of every vertex.

▶ **Axiom 1** (Monotonicity). *A centrality measure C satisfies the monotonicity axiom if for every graph G, $v \in V(G)$ and $e \notin E(G)$, it holds that $C(v, G) \leq C(v, G + e)$.*

Note that the axiom implies that if G_1 is a subgraph of G_2 and $v \in V(G_1)$, then $C(v, G_1) \leq C(v, G_2)$. This coincides with the intuition that v in G_2 has the same or more connections than in G_1 and, thus, its relevance in G_2 should be at least the one in G_1.

What property should a family of subgraphs $\mathcal{F}$ satisfy in order that $C_{\mathcal{F}}$ satisfy Axiom 1? Intuitively, when edge e is added to G we have $G \subseteq G + e$ and all subgraphs that are relevant for v in G should also be relevant for v in $G + e$. Moreover, if a subgraph S is relevant for v in $G + e$ but S is a subgraph of v in G, then it should also be a relevant subgraph of v in G. That is, all subgraphs of G that are relevant should also be relevant in $G + e$ and vice versa We call this the containment property.

▶ **Property 1** (Containment). *A family of subgraphs $\mathcal{F}$ satisfies the containment property if for every graphs G_1 and G_2 such that $G_1 \subseteq G_2$ and for every $v \in V(G_1)$ and $S \in \mathcal{A}(v, G_1)$, it holds that $S \in \mathcal{F}(v, G_1)$ if, and only if, $S \in \mathcal{F}(v, G_2)$.*

In particular, the containment property implies that $\mathcal{F}(v, G_1) \subseteq \mathcal{F}(v, G_2)$ whenever $G_1 \subseteq G_2$. As one could expect, the containment property is enough to satisfy the monotonicity axiom.

▶ **Theorem 8.** *If a family of subgraphs $\mathcal{F}$ satisfies the containment property, then the corresponding centrality measure $C_{\mathcal{F}}$ satisfies the monotonicity axiom.*

One can easily see that the family of all-subgraphs and trees satisfies the containment property and, therefore, the all-subgraphs centrality and trees centrality satisfy monotonicity as expected. Next, we show that this is not always the case.

▶ **Example 9.** Given a graph G and $v \in V(G)$, denote by $\mathcal{W}(v, G)$ all subgraphs $P \in \mathcal{A}(v, G)$ such that $P = v_0, \ldots, v_n$ is a geodesic path in G, namely, it is a path of minimal distance between v_0 and v_n. We assume here that the isolated vertex v is the only geodesic path from v to v. In [8], $|\mathcal{W}(v, G)|$ is defined as the stress centrality of vertex v. Then we define log-stress centrality of v in G as $C_{\mathcal{W}}(v, G)$. Of course, $C_{\mathcal{W}}(v, G)$ is not equivalent to Betweenness(v, G) as a value and in how we aggregate the number of geodesic paths. Nevertheless, it will be useful below to understand Betweenness in the context of counting subgraphs.

One can easily show that the family $\mathcal{W}$ does not satisfy the containment condition. Consider just a line $L_3 =$. Then if we connect the black vertices and make a triangle $K_3 =$, then the geodesic path is in $\mathcal{W}(1, L_3)$ but is not in $\mathcal{W}(1, K_3)$. Coincidentally, log-stress centrality (and betweenness centrality as well) do not satisfy the monotonicity axiom. Actually, one can show pathological examples where monotonicity does not hold [16]. For example, if one compares the circuit C_n with the clique K_n one can see that $C_n << K_n$ but $C_{\mathcal{W}}(0, C_n) > C_{\mathcal{W}}(0, K_n)$, and Betweenness$(0, C_n) >$ Betweenness$(0, K_n)$ as well.

It is important to note that, for some axiomatic approaches [28], it is desirable that the center of a star S_{n-1} is the most central node in a graph with n vertices, namely, a centrality measure C satisfies this axiom if, for any n and for any graph G with $|V(G)| = n$, it holds that $C(v, G) \leq C(0, S_{n-1})$ for every $v \in V(G)$. Unfortunately, this assumption contradicts the idea behind the monotonicity axiom, since we can add edges to S_{n-1} but the centrality of the center will never increase. Thus, given that this axiom contradicts the monotonicity axiom, we do not consider it in our analysis.

Rank monotonicity. Another axiom that has been remarked as important in the literature is rank monotonicity [5, 6, 11, 27]. Similar than for monotonicity, this axiom says that if v is more central than u in G, then when we add a new edge e to v the ranking between u and v is preserved. In particular, if v is the most central vertex in G, then it will be the most central vertex in $G + e$ as well. We generalize this intuition as follows.

▶ **Axiom 2** (Rank monotonicity). *A centrality measure C satisfies the rank monotonicity axiom if for every graph G, $u, v \in V(G)$ and $e \notin E(G)$ with $v \in e$, then $C(u, G) \leq C(v, G)$ implies that $C(u, G+e) \leq C(v, G+e)$.*

Note that with $e = \{u, v\}$ it could happen that the increment in centrality for u is bigger than the increment on v, but the axiom says that the centrality of v will be still bigger than the centrality of u. In other words, if I meet Donald Trump, my centrality will rise more than his centrality, however, Donald Trump will still be the president of US.

It is important to say that in [6] an axiom called *density axiom* was proposed, which is a special case of rank monotonicity. Specifically, take a clique K_n, a circuit C_n, and vertices $u \in V(K_n)$ and $v \in V(C_n)$. Then the density axiom says that if we connect u and v with an edge $e = \{u, v\}$, then $G = (K_n \cup C_n) + e$ satisfies $C(u, G) > C(v, G)$ for a centrality measure C. Intuitively, given that the neighborhood of u is more dense that in v, then its centrality should be bigger. One can see that if C satisfies monotonicity (i.e. vertices in K_n has more centrality than in C_n), then rank monotonicity implies the density axiom [6]. Therefore, we can see rank monotonicity as a generalization of the density axiom in [6].

The containment property is useful to imply rank monotonicity but it is not enough. One can easily find centrality measures that satisfies Axiom 1 but it does not satisfy Axiom 2 (see Example 11 below). For this, one needs a notion of "fairness" in the family of subgraphs. Intuitively, if S is a relevant subgraph for v in G and S contains a vertex u, then S should also be relevant for u in S.

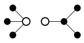
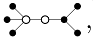
▶ **Property 2** (Fairness). *A family of subgraphs $\mathcal{F}$ satisfies the fairness property if for every graph G, $u, v \in V(G)$ and $S \subseteq G$ with $u, v \in V(S)$ it holds that $S \in \mathcal{F}(u, G)$ iff $S \in \mathcal{F}(v, G)$.*

As we show next, fairness is what you need if you want to preserve the ranking between vertices in a graph.

▶ **Theorem 10.** *If a family of subgraphs $\mathcal{F}$ satisfies the containment property and fairness, then the corresponding centrality measure $C_{\mathcal{F}}$ satisfies the rank monotonicity axiom.*

The family of all-subgraphs, trees and even betweenness (i.e. geodesic paths) satisfy fairness. Given that all-subgraphs and trees also satisfy the containment property, we conclude that both satisfy the rank monotonicity axiom. Next we show a natural family that satisfy the containment property but does not satisfy fairness.

▶ **Example 11.** A natural approach to define a family of subgraphs is to consider subpatterns on a neighborhood of bounded size around a vertex. Intuitively, an expert would not care if a vertex v can reach a far vertex u as long as there are many other substructures close to v. To formalize this, let $k \geq 1$. For a graph G, fix a vertex v and let N_k be the induced subgraph of all vertices at distance at most k of v, i.e., $V(N_k) = \{u \in V(G) \mid \text{dist}_G(u, v) \leq k\}$ and $E(N_k) = \{e \in E(G) \mid e \subseteq V(N_k)\}$. We define the family of subgraphs $\mathcal{N}_k$ such that $\mathcal{N}_k(v, G) = \mathcal{A}(v, N_k)$, that is, all subgraphs in the neighborhood of v with radius k. Then we define the k-neighborhood centrality of v on G as $C_{\mathcal{N}_k}(v, G)$. Note that if the diameter of the graph is less than k then $\mathcal{N}_k(v, G)$ and $\mathcal{A}(v, G)$ coincide.

The family of k-neighborhood satisfies monotonicity but it does not satisfy fairness. Moreover, it does not satisfy the rank monotonicity axiom. To see this, consider the family $\mathcal{N}_2$ and $G_1 =$ . By counting, one can check that the left white vertex, called u, and the right white vertex, called v, satisfy $|\mathcal{N}_2(u, G_1)| = 8$ and $|\mathcal{N}_2(v, G_1)| = 5$, respectively. Then $C_{\mathcal{N}_2}(u, G_1) > C_{\mathcal{N}_2}(v, G_1)$. However, if we add an edge e between the two and create the graph $G_1 + e =$ , then one can check that $\mathcal{N}_2$ does not satisfy fairness. For instance, the whole graph $G_1 + e \in \mathcal{N}_2(v, G_1 + e)$, contains u and v, but $G_1 + e \notin \mathcal{N}_2(u, G_1 + e)$. One can also check by counting that $|\mathcal{N}_2(u, G_1 + e)| = 24$ and $|\mathcal{N}_2(v, G_1 + e)| = 45$. Thus, $C_{\mathcal{N}_2}(u, G_1 + e) < C_{\mathcal{N}_2}(v, G_1 + e)$ and 2-neighborhood does not satisfy the rank monotonicity axiom as well.

The previous example shows that, if we want to approximate all-subgraphs centrality by only counting subgraphs up to a certain radius, one will have to loose some natural properties, like rank monotonicity.

Line minimization. Everyone would agree that any reasonable notion for centrality should assign 0 centrality to an isolated vertex [3, 28]. Basically, there is nothing less central to a community than the vertex that is not connected to any other vertex. One can generalize this idea by considering, what is the most sparse connected graph with n vertices. Clearly, the line L_n should be this graph: it is the only graph with n vertices that maximizes the diameter. Then the vertices that minimize the centrality in the line L_n are its extreme points, 0 and $n - 1$, and one would expect that this should be the vertices that have less centrality over all connected graphs with n-vertices.

▶ **Axiom 3** (Line minimization)**.** *A centrality measure C satisfies the line minimization axiom if for every n and every connected graph G with $|V(G)| = n$ it holds that $C(0, L_n) \leq C(v, G)$ for every $v \in V(G)$.*

All centralities that we consider in this paper satisfy the line minimization axiom. Of course, one can manage to find unnatural families of subgraphs that produce centrality measures not satisfying this axiom. Still, one would like to find under which circumstances a centrality measure defined from a family of subgraphs satisfies it. For this, we need to introduce the following property.

▶ **Property 3** (Inclusion)**.** *A family of subgraphs $\mathcal{F}$ satisfies the inclusion property if for every graph G, $v \in V(G)$, and $S \in \mathcal{F}(v, G)$, if $S' \subseteq S$ and $v \in V(S')$, then $S' \in \mathcal{F}(v, G)$.*

Intuitively, this property is saying that every subgraph of a relevant subgraph should also be relevant for the family. Actually, this property is satisfied by all families of subgraphs proposed so far.

▶ **Theorem 12.** *If a family of subgraphs $\mathcal{F}$ satisfies the containment and inclusion properties, then the corresponding centrality measure $C_{\mathcal{F}}$ satisfies the line minimization axiom.*

Continuity. The inclusion property plus the containment property actually imply a natural property over centrality measures defined by family of subgraphs. Given that all subgraphs of a relevant subgraph are also included, it gives a sense of "continuity" in the centrality notion. Specifically, each time that we add a set of edges that rises the centrality of a vertex, there exists a way to add them, one at a time, in such a way that the centrality of the vertex always increases. We formalize this intuition as follows.

▶ **Axiom 4** (Continuity)**.** *A centrality measure C satisfies the continuity axiom if for every graphs G and F, and $v \in V(G)$, if $C(v, G) < C(v, G \cup F)$, then there exists edges $e_1, \ldots, e_k \in E(F)$ such that: $C(v, G) < C(v, G + e_1) < \ldots < C(v, G + e_1 + \ldots + e_k) = C(v, G \cup F)$.*

To the best of our knowledge, the continuity axiom has not been proposed before in the literature. Furthermore, the inclusion and containment property implies the continuity axiom over the corresponding centrality measure.

▶ **Theorem 13.** *If a family of subgraphs $\mathcal{F}$ satisfies the inclusion and containment properties, then the corresponding centrality measure $C_{\mathcal{F}}$ satisfies the continuity axiom.*

All families of subgraphs so far satisfy the inclusion property and their corresponding centrality measures satisfy the continuity axiom as well. We give below a centrality measure based on cliques as a counter-example of this theorem.

▶ **Example 14.** Cliques are relevant substructure in network analysis and they are usually used to measure the importance of vertices [25]. In [14], this idea has been taken a step further by counting the number of cliques that a vertex belongs, which is called the cross-clique centrality. We can define this centrality with families of subgraphs as follows. Define the family $\mathcal{K}$ such that $\mathcal{K}(v, G)$ contains all subgraphs $K \in \mathcal{A}(v, G)$ such that K is a clique of size 1 (i.e. v) or size greater than 2 for every graph G and $v \in V(G)$. Then the clique centrality of v on G is defined as $C_{\mathcal{K}}(v, G)$. Note that $C_{\mathcal{K}}(v, G) = \log(\text{Cross-Clique}(v, G) + 1)$ and, thus, we can use $C_{\mathcal{K}}$ as a proxy to understand cross-clique centrality.

Cliques $\mathcal{K}$ is a family that does not satisfy the inclusion property. Indeed, any subgraph of a clique is not necessarily a clique. One can also check that its centrality $C_{\mathcal{K}}$ also does not satisfy the continuity property. For example, consider a single edge $G =$ where the white vertex v has clique centrality $C_{\mathcal{K}}(v, G) = 0$. Then, if a triangle $F =$ is added to G, producing the graph $G + F =$ with $C_{\mathcal{K}}(v, G + F) = 1$, there is no way to rise the centrality of v from 0 to 1 by adding the edges of the triangle one-by-one.

Size. The last axiom that we study here is the one proposed in [6] about size. This was formalized as follows: for any $n > 0$ if we consider clique K_n and a circuit C_n, for a centrality measure C one would expect that $C(0, K_n) > C(0, C_n)$. Then no matter how big is $C(0, K_n)$, there should exists a value $m > n$ where the centrality of the cycle C_m passes the centrality of the clique K_n, namely, $C(0, K_n) < C(0, C_m)$. This argument is related to the size of graphs in the sense that no matter how slow the centrality of C_m grows, at some point it should beat the clique of size n. We propose a generalization of this axiom as follows.

▶ **Axiom 5** (Size). *A centrality measure C satisfies the size axiom if for every infinite sequence $\{G_n\}_{0 \leq n}$ of connected graphs with $V(G_n) = \{0, \dots, n\}$ and for every value N there exists m such that $C(0, G_m) \geq N$.*

Here the sequence $\{G_n\}_{0 \leq n}$ is playing the role of the circuits and N the role of the centrality in the clique. Thus, if a centrality measures satisfies Axiom 5 then it satisfies the size axiom of [6], but the converse of course is not true.

This axiom is clearly satisfied by all-subgraphs and trees centrality. Indeed, by Proposition 6 we know that $C_{\mathcal{A}}(v, G)$ is always bounded below by $\log(n)$ and thus the all-subgraphs satisfy the axiom (similar argument can be given for trees centrality). Typical centrality measures that do not satisfy the size axiom are "local measures" that only consider subgraphs of bounded size, i.e., degree or k-neighborhood centrality. However, there are families of subgraphs of unbounded size that also do not satisfy this axiom, i.e., clique centrality. In both cases, if we consider the sequence of lines $\{L_n\}_{0 \leq n}$, we can see that the centrality on the vertex 0 is not growing and, thus, for a reasonable N the axiom does not hold. Actually, the next theorem shows that this counter-example is enough to show whether a centrality measure satisfy the size axiom or not.

▶ **Theorem 15.** *Let $\mathcal{F}$ be a family of subgraphs that satisfies the line minimization axiom. Then $C_{\mathcal{F}}$ satisfies the size axiom if, and only if, $\lim_{n \to \infty} |\mathcal{F}(0, L_n)| = \infty$.*

We remark that all centrality measures consider in this paper satisfies the line minimization axiom. Therefore, it is enough to check whether the family of subgraphs grows on the line to see whether the centrality notion is "local" or not.

We want to end this section by pointing out that in [6] it was shown that all standard notions of centrality in the literature (like closeness [4], betweenness [15], Page Rank [9], Katz index [20], etc) do not satisfy at least one of its axioms and, therefore, do not satisfy at least one of the general axioms stated above. This shows that all the standard notions for centrality studied in the literature are different with all-subgraphs centrality.

6 Extension to group centrality measures

Any natural centrality measure should come with a simple extension to measure the centrality of sets of vertices (also called *group centrality*). Although this is a desirable property, it is not always clear how to do it (i.e. not many centrality measures in the literature have a standard extension to group centrality). In this section, we embark on extending our families of centralities from vertices to sets and give a natural characterization for all-subgraphs group centrality. Towards the end, we show an application of this notion regarding the centrality maximization of a vertex.

Given an arbitrary family of subgraphs $\mathcal{F}$, what should be its extension to groups? A first approach is to consider all connected subgraphs in $\mathcal{F}$ that contains all elements in the group. Formally, given $U \subseteq V(G)$ one could consider the family of relevant subgraphs:

$$\mathcal{F}^*(U, G) \;=\; \{\, S \subseteq G \;\mid\; U \subseteq V(S) \,\wedge\, \exists v \in U.\; S \in \mathcal{F}(v, G)\,\}.$$

In other words, all relevant subgraphs of vertices in U that cover U. Although this is the direct extension for connected subgraphs, this definition rises two issues. First, some local families (e.g. k-neighborhood) could not keep the restriction of having all vertices in U inside a subgraph (i.e. $U \subseteq V(S)$). Moreover, if the size of U grows then there will be less subgraphs satisfying such restriction, making the definition impractical for some families of subgraphs. Second, sets that have more relevant subgraphs under this definition are likely to be closer in the graph. For example, if we look at the extension of all-subgraphs $\mathcal{A}^*(U, G)$, then in a circuit C_n a set U of k-vertices that has maximum centrality will be any set of k contiguous vertices. Clearly, if one looks for a central group of k-vertices in C_n, one would prefer a set of k-vertices that are equidistant in C_n because they cover more relevant structures of the graph as a group.

Given the previous discussion, we define the group extension of $\mathcal{F}$ to sets of vertices U on G, denoted as $\mathcal{F}(U, G)$, as follows:

$$\mathcal{F}(U, G) \;=\; \{\, S \subseteq G \;\mid\; U \subseteq V(S) \,\wedge\, \forall H \in \mathrm{ConnComp}(S).\; \exists v \in U.\; H \in \mathcal{F}(v, G)\,\}.$$

Note that this extension is similar to the one discussed above (i.e. $\mathcal{F}^*(U, G)$), but we asked that each connected component from S comes from a relevant subgraph of a vertex in U. This allows to use disconnected subgraphs to cover U and, at the same time, each connected component comes from connected subgraphs in $\mathcal{F}$. Unlike our first extension, this definition is not local anymore and gives meaningful results for any set U. In particular, when $U = \{v\}$ this definition generalizes the family of subgraphs for vertices given that $\mathcal{F}(U, G) = \mathcal{F}(v, G)$.

With a family of subgraphs for sets of vertices, it is natural to develop its corresponding group centrality. Similar than for vertices, given a set $U \subseteq V(G)$ from a graph G, we define the $\mathcal{F}$-group centrality measure of U in G as the worst-case entropy of $\mathcal{F}(U, G)$, namely:

$$C_{\mathcal{F}}(U, G) \;=\; \log\left(|\mathcal{F}(U, G)|\right).$$

All families introduced in previous sections have a corresponding group centrality measure. From now, we restrict our analysis to the all-subgraphs family and its centrality over groups, and leave the understanding of other families for future work.

▶ **Example 16.** Let C_n be a circuit of length $n \geq 3$ and consider all sets $U \subseteq V(C_n)$ of two vertices. Then one can check that the set U that maximizes $C_{\mathcal{A}}(U, C_n)$ is any pair of vertices that are at distance $\frac{n}{2}$ (assuming n even). Furthermore, if U are sets of k vertices with k a factor of n, then $C_{\mathcal{A}}(U, C_n)$ is maximized when all vertices in U are distributed in C_n with equal distance. Intuitively, this is the best way of covering a circuit C_n with k vertices.

Next we show that all-subgraphs group centrality over U can be reduced to computing the centrality of a vertex. Recall that we denote by G/U the set contraction of U on G, namely, to merge the vertices U to one vertex and keeping multi-edges into U (see Section 2). In particular, recall that U is a vertex in the multigraph G/U.

▶ **Theorem 17.** *Let G be a graph and $U \subseteq V(G)$. Then:*

$$C_{\mathcal{A}}(U, G) \;=\; C_{\mathcal{A}}(U, G/U) \;+\; |\{e \in E(G) \mid e \subseteq U\}|$$

The all-subgraphs group centrality of a set U in G is then reduced to the centrality of U (i.e. as a vertex) in the set-contraction of U on G plus the number of edges between vertices in U. Note that, in particular, this shows that if we look for k-sets of high centrality, then the all-subgraphs centrality is balancing between the number of edges of the set (i.e. how similar is the set to a clique) versus how central it is if we contract it into a vertex.

This connection between both definitions (i.e. vertices and sets) for all-subgraphs centrality is strictly related to the properties of the family. Given two subgraphs G_1 and G_2 of G with $V(G_1) \cap V(G_2) \neq \emptyset$, we can generate a new subgraph $G_1 \cup G_2$ by merging the nodes they share. Unfortunately, this is not possible for all families like the family of trees $\mathcal{T}$, that is, the union of two trees is not necessarily in $\mathcal{T}$. This means that Theorem 17 cannot be directly extended for families like trees, in particular, for trees centrality.

To end this section, we show an example how the all-subgraphs group centrality allows us to study simple questions regarding the maximization of the centrality of a vertex. Given a graph G and a vertex $v \in V(G)$, with whom should we connect v in G in order to maximize its centrality? In other words, if I am in a social network, with whom should I connect in order to maximize my centrality? A naive answer to this question is to connect v to the most central vertex in G. Actually, from the perspective of all-subgraphs centrality this is not the right answer: connecting to the most central node will rise its centrality but maybe the centrality of the most central vertex is highly dependent of v's centrality. Instead, all-subgraphs centrality says that v must be connected to the vertex u where $\{u, v\}$ (as a group) is more central in G.

▶ **Theorem 18.** *Given G and $v \in V(G)$ with $\{u \in V(G) \mid \{u, v\} \notin E(G)\} \neq \emptyset$, it holds that:*

$$\arg\max_{u \in V(G)} C_{\mathcal{A}}(v, G + \{v, u\}) \;=\; \arg\max_{\{u,v\} \notin E(G)} C_{\mathcal{A}}(\{v, u\}, G)$$

7 On computing centrality measures based on subgraphs

We study here the problem of computing centrality measures based on subgraphs. In particular, we study the problem of computing the all-subgraphs centrality. We state the problem as follows: given a family of subgraphs $\mathcal{F}$, consider the problem

Problem:	$\textsc{Count}(\mathcal{F})$
Input:	A graph G and a vertex $v \in V(G)$
Output:	$\lvert\mathcal{F}(v, G)\rvert$

Furthermore, given a class of graphs $\mathcal{G}$ we write $\textsc{Count}(\mathcal{F})[\mathcal{G}]$ for the parametrized version of $\textsc{Count}(\mathcal{F})$ when input graph G is restricted to $\mathcal{G}$. Of course, given a family $\mathcal{F}$ computing its centrality $C_{\mathcal{F}}$ requires also taking the logarithm to the output of $\textsc{Count}(\mathcal{F})$. Although these are not the same problems, the conclusions obtained here sheds light on the pitfalls of computing a centrality based on a family $\mathcal{F}$.

We start by giving an algorithm for computing $\textsc{Count}$ over all-subgraphs $\mathcal{A}$. Algorithm 1 shows a simple recursive algorithm for counting all connected subgraphs that contains a vertex $v \in V(G)$ in a (multi)graph G. The main idea is indeed very simple. Recall that $N(v, G)$ denotes the neighborhood of v in G (see Section 2). If $N(v, G) = \emptyset$, the vertex v is an isolated vertex and there is exactly one subgraph. Otherwise, v is connected to at least one vertex, called it $u \in N(v, G)$, and by some edge $e = \{u, v\}$. Then we can partition the set of connected subgraphs $\mathcal{A}(v, G)$ into those that u and v are directly connected by some edge, and those that are not. For the former, we can compute the exact number recursively as $\textsc{CountAll}(G - e, v)$ (recall here that $G - e$ contains no edges between u and v). For the latter, let $w(e)$ be the number of edges between u and v in G (recall that G could be a multigraph). Then all connected subgraphs where u and v are directly connected by some edge can be formed by choosing a non-empty set of edges between u and v (i.e. $2^{w(e)} - 1$ many possibilities) plus a connected subgraph from $\mathcal{A}(e, G/e)$ where G/e is the set contraction of e on G (i.e. $\textsc{CountAll}(G/e, e)$ many possibilities). Therefore, we can compute $\textsc{CountAll}(G, v)$ by recursively computing $\textsc{CountAll}(G - e, v)$ and $\textsc{CountAll}(G/e, e)$. In both cases, the number of edges or the number of vertices is reduced, and $\textsc{CountAll}$ will eventually finish.

Although Algorithm 1 is easy to implement, it could take exponential time in the number of edges. Actually, this is the best that one can hope as we show in the next result. Recall that #P is the class of counting problems that can be defined as counting the number of accepting runs of a polynomial-time non-deterministic Turing machine. Further, a counting problem is #P-complete if it is in #P and all counting problems in #P can be reduced to it [30]. It is known that a polynomial-time algorithm for solving a #P-complete problem, if it existed, would imply that P = NP. For this reason, #P-complete is a class of counting problems considered as hard [2].

▶ **Theorem 19.** *$\textsc{Count}(\mathcal{A})$ and $\textsc{Count}(\mathcal{T})$ are #P-complete.*

This is a negative result for using all-subgraphs centrality or trees centrality in practice. Nevertheless, we believe that this should not overshadow the impact that both measures can have in defining good centrality notions. As we show in Section 5, both notions behaved well as centrality measures and, although they are difficult to compute, they can still be used, for example, to guide the definition of new centrality measures or to design new efficient algorithms for computing the most relevant vertices in a graph.

Given that computing all-subgraphs over any graph is a difficult problem, our next step is to consider classes of graphs $\mathcal{G}$ where $\textsc{Count}(\mathcal{F})[\mathcal{G}]$ can be solved efficiently. A natural class to start here are trees. Indeed, when G is a tree every internal vertex is a cut-vertex and we can use the ideas of Lemma 3 for computing $|\mathcal{A}(v, G)|$ efficiently. More specific, from Lemma 3 one can show that if v is a cut-vertex of a graph G and $G_1, \ldots, G_n$ are subgraphs that partitions G on v (i.e. $V(G) = \cup_{i=1}^{n} V(G_i)$, $E(G) = \cup_{i=1}^{n} E(G_i)$, and

Algorithm 1 All-subgraphs counting.

```
Require: A graph $G$ and vertex $v \in V(G)$
procedure CountAll($G$, $v$)
    if $N(v, G) = \emptyset$ then
        return 1
    else
        let $u \in N(v, G)$
        $e \leftarrow \{u, v\}$
        return CountAll($G - e, v$) +
                $(2^{w(e)} - 1) \cdot$ CountAll($G/e, e$)
```

Algorithm 2 All-subgraphs on trees.

```
Require: A tree $T$ and vertex $v \in V(T)$
procedure CountTrees($T$, $v$)
    if $N(v, T) = \emptyset$ then
        return 1
    else
        let $u \in N(v, T)$
        $e \leftarrow \{u, v\}$
        return CountTrees($T - e, v$) $\cdot$
                (CountTrees($T - e, u$) + 1)
```

$V(G_i) \cap V(G_j) = \{v\}$ for $i \neq j$), then $\mathcal{A}(v, G) = \prod_{i=1}^{n} \mathcal{A}(v, G_i)$. We can exploit this in a tree by considering all subtrees $T_1, \ldots, T_n$ hanging from v and computing $\mathcal{A}(v, G)$ as the product of $\mathcal{A}(v, T_i)$.

In the procedure CountTrees of Algorithm 2 we use the previous idea for computing $|\mathcal{A}(v, T)|$ when T is a tree and $v \in V(T)$. It follows a similar approach to that in Algorithm 1. First, if v is an isolated vertex (i.e. $N(v, T) = \emptyset$), then it outputs 1. Otherwise, it takes a vertex $u \in N(v, T)$, defines the edge $e = \{u, v\}$, and decompose T in two subtrees by removing e from the graph. Notice that, if we remove e from T, we create two connected components T_v and T_u, where T_v and T_u contains v and u, respectively. One can easily check that T_v and $T_u + e$ partitions T on v and we have $|\mathcal{A}(v, T)| = |\mathcal{A}(v, T_v)| \cdot |\mathcal{A}(v, T_u + e)|$ by the previous discussion above. Furthermore, it is straightforward to check that $|\mathcal{A}(v, T_v)| = |\mathcal{A}(v, T - e)|$ and $|\mathcal{A}(v, T_u + e)| = |\mathcal{A}(u, T - e) + 1|$. Thus, we can compute $|\mathcal{A}(v, T)|$ by recursively computing CountTrees($T - e, v$) multiplied by CountTrees($T - e, u$) + 1.

In contrast to Algorithm 1, the recursion in CountTrees separates the graph in two disjoint subtrees. This implies that the recursion eventually finishes and, moreover, it takes linear time in the size of the tree. Interestingly, we can extend this idea to any graph of bounded tree-width. To formalize the notion of bounded tree-width, we need to introduce some notation. Given a graph G, a tree decomposition T of G is a tree such that $V(T)$ are sets of $V(G)$ (i.e. $X \subseteq V(G)$ for every $X \in V(T)$) and satisfies the following three properties: (1) $V(G) = \bigcup_{X \in V(T)} X$, (2) if $v \in X \cap Y$ for $X, Y \in V(T)$, then $v \in Z$ for all $Z \in V(T)$ in the simple path from X to Y in T, and (3) for every $e \in E(G)$, there exists $X \in V(T)$ such that $e \subseteq X$. The width of a tree decomposition T is equal to $\max_{X \in V(T)} |X| - 1$ and the tree-width $\text{tw}(G)$ of G is the minimum width among all possible tree decompositions of G [26]. A class $\mathcal{G}$ of graphs has bounded tree width if there exists a uniform bound k such that $\text{tw}(G) \leq k$ for every $G \in \mathcal{G}$. For example, all trees is a class that has tree-width bounded by 1.

▶ **Theorem 20.** *If $\mathcal{G}$ has bounded tree-width, then* Count$(\mathcal{A})[\mathcal{G}]$ *can be solved in* PTime.

The previous result shows that the problem becomes tractable when graphs has bounded tree-width. Despite that graphs have high tree-width in practice [23], this result gives some clues on how to tackle the problem of computing the all-subgraphs centrality.

8 Future work

This work arises several research opportunities regarding centrality measures based on subgraphs, which are briefly discussed here. One of the most important question is whether all-subgraphs centrality can be approximated efficiently, or even if the rank order given by this measure can be approximated. Another interesting question is to consider when a family of graphs can approximate another family over some particular class of graphs (e.g. plain graphs). For the sake of simplification, we only considered undirected graphs but another relevant question is to study how to extend these results to directed graphs or to hypergraphs. Furthermore, the initial motivation of our approach came from centrality measures for graph query languages, but in order to incorporate this approach, several properties must be understood like, for example, how to mix the centrality measures to the output of a query. Finally, it would be interesting to consider a randomized version of our approach where not all subgraphs have the same chances to appear. Instead of considering the worst-case entropy, one could study the entropy of a family given a particular distribution and study their properties. We leave this and other questions for future work.

References

1 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

3 Sambaran Bandyopadhyay, Ramasuri Narayanam, and M Narasimha Murty. A Generic Axiomatic Characterization for Measuring Influence in Social Networks. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2606–2611. IEEE, 2018.

4 Alex Bavelas. Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

5 Paolo Boldi, Alessandro Luongo, and Sebastiano Vigna. Rank monotonicity in centrality measures. *Network Science*, 5(4):529–550, 2017.

6 Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

7 Stephen P Borgatti and Martin G Everett. A graph-theoretic perspective on centrality. *Social networks*, 28(4):466–484, 2006.

8 Ulrik Brandes. *Network analysis: methodological foundations*, volume 3418. Springer Science & Business Media, 2005.

9 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

10 Carlos Buil-Aranda, Martın Ugarte, Marcelo Arenas, and Michel Dumontier. A preliminary investigation into SPARQL query complexity and federation in Bio2RDF. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, page 196, 2015.

11 Steve Chien, Cynthia Dwork, Ravi Kumar, Daniel R Simon, and D Sivakumar. Link evolution: Analysis and algorithms. *Internet mathematics*, 1(3):277–304, 2004.

12 Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

13 Ernesto Estrada and Juan A Rodriguez-Velazquez. Subgraph centrality in complex networks. *Physical Review E*, 71(5):056103, 2005.

14 Mohammad Reza Faghani and Uyen Trang Nguyen. A study of XSS worm propagation and detection mechanisms in online social networks. *IEEE transactions on information forensics and security*, 8(11):1815–1826, 2013.

15 Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

16 Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

17 Manuj Garg. Axiomatic foundations of centrality in networks. *Available at SSRN 1372441*, 2009.

18 Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. Hypertree Decompositions: Questions and Answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 57–74, 2016.

19 Hawoong Jeong, Sean P Mason, A-L Barabási, and Zoltan N Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41, 2001.

20 Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

21 Harold J Leavitt. Some effects of certain communication patterns on group performance. *The Journal of Abnormal and Social Psychology*, 46(1):38, 1951.

22 Johannes Lorey and Felix Naumann. Detecting SPARQL query templates for data prefetching. In *Extended Semantic Web Conference*, pages 124–139. Springer, 2013.

23 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 12:1–12:18, 2019.

24 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

25 Mark Newman. *Networks: an introduction*. Oxford university press, 2010.

26 Neil Robertson and Paul D Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

27 Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

28 Oskar Skibski, Talal Rahwan, Tomasz P Michalak, and Makoto Yokoo. Attachment centrality: An axiomatic approach to connectivity in networks. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 168–176. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

29 Oskar Skibski and Jadwiga Sosnowska. Axioms for distance-based centralities. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

30 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

31 Tomasz Wąs and Oskar Skibski. An axiomatization of the eigenvector and Katz centralities. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Reverse Prevention Sampling for Misinformation Mitigation in Social Networks

Michael Simpson
Department of Computer Science, University of Victoria, Canada
simpsonm@uvic.ca

Venkatesh Srinivasan
Department of Computer Science, University of Victoria, Canada
srinivas@uvic.ca

Alex Thomo
Department of Computer Science, University of Victoria, Canada
thomo@uvic.ca

Abstract

In this work, we consider misinformation propagating through a social network and study the problem of its prevention. In this problem, a "bad" campaign starts propagating from a set of seed nodes in the network and we use the notion of a limiting (or "good") campaign to counteract the effect of misinformation. The goal is to identify a set of k users that need to be convinced to adopt the limiting campaign so as to minimize the number of people that adopt the "bad" campaign at the end of both propagation processes.

This work presents *RPS* (Reverse Prevention Sampling), an algorithm that provides a scalable solution to the misinformation prevention problem. Our theoretical analysis shows that *RPS* runs in $O((k+l)(n+m)(\frac{1}{1-\gamma})\log n/\epsilon^2)$ expected time and returns a $(1-1/e-\epsilon)$-approximate solution with at least $1-n^{-l}$ probability (where γ is a typically small network parameter and l is a confidence parameter). The time complexity of *RPS* substantially improves upon the previously best-known algorithms that run in time $\Omega(mnk \cdot POLY(\epsilon^{-1}))$. We experimentally evaluate *RPS* on large datasets and show that it outperforms the state-of-the-art solution by several orders of magnitude in terms of running time. This demonstrates that misinformation prevention can be made practical while still offering strong theoretical guarantees.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Graph Algorithms, Social Networks, Misinformation Prevention

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.24

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.01162`.

1 Introduction

Social networks allow for widespread distribution of knowledge and information in modern society as they have rapidly become a place to hear the news and discuss social topics. Information can spread quickly through the network, eventually reaching a large audience, especially so for influential users. While the ease of information propagation in social networks can be beneficial, it can also have disruptive effects. In recent years, the number of high profile instances of misinformation causing severe real-world effects has risen sharply. These examples range across a number of social media platforms and topics [9, 23, 11, 13, 29, 1]. Thus, in order for social networks to serve as a reliable platform for disseminating critical information, it is necessary to have tools to limit the spread of misinformation.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 24; pp. 24:1–24:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Budak et al. [4] were among the first to formulate the problem of misinformation prevention as a combinatorial optimization problem. By building upon the seminal work of Kempe et al. [16] on *influence maximization* to a model that can handle multiple campaigns ("bad" and "good"), they present a greedy approach that provides a $(1 - 1/e - \epsilon)$-approximate solution. Unfortunately, the greedy approach of [4] is plagued by the same scaling issues as [16] when considering large social networks and is further exacerbated by the added complexity of tracking multiple cascades which requires costly shortest path computations. This leads us to the motivating question for this paper: Can we find scalable algorithms for the misinformation prevention problem introduced in [4]?

The scalability hurdle in the single campaign setting was recently resolved by Borgs et al. [3] when they made a theoretical breakthrough that fundamentally shifts the way in which we view the influence maximization problem. Their key insight was to reverse the question of "what subset of the network can a particular user influence" to "who could have influenced a particular user". Their sampling method runs in close to linear time and returns a $(1 - 1/e - \epsilon)$-approximate solution with at least $1 - n^{-l}$ probability. In addition, Tang et al. [30] presented a significant advance that improved the practical efficiency of Borgs et al. through a careful theoretical analysis that rids their approach of a large hidden constant in the runtime guarantee. Borgs et al. [3] leave open the question whether their framework can be extended to other influence propagation models.

In this work, we resolve the question of [3] for the misinformation prevention problem and achieve scalability in the multi-campaign model. We complement our theoretical analysis with extensive experiments which show an improvement of several orders of magnitude over Budak et al. [4]. Since influence in the single campaign setting corresponds to reachability in the network, our solution requires mapping the concept of reachability to an analogous notion in the multi-campaign model for misinformation prevention. Our first contribution is to show that reachability alone is not sufficient in determining the ability to save a particular node from the bad campaign. In order to address this challenge, we introduce a crucial notion of "obstructed" nodes, which are nodes such that all paths leading to them can be blocked by the bad campaign.

Using our newly defined notion of obstruction, we develop an efficient algorithm for the misinformation prevention problem that provides much improved scalability over the existing Monte Carlo-based greedy approach of [4]. A novel component of this algorithm is a procedure to compute the set of unobstructed nodes that could have saved a particular node from adopting the misinformation. We obtain theoretical guarantees on the expected runtime and solution quality for our new approach and show that its expected runtime substantially improves upon the expected runtime of [4]. Additionally, we rule out sublinear algorithms for our problem through a lower bound on the time required to obtain a constant approximation.

Finally, from an experimental point of view, we show that our algorithm gives a significant improvement over the state of the art algorithm and can efficiently handle graphs with more than 50 million edges. In summary, the contributions of this paper are:

1. We introduce the concept of *obstructed* nodes that fully captures the necessary conditions for preventing the adoption of misinformation in the multi-campaign model. In the process, we close a gap in the work of [4].
2. We design and implement a novel procedure for computing the set of nodes that could save a particular user from adopting the misinformation.
3. We propose a misinformation prevention approach that returns a $(1-1/e-\epsilon)$-approximate solution with high probability in the multi-campaign model and show that its expected runtime substantially improves upon that of the algorithm of Budak et al. [4].

4. We give a lower bound of $\Omega(m+n)$ on the time required to obtain a constant approximation for the misinformation prevention problem.
5. Our experiments show that our algorithm gives an improvement of several orders of magnitude over Budak et al. [4] and can handle graphs with more than 50 million edges.

2 Related Work

There exists a large body of work on the Influence Maximization problem first proposed by Kempe et al. [16]. The primary focus of the research community has been related to improving the practical efficiency of the Monte Carlo-based greedy approach under the Independent Cascade (IC) or Linear Threshold (LT) propagation models. These works fall into two categories: heuristics that trade efficiency for approximation guarantees [15, 32] and practical optimizations that speed up the Monte Carlo-based greedy approach while retaining the approximation guarantees [18, 6, 10]. Despite these advancements, it remains infeasible to scale the Monte Carlo-based approach to web-scale networks.

Borgs' et al. [3] brought the first asymptotic runtime improvements while maintaining the $(1 - 1/e - \epsilon)$-approximation guarantees with their *reverse influence sampling* technique. Furthermore, they prove their approach is near-optimal under the IC model. Tang et al. [30] presented practical and theoretical improvements to the approach and introduced novel heuristics that result in up to 100-fold improvements to the runtime.

Incorporating the spread of multiple campaigns is split between two main lines of work: (1) studying influence maximization in the presence of competing campaigns [2, 20, 24, 19] and (2) limiting the spread of misinformation and rumours by launching a truth campaign [4, 14, 7, 22]. In both cases, existing propagation models (such as IC and LT) are augmented or extended. The work of [4] best captures the idea of preventing the spread of misinformation in a multi-campaign version of the IC model since they aim to minimize the number of users that end up adopting the misinformation. Unfortunately, despite the objective function proving to be monotone and submodular, the Monte Carlo-based greedy solution used in [4] faces the same challenges surrounding scalability as [16].

Works [20, 8] extend the *reverse influence sampling* technique of [3] to competing campaigns (such as two competing products in [20] and spreading truth to combat misinformation in [8]). However, their work differs from ours in an important way: they use a model, different from ours, where the edge probabilities are *campaign oblivious*. This alternative model does not capture the notion of misinformation as well as the model we use, but instead is better suited for the influence maximization problem when there are multiple competing campaigns (see [4] for a discussion).

Finally, the misinformation problem has been tackled by a wide range of communities such as [17, 26, 25, 12, 31, 27].

3 Preliminaries

In this section, we formally define the multi-campaign diffusion model, the eventual influence limitation problem presented by Budak et al. [4], and present an overview of the state-of-the-art reverse sampling approach [16, 3, 30] for the influence maximization problem.

Diffusion Model

Let C (for "bad Campaign") and L (for "Limiting") denote two influence campaigns. Let $\mathcal{G} = (V, E, p)$ be a social network with node set V and directed edge set E ($|V| = n$ and $|E| = m$) where p specifies campaign-specific pairwise influence probabilities (or weights) between nodes. That is, $p : E \times Z \rightarrow [0,1]$ where $Z \in \{C, L\}$. For convenience, we use $p_Z(e)$ for $p(e, Z)$. Further, let $G = (V, E)$ denote the underlying unweighted directed graph. Given $\mathcal{G}$, the Multi-Campaign Independent Cascade model (MCIC) of Budak et al. [4] considers a time-stamped influence propagation process as follows:

1. At timestamp 1, we *activate* selected sets A_C and A_L of nodes in $\mathcal{G}$ for campaigns C and L respectively, while setting all other nodes *inactive*.
2. If a node u is first activated at timestamp i in campaign C (or L), then for each directed edge e that points from u to an inactive neighbour v in C (or L), u has $p_C(e)$ (or $p_L(e)$) probability to activate v at timestamp $i+1$. After timestamp $i+1$, u cannot activate any node.
3. In the case when two or more nodes from different campaigns are trying to activate v at a given time step we assume that the "good information" (i.e. campaign L) takes effect.
4. Once a node becomes activated in one campaign, it never becomes inactive or changes campaigns.

He et al. [14] consider the opposite policy to (3) where the misinformation succeeds in the case of a tie-break. We note that our algorithms presented in this work are applicable for both choices of the tie-break policy.

3.1 Formal Problem Statement

A natural objective, as outlined in [4], is "saving" as many nodes as possible. That is, we seek to minimize the number of nodes that end up adopting campaign C when the propagation process is complete. This is referred to as the *eventual influence limitation problem (EIL)*.

Let A_C and A_L be the set of nodes from which campaigns C and L start, respectively. Let $I(A_C)$ be the set of nodes that are activated in campaign C in the absence of L when the above propagation process converges and $\pi(A_L)$ be the size of the subset of $I(A_C)$ that campaign L prevents from adopting campaign C. We refer to A_L and A_C as the *seed sets*, $I(A_C)$ as the *influence* of campaign C, and $\pi(A_L)$ as the *prevention* of campaign L. The nodes that are prevented from adopting campaign C are referred to as *saved*. Note that $\pi(A_L)$ is a random variable that depends on the edge probabilities that each node uses in determining out-neighbors to activate.

Budak et al. [4] present a simplified version of the problem that captures the idea that it may be much easier to convince a user of the truth. Specifically, the information from campaign L is accepted by users with probability 1 ($p_L(e) = 1$ if edge e exists and $p_L(e) = 0$ otherwise) referred to as the *high effectiveness property*. In [4] it is shown that even with these restrictions EIL with the high effectiveness property is NP-hard. Interestingly, with the high effectiveness property, the prevention function is submodular and thus a Monte Carlo-based greedy approach (referred to here as *MCGreedy*) yields approximation guarantees.

We motivate the high effectiveness property with the following two real-world scenarios: (1) the phenomenon of "death hoaxes" (where celebrities or other notable figures are claimed to have died) have a strong corrective measure when the victim, or a close relative, makes an announcement on their personal account that contradicts the rumour and (2) false reporting of natural disasters can be countered by trusted news organizations providing coverage of the location of the purported scene. In both cases, the sharing of links to strong video, photographic, or text evidence that is also coming from a credible source lends itself to a scenario following the high effectiveness property. In addition to the scenarios we have outlined, the model is attractive because this assumption leads to interesting theoretical guarantees. Budak et al. study and obtain results for EIL with the high effectiveness property and is the problem that we consider in this work.

▶ **Problem 1.** *Given $\mathcal{G}$, seed set A_C, and a positive integer k, the eventual influence limitation (EIL) problem asks for a size-k seed set A_L maximizing the value of $\mathbb{E}[\pi(A_L)]$ under the MCIC model with the high effectiveness property.*

Possible Worlds Interpretation

To facilitate a better understanding of MCIC, we define a *Possible World (PW) model* that provides an equivalent view of the MCIC model and follows a widely used convention when studying IM and related problems [16, 4, 6, 10, 20, 14, 4, 7, 22]. Given a graph $\mathcal{G} = (V, E, p)$ and the MCIC diffusion model, a possible world X consists of two *deterministic graphs*, one for each campaign, sampled from a probability distribution over $\mathcal{G}$. The stochastic diffusion process under the MCIC model has the following equivalent description: we can interpret $\mathcal{G}$ as a distribution over unweighted directed graphs, where each edge e is independently realized with probability $p_C(e)$ (or $p_L(e)$). Observe, given the high effectivness property, the deterministic graph that defines the possible world for campaign L is simply the underlying unweighted graph G. Then, if we realize a graph g according to the probability distribution given by $p_C(e)$, we can associate the set of saved nodes in the original process with the set of nodes which campaign L reaches before campaign C during a *deterministic* diffusion process in $g \sim \mathcal{G}$ by campaign C and in G by campaign L. That is, we can compute the set of saved nodes with a deterministic cascade in the resulting possible world $X = (g, G)$. The following theorem from [5] establishes the equivalence between this possible world model and MCIC. This alternative PW model formulation of the EIL problem under the MCIC model will be used throughout the paper.

▶ **Theorem 1** ([5]). *For any fixed seed sets A_C and A_L, the joint distributions of the sets of C-activated nodes and L-activated nodes obtained (i) by running a MCIC diffusion from A_C and A_L and (ii) by randomly sampling a possible world $X = (g, G)$ and running a deterministic cascade from A_C in g and A_L in G, are the same.*

3.2 Reverse Sampling for Influence Maximization

In this section we review the state-of-the-art approach to the well studied *influence maximization problem (IM)*. This problem is posed in the popular Independent Cascade model (IC) which, unlike the MCIC model, only considers a single campaign. The goal here is to compute a seed set S_{IM} of size k that maximizes the influence of S_{IM} in $\mathcal{G}$. In a small abuse of notation, this section refers to a possible world as the single deterministic graph $g \sim \mathcal{G}$ where each edge in $\mathcal{G}$ is associated with a single influence probability $p(e)$.

Borgs et al. [3] were the first to propose a novel method for solving the IM problem under the IC model that avoids the limitations of the original Monte Carlo-based solution [16]. Their approach, which was later refined by Tang et al. [30], is based on the concept of *Reverse Reachable (RR) sets* and is orders of magnitude faster than the greedy algorithm with Monte Carlo simulations, while still providing approximation guarantees with high probability. We follow the convention of [30] and refer to the method of [3] as *Reverse Influence Sampling (RIS)*. To explain how *RIS* works, Tang et al. [30] introduce the following definitions:

▶ **Definition 1** (Reverse Reachable Set). *The reverse reachable set for a node v in $g \sim \mathcal{G}$ is the set of nodes that can reach v. (That is, for each node u in the RR set, there is a directed path from u to v in g.)*

▶ **Definition 2** (Random RR Set). *A random RR set is an RR set generated on an instance of $g \sim \mathcal{G}$, for a node selected uniformly at random from g.*

Note, a random RR set encapsulates two levels of randomness: (i) a deterministic graph $g \sim \mathcal{G}$ is sampled where each edge $e \in E$ is independently removed with probability $(1-p(e))$, and (ii) a "root" node v is randomly chosen from g. The connection between RR sets and node activation is formalized in the following crucial lemma.

▶ **Lemma 1.** [3] *For any seed set S and node v, the probability that an influence propagation process from S can activate v equals the probability that S overlaps an RR set for v.*

Based on this result, the *RIS* algorithm runs in two steps:

1. Generate random RR sets from $\mathcal{G}$ until a threshold on the total number of steps taken has been reached.
2. Consider the maximum coverage problem of selecting k nodes to cover the maximum number of RR sets generated. Use the standard greedy algorithm for the problem to derive a $(1-1/e)$-approximate solution S_k^*. Return S_k^* as the seed set to use for activation.

The rationale behind *RIS* is as follows: if a node u appears in a large number of RR sets it should have a high probability to activate many nodes under the IC model; hence, u's expected influence should be large. As such, we can think of the number of RR sets u appears in as an estimator for u's expected influence. By the same reasoning, if a size-k node set S_k^* covers most RR sets, then S_k^* is likely to have the maximum expected influence among all size-k node sets in $\mathcal{G}$ leading to a good solution to the IM problem. As shown in [30], Lemma 1 is the key result that underpins the approximation guarantees of *RIS*. The main contribution of Borgs et al. is an analysis of their proposed threshold-based approach: *RIS* generates RR sets until the total number of nodes and edges examined during the generation process reaches a pre-defined threshold Γ. Importantly, Γ must be set large enough to ensure a sufficient number of samples have been generated to provide a good estimator for expected influence. They show that when Γ is set to $\Theta((m+n)k \log n/\epsilon^2)$, *RIS* runs in near-optimal time $O((m+n)k \log n/\epsilon^2)$, and it returns a $(1-1/e-\epsilon)$-approximate solution to the IM problem with at least constant probability.

Due to the more complex dynamics involved in propagation under the MCIC model, adapting the reverse sampling approach to solve EIL is far from trivial.

4 New Definitions

In this section we introduce new definitions that are crucial to the development of our approach. In particular, we formalize the notion of *obstructed* nodes which is required to capture the necessary conditions for saving a node.

Identifying Saved Nodes

Given set A_L of vertices and (unweighted) directed graph $g \sim \mathcal{G}$, write $cl_g(A_L)$ for the set of nodes closer to A_L in G than to A_C in g. That is, a node $w \in cl_g(A_L)$ if there exists a node v such that $v \in A_L$ and $|SP_G(v,w)| \leq |SP_g(A_C,w)|$ where $SP_H(v,w)$ denotes a shortest path from node v to w in graph H and $SP_H(S,w)$ for a set S denotes the shortest path from any node $v \in S$ to w in graph H. When g is drawn from $\mathcal{G}$ this is a necessary, but not sufficient[1], condition for the set of nodes *saved* by A_L. We also require that the nodes in $cl_g(A_L)$ not be *obstructed* by the diffusion of campaign C in g.

[1] In Budak et al.'s work, the set of nodes closer to A_L than A_C is established as a necessary and sufficient condition to *save* a node in the MCIC model, but we note that this should be revised to include our *obstructed* condition due to a gap in the proof of Claim 1 in [4].

Table 1 Frequently used notation.

Notation	Description
$\mathcal{G}$	a social network represented as a weighted directed graph $\mathcal{G}$
G, G_T	the underlying unweighted graph G and its transpose G_T constructed by reversing the direction of each edge
g	a possible world for campaign C obtained by sampling each edge $e \in \mathcal{G}$ independently with probability $p_C(e)$
n, m	the number of nodes and edges in $\mathcal{G}$ respectively
k	the size of the seed set for misinformation prevention
C, L	the misinformation campaign C and the limiting campaign L
$p_C(e)$, $p_L(e)$	the propagation probability on an edge e for campaigns C and L respectively
$\pi(S)$	the prevention of a node set S in a misinformation propagation process on $\mathcal{G}$ (see Section 4)
$\omega(R)$, $\omega_\pi(R)$	the number of edges considered in generating an RRC set and that originate from nodes in an RRC set R (see Equation 3)
$\mathcal{R}$	the set of all RRC sets generated by Algorithm 1
$\mathcal{F}_\mathcal{R}(S)$	the fraction of RRC sets in $\mathcal{R}$ that are covered by a node set S
EPT	the expected width of a random RRC set
OPT_L	the maximum $\pi(S)$ for any size-k seed set S
λ	see Equation 4

▶ **Definition 3** (Obstructed Nodes). *A node $w \in cl_g(A_L)$ is* obstructed *and cannot be saved by A_L if for every path p from A_L to w there exists a node u on p such that $|SP_g(A_C, u)| < |SP_G(A_L, u)|$.*

Let $obs_g(A_L)$ be the set of obstructed nodes for A_L. Conceptually, the nodes in $obs_g(A_L)$ are cutoff because some node on the paths from A_L is reached by campaign C before L which stops the diffusion of L.

To help illustrate the concept of obstructed nodes, consider the graph presented in Figure 1 and the following possible world instance. Assume that the solid lines are *live* edges that make up the deterministic graph $g \sim \mathcal{G}$ for campaign C in the influence propagation process. The dashed lines are edges that were not realized for campaign C. The adversary campaign C starts from v_c while the limiting campaign L starts from v. Recall, the deterministic graph G for campaign L in this possible world instance is comprised of *both* the solid and dashed edges due to the high effectiveness property. Observe that $|SP_G(v, w)| = 4$ and $|SP_g(A_C, w)| = 5$. However, w cannot be saved in the resulting cascade since at timestamp 1 the node u will adopt campaign C. This intersects the shortest path from v to w and therefore campaign L will not be able to reach node w since a node never switches campaigns. Thus, we say that node w is *obstructed* by C.

Prevention & Saviours

Next, we formally define the prevention, $\pi(A_L)$, which corresponds to the number of nodes saved by A_L. That is, $\pi(A_L) = |R_g(A_C) \cap (cl_g(A_L) \setminus obs_g(A_L))|$ where $R_H(S)$ is the set of nodes in graph H that are *reachable* from set S (a node v in H is reachable from S if there exists a directed path in H that starts from a node in S and ends at v). We write $\mathbb{E}[\pi(A_L)] = \mathbb{E}_{g\sim\mathcal{G}}[\pi(A_L)]$ for the expected prevention of A_L in $\mathcal{G}$. Finally, let $OPT_L = max_{S:|S|=k}\{\mathbb{E}[\pi(S)]\}$ be the maximum expected prevention of a set of k nodes.

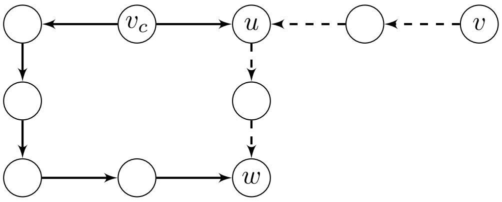


Figure 1 An example illustrating the concept of obstructed nodes where the possible world graph for campaign C is made up of the solid edges and the possible world for campaign L is made up of both solid and dashed lines.

We refer to the set of nodes that could have saved u as the *saviours* of u. A node w is a candidate saviour for u if there is a directed path from w to u in G (i.e. reverse reachability). Then, w is a saviour for u subject to the additional constraint that w would not be cutoff by the diffusion of A_C in g. That is, a candidate saviour w would be cutoff and cannot be a saviour for u if for every path p from w to u there exists a node v_b such that $|SP_g(A_C, v_b)| < |SP_G(w, v_b)|$. We refer to the set of candidate saviours for u that are cutoff as $\tau_g(u)$. Thus, we can define the saviours of u as the set $R_{G^T}(u) \setminus \tau_g(u)$. Therefore, we have:

▶ **Definition 4** (Reverse Reachability without Cutoff Set). *The reverse reachability without cutoff (RRC) set for a node v in $g \sim \mathcal{G}$ is the set of saviour nodes of v, i.e. the set of nodes that can save v. (That is, for each node u in the RRC set, $u \in R_{G^T}(u) \setminus \tau_g(u)$.) If $v \notin R_g(A_C)$ then we define the corresponding RRC set as empty since v is not eligible to be saved.*

▶ **Definition 5** (Random RRC Set). *A random RRC set is an RRC set generated on an instance of $g \sim \mathcal{G}$, for a node selected uniformly at random from g.*

Closing the Gap

Before presenting our reverse sampling approach, we make the following remark regarding obstruction in the context of prior work. The key observation that lead to our definition of obstructed nodes is that the shortest path condition must hold along the *entire* path. This observation was missed by [4] in the MCIC model. Instead, a correct *recursive* definition was provided for the set of nodes that are saved, but the resulting characterization based on shortest paths misses the crucial case of nodes that are obstructed.

Importantly, the solution in [4] can be recovered with a modified proof for Claim 1 and Theorem 4.2. In particular, the statements must include the notion of obstructed nodes in their *inoculation graph* definition, but a careful inspection shows that their objective function remains submodular after this inclusion. As a result, the greedy approach of [4] still provides the stated approximation guarantees and also allows us to incorporate the ideas of [3] in our solution (as [3] requires a submodular objective function as well).

5 Reverse Prevention Sampling

This section presents our misinformation prevention method, *Reverse Prevention Sampling (RPS)*. At a high level, *RPS*, in the same spirit as *RIS*, consists of two steps. In the first step it derives a parameter θ that ensures a solution of high quality will be produced. In the second step, using the estimate θ from step one, it generates θ RRC sets and then computes the maximum coverage on the resulting collection. More precisely, the two steps are:

1. **Parameter Estimation.** Compute a lower-bound for the maximum expected prevention among all possible size-k seed sets for A_L and then use the lower-bound to derive a parameter θ.
2. **Node Selection.** Sample θ random RRC sets from $\mathcal{G}$ to form a set $\mathcal{R}$ and then compute a size-k seed set S_k^* that covers a large number of RRC sets in $\mathcal{R}$. Return S_k^* as the final result.

In the rest of this section, we first tackle the challenging task of correctly generating RRC sets in the Node Selection step under the MCIC model. Next, we identify the conditions necessary for the Node Selection of *RPS* to return a solution of good quality and then describe how these conditions are achieved in the Parameter Estimation phase. Table 1 provides a reference to some of the frequently used notation. All proofs can be found in the full version [28].

Node Selection

The pseudocode of *RPS*'s Node Selection step is presented in Algorithm 1. Given $\mathcal{G}$, k, A_C, and a constant θ as input, the algorithm stochastically generates θ random RRC sets, accomplished by repeated invocation of the prevention of misinformation process, and inserts them into a set $\mathcal{R}$. Next, the algorithm follows a greedy approach for the *maximum coverage problem* to select the final seed set. In each iteration, the algorithm selects a node v_i that covers the largest number of RRC sets in $\mathcal{R}$, and then removes all those covered RRC sets from $\mathcal{R}$. The k selected nodes are put into a set S_k^*, which is returned as the final result.

Algorithm 1 NodeSelection($\mathcal{G}$,k,A_C,θ).

```
1: $\mathcal{R} \leftarrow \emptyset$
2: Generate $\theta$ random RRC sets and insert them into $\mathcal{R}$.
3: Initialize a node set $S_k^* \leftarrow \emptyset$
4: for $i = 1,\ldots,k$ do
5:     Identify the node $v_i$ that covers the most RRC sets in $\mathcal{R}$
6:     Add $v_i$ into $S_k^*$
7:     Remove from $\mathcal{R}$ all RRC sets that are covered by $v_i$
8: return $S_k^*$
```

Lines 4-8 in Algorithm 1 correspond to a standard greedy approach for a *maximum coverage problem*. The problem is equivalent to maximizing a submodular function with cardinality constraints for which it is well known that a greedy approach returns a $(1 - 1/e)$-approximate solution in linear time [21].

5.1 RRC set generation

Next, we describe how to generate RRC sets correctly for the EIL problem under the MCIC model, which is more complicated than generating RR sets for the IC model [30]. The construction of RRC sets is done according to Definition 4. Recall that in the MCIC model, whether a node can be saved or not is based on a number of factors such as whether v is reachable via a path in $g \sim \mathcal{G}$ from A_C and the diffusion history of each campaign. Our algorithms tackle the complex interactions between campaigns by first identifying nodes that can be influenced by C which reveals important information for generating RRC sets for L.

Line 2 generates $\mathcal{R}$ by repeated simulation of the misinformation prevention process. The generation of each random RRC set is implemented as two breath-first searches (BFS) on $\mathcal{G}$ and G^T respectively. The first BFS is a *forward labelling* process from A_C implemented as a forward BFS on $\mathcal{G}$ that computes the influence set of A_C in a possible world. The second BFS on G^T is a novel bounded-depth BFS with pruning that carefully tracks which nodes will become obstructed and is described in detail below.

Forward BFS with Lazy Sampling

We first describe the forward labelling process. As the forward labeling is unlikely to reach the whole graph, we simply reveal edge states on demand ("lazy sampling"), based on the principle of deferred decisions. Given the seed set A_C of campaign C, we perform a randomized BFS starting from A_C where each outgoing edge e in $\mathcal{G}$ is traversed with $p_C(e)$ probability. The set of nodes traversed in this manner ($R_g(A_C)$) is equivalent to $I(A_C)$ for $g \sim \mathcal{G}$, due to deferred randomness. Note that in each step of the above BFS we record at each node w the minimum distance from A_C to w, denoted $D(w)$, for use in the second BFS.

Given a randomly selected node u in G, observe that for u to be able to be saved we require $u \in R_g(A_C)$. Therefore, if the randomly selected node $u \notin R_g(A_C)$ then we return an empty RRC set. On the other hand, if $u \in R_g(A_C)$, we have $D(u) = |SP_g(A_C, u)|$ as a result of the above randomized BFS which indicates the maximum distance from u that candidate saviour nodes can exist. We run a second BFS from u in G^T to depth $D(u)$ to determine the saviour nodes for u by carefully pruning those nodes that would become obstructed.

Bounded-depth BFS with Pruning

The second BFS on G^T, presented in Algorithm 2, takes as input a source node u, the maximum depth $D(u)$, and a directed graph G^T. Algorithm 2 utilizes special indicator values associated with each node w to account for potential cutoffs from C. Each node w holds a variable, $\beta(w)$, which indicates the distance beyond w that the BFS can go before the diffusion would have been cutoff by C propagating in g. The β value for each node w is initialized to $D(w)$. In each round, the current node w has an opportunity to update the β value of each of its successors only if $\beta(w) > 0$. For each successor z of w, we assign $\beta(z) = \beta(w) - 1$ if $\beta(z) = \texttt{null}$ or if $\beta(z) > 0$ and $\beta(w) - 1 < \beta(z)$. In this way, each ancestor of z will have an opportunity to apply a β value to z to ensure that if any ancestor has a β value then so will z and furthermore, the β variable for z will be updated with the smallest β value from its ancestors. We terminate the BFS early if we reach a node w with $\beta(w) = 0$.

Figure 2 captures the primary scenarios encountered by Algorithm 2 when initialized at u. The enclosing dotted line represents the extent of the influence of campaign C for the current influence propagation process. First, notice that if the BFS moves away from $A_C = \{v_c\}$, as in the case of node z, that, once we move beyond the influence boundary of C, there will be no potential for cutoff. As such, the BFS is free to traverse until the maximum depth $D(u)$ is reached. On the other hand, if the BFS moves towards (or perpendicular to) v_c then we must carefully account for potential cutoff. For example, when the BFS reaches v, we know the distance from v_c to v: $D(v) = SP_g(v_c, v)$. Therefore, the BFS must track the fact that there cannot exist saviours at a distance $D(v)$ beyond v. In other words, if we imagine initializing a misinformation prevention process from a node w such that $SP_G(v, w) > D(v)$ then v will adopt campaign C before campaign L can reach v. Therefore, at each out-neighbour of v we use the knowledge of $D(v)$ to track the distance beyond v that saviours can exist. This updating process tracks the smallest such value and is allowed to cross the enclosing influence boundary of campaign C ensuring that all potential for cutoff is tracked.

Finally, we collect all nodes visited during the process (including u), and use them to form an RRC set. The runtime of this procedure is precisely the sum of the degrees (in G) of the nodes in $R_g(A_C)$ plus the sum of the degrees of the nodes in $R_{G^T}(u) \setminus \tau(u)$.

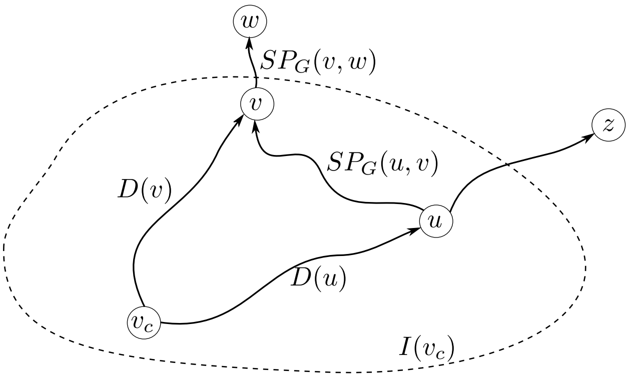


Figure 2 An overview of the primary scenarios encountered by Algorithm 2.

We briefly note another key difference between *RPS* and *RIS* occurs in the RRC set generation step. Unlike in the single campaign setting, generating an RRC set is comprised of two phases instead of just one. First, we are required to simulate the spread of misinformation since being influenced by campaign C is a pre-condition for being saved. As a result, only a fraction of the simulation steps of *RPS* provide signal for the prevention value we are trying to estimate. This difference is made concrete in the running time analysis to follow.

Algorithm 2 generateRRC(u, $D(u)$, G^T).

```
let $R \leftarrow \emptyset$, $Q$ be a queue and $Q.enqueue(u)$
set $u.depth = 0$ and label $u$ as discovered
while $Q$ is not empty do
    $w \leftarrow Q.dequeue()$, $R \leftarrow R \cup \{w\}$
    if $w.depth = D(u)$ OR $\beta(w) = 0$ then
        continue
    for all nodes $z$ in $G^T.adjacentEdges(w)$ do
        if $\beta(w) > 0$ AND $\beta(z) > 0$ then
            if $\beta(w) - 1 < \beta(z)$ then
                $\beta(z) \leftarrow \beta(w) - 1$
        else if $\beta(w) > 0$ then
            $\beta(z) \leftarrow \beta(w) - 1$
        if $z$ is not labelled as discovered then
            set $z.depth = w.depth + 1$, label $z$ as discovered and $Q.enqueue(z)$
return $R$
```

5.2 Analysis

In this section we focus on two parameters: solution quality and runtime. For Algorithm 1 to return a solution with approximation guarantee, we will provide a lower bound on θ. Then, we will analyze the running time of the algorithm in terms of θ and a quantity *EPT* that captures the expected number of edges traversed when generating a random RRC set.

Approximation Guarantee

We begin by establishing the crucial connection between RRC sets and the prevention process on $\mathcal{G}$. That is, the prevention of a set of nodes S is precisely n times the probability that a node u, chosen uniformly at random, has a saviour from S.

▶ **Lemma 2.** *For any seed set S and any node v, the probability that a prevention process from S can save v equals the probability that S overlaps an RRC set for v.*

For any node set S, let $F_{\mathcal{R}}(S)$ be the fraction of RRC sets in $\mathcal{R}$ covered by S. Then, based on Lemma 2, we can prove that the expected value of $n \cdot F_{\mathcal{R}}(S)$ equals the expected prevention of S in $\mathcal{G}$.

▶ **Corollary 1.** $\mathbb{E}[n \cdot F_{\mathcal{R}}(S)] = \mathbb{E}[\pi(S)]$

Corollary 1 implies that we can estimate $\mathbb{E}[\pi(S)]$ by estimating the fraction of RRC sets in $\mathcal{R}$ covered by S. The number of sets covered by a node v in $\mathcal{R}$ is precisely the number of times we observed that v was a saviour for a randomly selected node u. We can therefore think of $n \cdot F_{\mathcal{R}}(S)$ as an estimator for $\mathbb{E}[\pi(S)]$. Our primary task is to show that it is a *good* estimator. Using Chernoff bounds, we show that $n \cdot F_{\mathcal{R}}(S)$ is an accurate estimator of any node set S's expected prevention, when θ is sufficiently large:

▶ **Lemma 3.** *Suppose that θ satisfies*

$$\theta \geq (8 + 2\epsilon)n \cdot \frac{l \log n + \log \binom{n}{k} + \log 2}{OPT_L \cdot \epsilon^2} \tag{1}$$

Then, for any set S of at most k nodes, the following inequality holds with at least $1 - n^{-l}/\binom{n}{k}$ probability:

$$\left| n \cdot F_{\mathcal{R}}(S) - \mathbb{E}[\pi(S)] \right| < \frac{\epsilon}{2} \cdot OPT_L \tag{2}$$

Based on Lemma 3, we prove that if Eqn. 1 holds, Algorithm 1 returns a $(1 - 1/e - \epsilon)$-approximate solution with high probability by a simple application of Chernoff bounds.

▶ **Theorem 2.** *Given a θ that satisfies Equation 1, Algorithm 1 returns a $(1 - 1/e - \epsilon)$-approximate solution with at least $1 - n^{-l}$ probability.*

Runtime

First, we will define EPT which captures the expected number of edges traversed when generating a random RRC set. After that, we define the expected runtime of RPS in terms of EPT and the parameter θ.

Let M_R be the instance of $R_g(A_C)$ used in computing an RRC set R. Then, we define the *width* of an RRC set R, denoted as $\omega(R)$, as the number of edges in G that point to nodes in R plus the number of edges in G that originate from nodes in M_R. That is

$$\omega(R) = \sum_{u \in M_R} outdegree_G(u) + \sum_{v \in R} indegree_G(v) \tag{3}$$

Let EPT be the expected width of a random RRC set, where the expectation is taken over the randomness in R and M_R, and observe that Algorithm 1 has an expected runtime of $O(\theta \cdot EPT)$. This can be observed by noting that EPT captures the expected number of edge traversals required to generate a random RRC set since an edge is only considered in the propagation process (either of the two BFS's) if it points to a node in R or originates from a node in M_R.

An important consideration is that, since OPT_L is unknown, we cannot set θ directly from Equation 1. For simplicity, we define

$$\lambda = (8 + 2\epsilon)n \cdot \left(l \log n + \log \binom{n}{k} + \log 2\right) \cdot \epsilon^{-2} \tag{4}$$

and rewrite Equation 1 as $\theta \geq \lambda/OPT_L$. In the parameter estimation step we employ the techniques of [30] to derive a θ value for RPS that is above the threshold but also allows for practical efficiency.

5.3 Parameter Estimation

Our objective in this section is to identify a θ that makes $\theta \cdot EPT$ reasonably small, while still ensuring $\theta \geq \lambda/OPT_L$. We begin with some definitions. Let $\mathcal{V}^*$ be a probability distribution over the nodes in G, such that the probability mass for each node is proportional to its indegree in G. Let v^* be a random variable following $\mathcal{V}^*$ and recall that M_R is a random instance of $R_g(A_C)$ that is equivalent to the influence $I(A_C)$ for a possible world g. Furthermore, define $\omega(M_R)$, the number of edges in G that originate from nodes in M_R, as $\omega(M_R) = \sum_{u \in M_R} outdegree_G(u)$. Then we prove the following.

▶ **Lemma 4.** $\frac{m}{n} \cdot \mathbb{E}[\pi(\{v^*\})] = EPT - \mathbb{E}[\omega(M_R)]$, *where the expectation of* $\pi(\{v^*\})$ *and* $\omega(M_R)$ *is taken over the randomness in* v^* *and the prevention process.*

Lemma 4 shows that if we randomly sample a node from $\mathcal{V}^*$ and calculate its expected prevention p, then on average we have $p = \frac{n}{m}(EPT - \mathbb{E}[\omega(M_R)])$. This implies that $\frac{n}{m}(EPT - \mathbb{E}[\omega(M_R)]) \leq OPT_L$, since OPT_L is the maximum expected prevention of any size-k node set.

Recall that the expected runtime complexity of Algorithm 1 is $O(\theta \cdot EPT)$. Now, suppose we are able to identify a parameter t such that $t = \Omega(\frac{n}{m}(EPT - \mathbb{E}[\omega(M_R)]))$ and $t \leq OPT_L$. Then, by setting $\theta = \lambda/t$, we can guarantee that Algorithm 1 is correct, since $\theta \geq \lambda/OPT_L$, and has an expected runtime complexity of

$$O(\theta \cdot EPT) = O\Big(\frac{\lambda}{t} \cdot EPT\Big) = O\left(\frac{\lambda \cdot EPT}{\frac{n}{m}(EPT - \mathbb{E}[\omega(M_R)])}\right) \tag{5}$$

Furthermore, if we define a ratio $\gamma \in (0, 1)$ which captures the relationship between $\mathbb{E}[\omega(M_R)]$ and EPT by writing $\mathbb{E}[\omega(M_R)] = \gamma EPT$, we can rewrite Equation 5 as

$$O\Big(\frac{m}{n}\Big(\frac{1}{1-\gamma}\Big)\lambda\Big) = O((k + l)(m + n)(1/(1 - \gamma)) \log n/\epsilon^2) \tag{6}$$

Note that γ is a data-dependent approximation factor not present in [30], but arises from the MCIC model. In particular, the RRC set generation relies crucially on first computing the spread of misinformation from campaign C in order to determine the set of nodes that can be saved. See Section 6 for a detailed discussion of γ.

Computing t

We postpone the details of how to derive $t = \Omega(\frac{n}{m}(EPT - \mathbb{E}[\omega(M_R)]))$, a lower bound for the optimal prevention value, to the full version of the paper [28]. Briefly, we mimic the adaptive sampling approach of [30], which estimates a lower bound LB by dynamically adjusting the number of measurements based on the observed values of LB. The runtime required for the lower bound estimation is linear in Equation 6.

Wrapping Up

As a result, by Equation 6, *RPS* runs in $O((k+l)(m+n)(1/(1-\gamma))\log n/\epsilon^2)$ expected time. Furthermore, by Theorem 2 and the lower bound estimation, *RPS* returns a $(1-1/e-\epsilon)$-approximate solution with at least $1-3n^{-l}$ probability and the success probability can be increased to $1-n^{-l}$ by scaling l up by a factor of $1+\log 3/\log n$.

Finally, we note that the time complexity of *RPS* is *near-optimal* up to the instance-specific factor γ under the MCIC model, as it is only a $(\frac{1}{1-\gamma})\log n$ factor larger than the $\Omega(m+n)$ lower-bound proved in Section 6 (for fixed k, l, and ϵ).

6 Lower Bounds

Comparison with *MCGreedy*

MCGreedy runs in $O(kmnr)$ time, where r is the number of Monte Carlo samples used to estimate the expected prevention of each node set. Budak et al. do not provide a detailed analysis related to how r should be set to achieve a $(1-1/e-\epsilon)$-approximation ratio in the MCIC model, only pointing out that when each estimation of expected prevention has ϵ relative error, *MCGreedy* returns a $(1-1/e-\epsilon')$-approximate solution for a particular ϵ' [4]. In the following lemma, we present a more precise characterization of the relationship between r and *MCGreedy*'s approximation ratio in the MCIC model.

▶ **Lemma 5.** MCGreedy *returns a $(1-1/e-\epsilon)$-approximate solution with at least $1-n^{-l}$ probability, if*

$$r \geq (8k^2+2k\epsilon)\cdot n\cdot \frac{(l+1)\log n+\log k}{\epsilon^2\cdot OPT_L} \tag{7}$$

Assume that we know OPT_L in advance and set r to the smallest value satisfying the above inequality, in *MCGreedy*'s favour. In that case, the time complexity of *MCGreedy* is $O(k^3lmn^2\epsilon^{-2}\log n/OPT_L)$. Towards comparing *MCGreedy* to *RPS*, we show the following upper bound on the value of γ.

▶ **Claim 1.** $\gamma \leq \frac{n}{n+1}$

Claim 1 shows that the expected runtime for *RPS* is at most $O((k+l)mn\epsilon^{-2}\log n)$. As a consequence, given that $OPT_L \leq n$, the expected runtime of *MCGreedy* is always more than the expected runtime of *RPS*. In practice, we observe that for typical social networks $OPT_L \ll n$ and $\frac{1}{1-\gamma} \ll n+1$ resulting in superior scalability of *RPS* compared to *MCGreedy*.

A Lower Bound for EIL

In the theorem below, we provide a lower bound on the time it takes for any algorithm to compute a β-approximation for the EIL problem given uniform node sampling and an adjacency list representation. Thus, we rule out the possibility of a sublinear time algorithm for the EIL problem for an arbitrary β.

▶ **Theorem 3.** *Let $0<\epsilon<\frac{1}{10e}$, $\beta\leq 1$ be given. Any randomized algorithm for EIL that returns a set of seed nodes with approximation ratio β, with probability at least $1-\frac{1}{e}-\epsilon$, must have a runtime of at least $\frac{\beta(m+n)}{24\min\{k,1/\beta\}}$.*

7 Generalization to the Multi-Campaign Triggering Model

The *triggering model* is an influence propagation model that generalizes the IC and LT models. It assumes that each node v is associated with a triggering distribution $\mathcal{T}(v)$ over the power set of v's incoming neighbors. An influence propagation process under the triggering model works as follows: (1) for each node v, take a sample from $\mathcal{T}(v)$ and define the sample as the triggering set of v, then (2) at timestep 1 activate the seed set S, and (3) in subsequent timesteps, if an active node appears in the triggering set of v, then v becomes active. The propagation terminates when no more nodes can be activated.

We can define a *multi-campaign* version of the triggering model (MCT) that generalizes the MCIC model by associating each node with a *campaign-specific* triggering distribution $\mathcal{T}_Z(v)$ where $Z \in \{C, L\}$. The propagation process under MCT proceeds exactly as under MCIC with the exception that activation between rounds (step 2) is determined by $\mathcal{T}_C(v)$ and $\mathcal{T}_L(v)$. To the best of our knowledge, we are the first to formally define a multi-campaign version of the triggering model.

The key aspect of the MCIC model that enabled the existence of obstructed nodes is that the two campaigns are allowed to propagate along *different sets of edges* in a possible world X. This is exactly the intuition captured by the example in Figure 1 and is caused by L and C having separate propagation probabilities in $\mathcal{G}$. As a result, the campaigns traverse potentially unique graphs in X and results in the possibility of the obstruction of L by C. This observation holds under the more general setting of MCT due to the campaign-specific triggering sets and so the obstruction phenomenon exists under the MCT model.

Following the observations made in [30], our solutions can be easily extended to operate under the multi-campaign triggering (MCT) model with a modified high effectiveness property. Under MCT, the high effectiveness property asserts that $\mathcal{T}_L(v) = in(v)$ where $in(v)$ is the set of in-neighbours of v in G. Observe that Algorithm 1 does not rely on anything specific to the MCIC model, except a subroutine to generate random RRC sets. Thus, we can revise the definition of RRC sets to accommodate the MCT model.

Due to space constraints, we delay the details showing that this revised solution retains the performance guarantees of *RPS* under the MCT model to the full version of the paper. However, we note that all of the theoretical analysis of *RPS* is based on the Chernoff bounds and Lemma 2, without relying on any other results specific to the MCIC model. Therefore, once we establish an equivalent to Lemma 2, it is straightforward to combine it with the Chernoff bounds to show that, under the MCT model, *RPS* provides the same performance guarantees as in the case of the MCIC model. Thus, we have the following theorem:

▶ **Theorem 4.** *Under MCT,* RPS *runs in* $O((k + l)(m + n)(1/(1 - \gamma)) \log n/\epsilon^2)$ *expected time, and returns a* $(1 - 1/e - \epsilon)$*-approximate solution with at least* $1 - 3n^{-l}$ *probability.*

8 Summary of Experiments

The focus of our experiments is *algorithm efficiency* measured in runtime where our goal is to demonstrate the superior performance of *RPS* compared to *MCGreedy*. We observe that *RPS* provides a significant improvement of several orders of magnitude over *MCGreedy*. Further, we confirm that $\frac{1}{1-\gamma} \ll n + 1$ on our small datasets which is strong evidence that *RPS* will outperform *MCGreedy* on typical social networks. Finally, we observe that the vast majority of the computation time is spent on generating the RRC sets for $\mathcal{R}$. A detailed experimental analysis and discussion is provided in the full version [28].

9 Conclusion & Future Work

In this work we presented *RPS*, a novel and scalable approach to the EIL problem. We showed the correctness and a detailed running-time analysis of our approach. Furthermore, we provided two lower bound results: one on the running-time requirement for any approach to solve the EIL problem and another on the number of Monte Carlo simulations required by *MCGreedy* to return a correct solution with high probability. As a result, the expected runtime of *RPS* is always less than the expected runtime of *MCGreedy*. Finally, we describe how our approach can be generalized to a multi-campaign triggering model. In future work we plan to investigate how to adapt our approach to a scenario where the source of the misinformation is only partially known.

References

1 Bob Abeshouse. Troll factories, bots and fake news: Inside the Wild West of social media, February 2018. URL: `https://www.aljazeera.com/blogs/americas/2018/02/troll-factories-bots-fake-news-wild-west-social-media-180207061815575.html`.

2 Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *WINE'07*, pages 306–311, Berlin, Heidelberg, 2007. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1781894.1781932`.

3 Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. Influence Maximization in Social Networks: Towards an Optimal Algorithmic Solution. *CoRR*, abs/1212.0884, 2012. URL: `http://arxiv.org/abs/1212.0884`.

4 C. Budak, D. Agrawal, and A. El Abbadi. Limiting the spread of misinformation in social networks. In *WWW'11*, 2011.

5 Wei Chen, Laks V. S. Lakshmanan, and Carlos Castillo. *Information and Influence Propagation in Social Networks*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013. `doi:10.2200/S00527ED1V01Y201308DTM037`.

6 Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *2010 IEEE international conference on data mining*, pages 88–97. IEEE, 2010.

7 Lidan Fan, Zaixin Lu, Weili Wu, Bhavani Thuraisingham, Huan Ma, and Yuanjun Bi. Least cost rumor blocking in social networks. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 540–549. IEEE, 2013.

8 Qizhi Fang, Xin Chen, Qingqin Nong, Zongchao Zhang, Yongchang Cao, Yan Feng, Tao Sun, Suning Gong, and Ding-Zhu Du. General Rumor Blocking: An Efficient Random Algorithm with Martingale Approach. In *International Conference on Algorithmic Applications in Management*, pages 161–176. Springer, 2018.

9 Peter Foster. 'Bogus' AP tweet about explosion at the White House wipes billions off US markets, April 2018. URL: `https://www.telegraph.co.uk/finance/markets/10013768/Bogus-AP-tweet-about-explosion-at-the-White-House-wipes-billions-off-US-markets.html`.

10 Amit Goyal, Francesco Bonchi, Laks V. S. Lakshmanan, and Suresh Venkatasubramanian. On minimizing budget and time in influence propagation over social networks. *Social Netw. Analys. Mining*, 3(2):179–192, 2013. `doi:10.1007/s13278-012-0062-z`.

11 Chris Graham. YouTube employee's Twitter account hacked to spread fake news during attack, April 2018. URL: `https://www.telegraph.co.uk/technology/2018/04/04/youtube-employees-twitter-account-hacked-spread-fake-news-attack/`.

12 Naeemul Hassan, Gensheng Zhang, Fatma Arslan, Josue Caraballo, Damian Jimenez, Siddhant Gawsane, Shohedul Hasan, Minumol Joseph, Aaditya Kulkarni, Anil Kumar Nayak, et al. Claimbuster: The first-ever end-to-end fact-checking system. *PVLDB*, 10(12):1945–1948, 2017.

13 Laura Hautala. Reddit was a misinformation hotspot in 2016 election, study says, December 2018. URL: `https://www.cnet.com/news/reddit-election-misinformation-2016-research/`.

14 Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 463–474. SIAM, 2012.

15 Kyomin Jung, Wooram Heo, and Wei Chen. Irie: Scalable and robust influence maximization in social networks. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 918–923. IEEE, 2012.

16 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD'03*, 2003. `doi:10.1145/956750.956769`.

17 Jooyeon Kim, Behzad Tabibian, Alice Oh, Bernhard Schölkopf, and Manuel Gomez-Rodriguez. Leveraging the crowd to detect and reduce the spread of fake news and misinformation. In *WSDM*, pages 324–332. ACM, 2018.

18 Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *WWW '10*, pages 641–650, New York, NY, USA, 2010. ACM. `doi:10.1145/1772690.1772756`.

19 Yanhua Li, Wei Chen, Yajun Wang, and Zhi-Li Zhang. Influence Diffusion Dynamics and Influence Maximization in Social Networks with Friend and Foe Relationships. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 657–666, New York, NY, USA, 2013. ACM. `doi:10.1145/2433396.2433478`.

20 Yishi Lin and John CS Lui. Analyzing competitive influence maximization problems with partial information: An approximation algorithmic framework. *Performance Evaluation*, 91:187–204, 2015.

21 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978. `doi:10.1007/BF01588971`.

22 Nam P. Nguyen, Guanhua Yan, My T. Thai, and Stephan Eidenbenz. Containment of Misinformation Spread in Online Social Networks. In *Proceedings of the 4th Annual ACM Web Science Conference*, WebSci '12, pages 213–222, New York, NY, USA, 2012. ACM. `doi:10.1145/2380718.2380746`.

23 Maya Oppenheim. YouTube shooting: Twitter and Facebook explodes with misinformation and hoaxes, April 2018. URL: `https://www.independent.co.uk/news/world/americas/youtube-shooting-fake-news-twitter-facebook-identity-illegal-immigrant-hoax-misinformation-a8287946.html`.

24 Nishith Pathak, Arindam Banerjee, and Jaideep Srivastava. A generalized linear threshold model for multiple cascades. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 965–970. IEEE, 2010.

25 Kai Shu, H Russell Bernard, and Huan Liu. Studying fake news via network analysis: detection and mitigation. In *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*, pages 43–65. Springer, 2019.

26 Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

27 Michael Simpson, Venkatesh Srinivasan, and Alex Thomo. Clearing contamination in large networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1435–1448, 2016.

28 Michael Simpson, Venkatesh Srinivasan, and Alex Thomo. Reverse Prevention Sampling for Misinformation Mitigation in Social Networks, 2018. `arXiv:1807.01162`.

29 Olivia Solon. Facebook's failure: did fake news and polarized politics get Trump elected?, November 2018. URL: `https://www.theguardian.com/technology/2016/nov/10/facebook-fake-news-election-conspiracy-theories`.

30 Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence Maximization: Near-optimal Time Complexity Meets Practical Efficiency. In *Proceedings of the 2014 ACM SIGMOD International*

Conference on Management of Data, SIGMOD '14, pages 75–86, New York, NY, USA, 2014. ACM. doi:10.1145/2588555.2593670.

31 Sebastian Tschiatschek, Adish Singla, Manuel Gomez Rodriguez, Arpit Merchant, and Andreas Krause. Fake news detection in social networks via crowd signals. In *WWW*, pages 517–524. WWW, 2018.

32 Chi Wang, Wei Chen, and Yajun Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25(3):545–576, 2012.

A Simple Parallel Algorithm for Natural Joins on Binary Relations

Yufei Tao
Chinese University of Hong Kong, Hong Kong
taoyf@cse.cuhk.edu.hk

Abstract

In PODS'17, Ketsman and Suciu gave an algorithm in the MPC model for computing the result of any natural join where every input relation has two attributes. Achieving an optimal load $O(m/p^{1/\rho})$ – where m is the total size of the input relations, p the number of machines, and ρ the fractional edge covering number of the join – their algorithm requires 7 rounds to finish. This paper presents a simpler algorithm that ensures the same load with 3 rounds (in fact, the second round incurs only a load of $O(p^2)$ to transmit certain statistics to assist machine allocation in the last round). Our algorithm is made possible by a new theorem that provides fresh insight on the structure of the problem, and brings us closer to understanding the intrinsic reason why joins on binary relations can be settled with load $O(m/p^{1/\rho})$.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Natural Joins, Conjunctive Queries, MPC Algorithms, Parallel Computing

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.25

1 Introduction

Understanding the computational hardness of joins has always been a central topic in database theory. Traditionally, research efforts (see [1, 4, 8, 11, 12, 14, 15] and the references therein) have focused on discovering fast algorithms for processing joins in the *random access machine* (RAM) model. Nowadays, massively parallel systems such as Hadoop [6] and Spark [2] (`https://spark.apache.org`) have become the mainstream computing architecture for performing analytical tasks on gigantic volumes of data, where direct implementation of RAM join algorithms rarely gives satisfactory performance. A main reason behind this phenomenon is that, while a RAM algorithm is designed to reduce the CPU time, in systems like Hadoop and Spark it is much more important to minimize the amount of communication across the participating machines, because the overhead of delivering all the necessary messages typically overwhelms the cost of CPU calculation. This has motivated a line of research – which also includes this work – that aims to understand the communication complexities of join problems.

1.1 Problem Definition

In this subsection, we will first give a formal definition of *natural join* – the type of joins studied in this paper – and then elaborate on the computation model assumed.

Natural Joins. Let **att** be a countably infinite set where each element is called an *attribute*. Let **dom** be another countably infinite set. A *tuple* over a set $U \subseteq \mathbf{att}$ is a function $\boldsymbol{u} : U \to \mathbf{dom}$. Given a subset V of U, define $\boldsymbol{u}[V]$ as the tuple $\boldsymbol{v}$ over V such that $\boldsymbol{v}(X) = \boldsymbol{u}(X)$ for every $X \in V$. We say that $\boldsymbol{u}[V]$ is the *projection* of $\boldsymbol{u}$ on V.

A *relation* is defined to be a set R of tuples over the same set U of attributes. We say that R is *over* U, and that the *scheme* of R is U, represented with the notation $scheme(R) = U$.

23rd International Conference on Database Theory (ICDT 2020).
Editors: Carsten Lutz and Jean Christoph Jung; Article No. 25; pp. 25:1–25:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *arity* of R, denoted as $arity(R)$, equals $|scheme(R)|$. R is *unary* if $arity(R) = 1$, and *binary* if $arity(R) = 2$.

A *join query* is defined as a set $\mathcal{Q}$ of relations. If we let $attset(\mathcal{Q}) = \bigcup_{R \in \mathcal{Q}} scheme(R)$, the result of the query, denoted as $Join(\mathcal{Q})$, is the following relation over $attset(\mathcal{Q})$

$$\Big\{\text{tuple } \boldsymbol{u} \text{ over } attset(\mathcal{Q}) \;\Big|\; \forall R \in \mathcal{Q},\, \boldsymbol{u}[scheme(R)] \in R\Big\}.$$

If $\mathcal{Q}$ has only two relations R and S, we may also use $R \bowtie S$ as an alternative representation of $Join(Q)$. The integer $m = \sum_{R \in \mathcal{Q}} |R|$ is the *input size* of the query. Concentrating on *data complexity*, we will assume that both $|\mathcal{Q}|$ and $|attset(\mathcal{Q})|$ are constants.

A join query $\mathcal{Q}$ is

- *scheme-clean* if no distinct $R, S \in \mathcal{Q}$ satisfy $scheme(R) = scheme(S)$;
- *simple* if (i) $\mathcal{Q}$ is scheme-clean, and (ii) every $R \in \mathcal{Q}$ is binary.

The primary goal of this paper is to design parallel algorithms for processing simple queries efficiently.

Computation Model. We will assume the *massively parallel computation* (MPC) model which has been widely accepted as a reasonable abstraction of the massively parallel systems that exist today. In this model, there are p machines such that at the beginning the input elements are evenly distributed across these machines. For a join query, this means that each machine stores $\Theta(m/p)$ tuples from the input relations.

An algorithm is executed in *rounds*, each of which has two phases:

- In the first phase, each machine does computation on the data of its local storage.
- In the second phase, the machines communicate by sending messages to each other.

It is important to note that all the messages sent out in the second phase must have already been prepared in the first phase. This prevents a machine from, for example, sending information based on what has been received *during* the second phase. Another round is launched only if the problem has not been solved by the current round. In our context, *solving* a join query means that every tuple in the join result has been produced on at least one machine.

The *load* of a round is defined by the largest number of words that is received by a machine in this round, that is, if machine $i \in [1, p]$ receives x_i words, then the load is $\max_{i=1}^{p} x_i$. The performance of an algorithm is measured by two metrics: (i) the number of rounds, and (ii) the *load* of the algorithm, defined to be the largest load incurred by a round, among all the rounds. CPU computation, which takes place in the first phase of each round, is for free.

The number p of machines is assumed to be significantly less than m, which in this paper means $p^3 \leq m$ specifically. All the algorithms to be mentioned, including those reviewed in the next subsection and the ones we propose, are randomized. Their loads are all bounded in a "high probability manner". Henceforth, whenever we say that an algorithm has load at most L, we mean that its load is bounded by L with probability at least $1 - 1/p^2$. Finally, we consider that every value in **dom** can be encoded in a single word.

1.2 Previous Results

Afrati and Ullman [3] showed that any join query can be solved in a single round with load $\tilde{O}(m/p^{1/\min\{k, |\mathcal{Q}|\}})$ where $k = |attset(\mathcal{Q})|$, and the notation $\tilde{O}$ hides polylogarithmic factors. Improving upon the earlier work [5], Koutris, Beame, and Suciu [10] presented another single-round algorithm that solves a join query with load $\tilde{O}(m/p^{1/\psi})$ where ψ is the *fractional quasi-packing number* of the query. They also proved that $\Omega(m/p^{1/\psi})$ is a lower bound on the load of any one-round algorithm, under certain restrictions on the statistics that the algorithm knows about the input relations.

For algorithms that perform more than one, but still $O(1)$, rounds, $\Omega(m/p^{1/\rho})$ has been shown [10] to be a lower bound on the load, where ρ is the *factional edge covering number* of the query. The value of ρ never exceeds, but can be strictly smaller than, ψ, which implies that multi-round algorithms may achieve significantly lower loads than one-round counterparts, thus providing strong motivation for studying the former.

Though matching the lower bound of $\Omega(m/p^{1/\rho})$ algorithmically still remains open for arbitrary join queries, this has been achieved for several special query classes [3, 7, 9, 10]. In particular, Ketsman and Suciu [9] gave an algorithm, henceforth referred to as "the KS algorithm", that solves a simple query in 7 rounds with load $\tilde{O}(m/p^{1/\rho})$. The class of simple queries bears unique significance due to their relevance to *subgraph enumeration*, which is the problem of finding all occurrences of a subgraph $G' = (V', E')$ in a graph $G = (V, E)$. Regarding E as a relation of two attributes, i.e., source vertex and destination vertex, we can convert subgraph enumeration to a join query on $|E'|$ copies of the "relation" E, and renaming the attributes in each relation to reflect how the vertices of V' are connected in G'; see [12] for an example where G' is a clique of 3 vertices.

1.3 Our Contributions

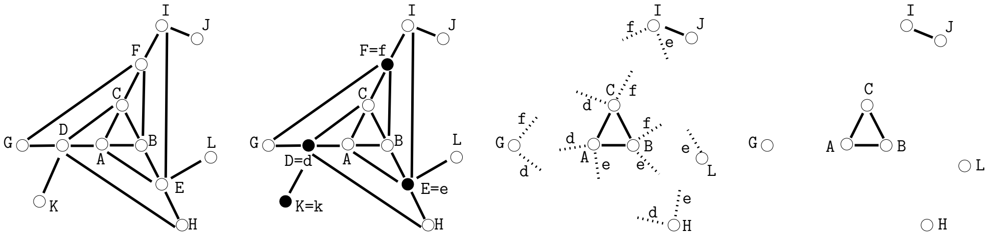
Our main result is that any simple join query can be solved in the MPC model with the optimal load $\tilde{O}(m/p^{1/\rho})$ using 3 rounds. Our algorithm is in fact similar to a subroutine deployed in the KS algorithm, which, however, also demands several other subroutines that entail a larger number of rounds, and are proved to be unnecessary by our solution. The improvement owes to a new theorem that reveals an intrinsic property of the problem, which will be explained shortly with an example. In retrospect, the algorithm of Kestman and Suciu [9] can be regarded as using sophisticated graph-theoretic ideas to compensate for not knowing that property. It is not surprising that their algorithm can be simplified substantially once our understanding on the structure of the problem has been strengthened.

To gain an overview of our techniques, let us consider the join query $\mathcal{Q}$ illustrated by the graph in Figure 1a. An edge connecting vertices X and Y represents a relation $R_{\{X,Y\}}$ with $\mathit{scheme}(R_{\{X,Y\}}) = \{X, Y\}$. $\mathcal{Q}$ is defined by the set of relations represented by the 18 edges in Figure 1a. Notice that $\mathit{attset}(\mathcal{Q}) = \{\mathtt{A}, \mathtt{B}, ..., \mathtt{L}\}$ has a size of 12.

We adopt an idea that is behind nearly all the join algorithms in the MPC model [7, 9, 10], namely, to divide the join result based on "heavy hitters". Let λ be an integer parameter whose choice will be clarified later. A value $x \in \mathbf{dom}$ is *heavy* if an input relation $R \in \mathcal{Q}$ has at least m/λ tuples carrying this value on an attribute $X \in \mathit{scheme}(R)$. The number of heavy values is $O(\lambda)$. A value $x \in \mathbf{dom}$ is *light* if x appears in at least one relation $R \in \mathcal{Q}$ but is not heavy. A tuple in the join result may take a heavy or light value on each of the 12 attributes $\mathtt{A}, ..., \mathtt{K}$. As there are at most $O(\lambda)$ choices on each attribute (namely, light value or one of the $O(\lambda)$ heavy values), there are $O(\lambda^{12})$ "combinations" of choices from all attributes; we will refer to each combination as a *configuration*. If we manage to design an algorithm to find the result tuples under each configuration, executing this algorithm for all configurations solves the query.

Figure 1b illustrates what happens in one of the possible configurations where we constrain attributes $\mathtt{D}$, $\mathtt{E}$, $\mathtt{F}$, and $\mathtt{K}$ to take heavy values $\mathtt{d}$, $\mathtt{e}$, $\mathtt{f}$, and $\mathtt{k}$ respectively, and the other attributes to take light values. Accordingly, vertices $\mathtt{D}$, $\mathtt{E}$, $\mathtt{F}$, and $\mathtt{K}$ are colored black in the figure. This configuration gives rise to a *residue query* $\mathcal{Q}'$ whose input relations are decided as follows:

- For each edge $\{X, Y\}$ with two white vertices, $\mathcal{Q}'$ has a relation $R'_{\{X,Y\}}$ that contains only the tuples in $R_{\{X,Y\}} \in \mathcal{Q}$ using light values on both X and Y;



(a) A join query (b) A residue query (c) After deleting black verticess (d) After semi-join reduction

Figure 1 Processing a join by constraining heavy values.

- For each edge $\{X, Y\}$ with a white vertex X and a black vertex Y, $\mathcal{Q}'$ has a relation $R'_{\{X,Y\}}$ that contains only the tuples in $R_{\{X,Y\}} \in \mathcal{Q}$ each of which uses a light value on X and the constrained heavy value on Y;
- For each edge $\{X, Y\}$ with two black vertices, $\mathcal{Q}'$ has a relation $R'_{\{X,Y\}}$ with only one tuple that takes the constrained heavy values on X and Y, respectively.

For example, in $R'_{\{\texttt{A},\texttt{B}\}}$, a tuple must use light values on both A and B; in $R'_{\{\texttt{D},\texttt{G}\}}$, a tuple must use value d on D and a light value on G; $R'_{\{\texttt{D},\texttt{K}\}}$ has only a single tuple with value d on D and k on K. Finding all result tuples for $\mathcal{Q}$ under the designated configuration amounts to evaluating the residue query $\mathcal{Q}'$.

Since the black attributes have had their values fixed in the configuration, they can be deleted from the residue query, after which some relations in $\mathcal{Q}'$ become unary or even disappear. Relation $R'_{\{\texttt{A},\texttt{D}\}} \in \mathcal{Q}'$, for example, is now regarded as a unary relation over $\{\texttt{A}\}$, with the understanding that every tuple is "piggybacked" the value d on D. Let us denote this unary relation as $R'_{\{\texttt{A}\}|\texttt{d}}$, which is illustrated in Figure 1c with a dotted edge extending from vertex A and carrying the label d. The deletion of D, E, F, and K results in 13 unary relations (e.g., two of them are over $\{\texttt{A}\}$, namely, $R'_{\{\texttt{A}\}|\texttt{d}}$ and $R'_{\{\texttt{A}\}|\texttt{e}}$). Attributes G, H, and L now become *isolated* because they are not connected to any other vertices by solid edges. Relations $R'_{\{\texttt{A},\texttt{B}\}}$, $R'_{\{\texttt{A},\texttt{C}\}}$, $R'_{\{\texttt{B},\texttt{C}\}}$, and $R'_{\{\texttt{I},\texttt{J}\}}$ still have arity 2 because their schemes do not have black attributes. $R'_{\{\texttt{D},\texttt{K}\}}$, on the other hand, has disappeared.

Our algorithm solves the residue query $\mathcal{Q}'$ of Figure 1c in two steps:

1. Perform a *semi-join reduction* which involves two substeps:
   - For every vertex X in Figure 1c, intersect all the unary relations over $\{X\}$ – if any – into a single list $R''_{\{X\}}$. For example, the two unary relations $R'_{\{\texttt{A}\}|\texttt{d}}$ and $R'_{\{\texttt{A}\}|\texttt{e}}$ of A are intersected on A to produce $R''_{\{\texttt{A}\}}$. Note that only the values in $R''_{\{\texttt{A}\}}$ can appear in the final join result.
   - For every non-isolated attribute X in Figure 1c, use $R''_{\{X\}}$ to shrink each non-unary relation $R'_{\{X,Y\}}$, for all relevant Y, to kick out those tuples whose X-values do not appear in $R''_{\{X\}}$. This reduces $R'_{\{X,Y\}}$ to a subset $R''_{\{X,Y\}}$. For example, after the shrinking, every tuple in $R''_{\{\texttt{A},\texttt{B}\}}$ uses a value in $R''_{\{\texttt{A}\}}$ on attribute A, and a value in $R''_{\{\texttt{B}\}}$ on attribute B.
2. Perform a cartesian product. To see how, first observe that the residue query $\mathcal{Q}'$ can now be further simplified into a join query $\mathcal{Q}''$ which includes:
   - The relation $R''_{\{X\}}$ for each isolated attribute X;
   - The relation $R''_{\{X,Y\}}$ for each solid edge in Figure 1c.

   Figure 1d gives a neater view of $\mathcal{Q}''$, from which it is easy to see that *Join*$(\mathcal{Q}'')$ equals the cartesian product of (i) three unary relations $R''_{\{\texttt{G}\}}$, $R''_{\{\texttt{H}\}}$, and $R''_{\{\texttt{L}\}}$, (ii) a binary relation

$R''_{\{\mathtt{I,J}\}}$, and (iii) the result of the "triangle join" $\{R''_{\{\mathtt{A,B}\}}, R''_{\{\mathtt{A,C}\}}, R''_{\{\mathtt{B,C}\}}\}$. This cartesian product can be generated in one round using the *hypercube algorithm* of [3], leveraging the fact that only light values are present in these relations. An optimal load can be achieved by setting $\lambda = \Theta(p^{1/(2\rho)})$.

The KS algorithm deploys a similar procedure to deal with a primitive type of configurations (see Lemma 14 of [9]). The main difference, however, is that while the KS algorithm resorts to more sophisticated and round-intensive procedures to tackle other configurations (e.g., the one in Figure 1b), we proceed in the same manner for all configurations anyway. This simplification is the side product of a theorem established in this paper, which shows that the cartesian product of all the *unary* relations $R''_{\{X\}}$ – one for each isolated attribute X (in our example, $R''_{\{\mathtt{G}\}}$, $R''_{\{\mathtt{H}\}}$, and $R''_{\{\mathtt{L}\}}$) – is not too large *on average*, over all the possible configurations. This property allows us to *duplicate* such cartesian products onto a large number of machines, which in turn is the key reason why the hypercube algorithm can be invoked to finish Step 2 in one round. In fact, handling those unary relations has been the main challenge in all the algorithms [7, 9, 10] applying the "decomposition by heavy-hitters" idea, because the binary relations obtained from the decomposition are so-called "skew-free", and hence, easy to process. In light of this, our theorem provides deeper insight into the reason why simple join queries can be processed with the optimal load.

It is worth mentioning that while our algorithm performs 3 rounds, the second round, which transmits certain statistics to assist machine allocation in the last round, incurs only a small load that is a polynomial of p and does not depend on m. In other words, the algorithm performs only 2 rounds whose loads are sensitive to m. This brings us very close to finally settling the problem with the optimal load using genuinely only 2 rounds, and leaves open the question: *is the transmission of those statistics absolutely necessary?*

2 Preliminaries

2.1 Hypergraphs

We define a *hypergraph* $\mathcal{G}$ as a pair $(\mathcal{V}, \mathcal{E})$ where:

- $\mathcal{V}$ is a finite set, where each element is called a *vertex*;
- $\mathcal{E}$ is a set of non-empty subsets of $\mathcal{V}$, where each subset is called a *hyperedge*.

Given a vertex $X \in \mathcal{V}$ and a hyperedge $e \in \mathcal{E}$, we say that X and e are *incident* to each other if $X \in e$. A vertex $X \in \mathcal{V}$ is *dangling* if it is not incident on any hyperedge in $\mathcal{E}$. In this paper, we consider only hypergraphs where there are no dangling vertices.

Two distinct vertices $X, Y \in V$ are *adjacent* to each other if there is an $e \in \mathcal{E}$ containing both X and Y. An edge e is *unary* if $|e| = 1$, or *binary* if $|e| = 2$. A *binary* hypergraph is one that has only binary edges. Given a subset $\mathcal{V}'$ of $\mathcal{V}$, we define the subgraph *induced* by $\mathcal{V}'$ as $(\mathcal{V}', \mathcal{E}')$ where $\mathcal{E}' = \{\mathcal{V}' \cap e \mid e \in \mathcal{E} \wedge \mathcal{V}' \cap e \neq \emptyset\}$.

2.2 Fractional Edge Coverings and Packings

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, and W a function mapping $\mathcal{E}$ to real values in $[0, 1]$. We call $W(e)$ the *weight* of the hyperedge e, and $\sum_{e \in \mathcal{E}} W(e)$ the *total weight* of W. Given a vertex $X \in \mathcal{V}$, we refer to $\sum_{e \in \mathcal{E}: X \in e} W(e)$, which is the sum of the weights of the edges incident to X, as the *weight* of X.

W is a *fractional edge covering* of $\mathcal{G}$ if the weight of every vertex $X \in \mathcal{V}$ is at least 1. The *fractional edge covering number* of $\mathcal{G}$, which is denoted as $\rho(\mathcal{G})$, equals the smallest total weight of all the fractional edge coverings. W is a *fractional edge packing* if the weight of every vertex $X \in \mathcal{V}$ is at most 1. The *fractional edge packing number* of $\mathcal{G}$, which is denoted as $\tau(\mathcal{G})$, equals the largest total weight of all the fractional edge packings. A fractional edge packing W is *tight* if it is simultaneously also a fractional edge covering. Likewise, a fractional edge covering W is *tight* if it is simultaneously also a fractional edge packing. Note that in both cases it must hold that the weight of every vertex $X \in \mathcal{V}$ is exactly 1.

The lemma below lists several useful properties of binary hypergraphs:

▶ **Lemma 1.** *If $\mathcal{G}$ is binary, then:*

- *$\rho(\mathcal{G}) + \tau(\mathcal{G}) = |\mathcal{V}|$ and $\rho(\mathcal{G}) \geq \tau(\mathcal{G})$, where the two equalities hold if and only if $\mathcal{G}$ admits a tight fractional edge packing (or covering).*
- *$\mathcal{G}$ admits a fractional edge packing W of total weight $\tau(\mathcal{G})$ such that*
  1. *The weight of every vertex $X \in \mathcal{V}$ is either 0 or 1.*
  2. *If $\mathcal{Z}$ is the set of vertices in $\mathcal{V}$ with weight 0, then $\rho(\mathcal{G}) - \tau(\mathcal{G}) = |\mathcal{Z}|$.*

Proof. The first bullet is proved in Theorem 2.2.7 of [13]. The fractional edge packing W in Theorem 2.1.5 of [13] satisfies Condition 1 of the second bullet. This W also fulfills Condition 2, as is proved in Lemma 16 of [9]. ◀

Example. Suppose that $\mathcal{G}$ is the binary hypergraph in Figure 1a. It has a fractional edge covering number $\rho(\mathcal{G}) = 6.5$, as is achieved by the function W_1 that maps {G, F}, {D, K}, {I, J}, {E, H}, and {E, L} to 1, {A, B}, {A, C}, and {B, C} to 1/2, and the other edges to 0. Its fractional edge packing number is $\tau(\mathcal{G}) = 5.5$, achieved by the function W_2 which is the same as W_1 except that W_2 maps {E, L} to 0. W_2 also satisfies both conditions of the second bullet (notice that $\mathcal{Z} =$ {L}). ◀

2.3 Hypergraph of a Join Query and the AGM Bound

Every join query $\mathcal{Q}$ defines a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = attset(\mathcal{Q})$ and $\mathcal{E} = \{scheme(R) \mid R \in \mathcal{Q}\}$. When $\mathcal{Q}$ is scheme-clean, for each hyperedge $e \in \mathcal{E}$ we denote by R_e the input relation $R \in \mathcal{Q}$ with $e = scheme(R)$. Note also that $\mathcal{G}$ must be binary if $\mathcal{Q}$ is simple. The following result is known as the *AGM bound*:

▶ **Lemma 2** ([4]). *Let $\mathcal{Q}$ be a scheme-clean join query, and W be a fractional edge covering of the hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ defined by $\mathcal{Q}$. Then, $|Join(\mathcal{Q})| \leq \prod_{e \in \mathcal{E}} |R_e|^{W(e)}$.*

2.4 MPC Building Blocks

Cartesian Products. Suppose that R and S are relations with disjoint schemes. Their *cartesian product*, denoted as $R \times S$, is a relation over $scheme(R) \cup scheme(S)$ which consists of all the tuples $\boldsymbol{u}$ over $scheme(R) \cup scheme(S)$ such that $\boldsymbol{u}[scheme(R)] \in R$ and $\boldsymbol{u}[scheme(S)] \in S$. Sometimes, we need to compute the *cartesian product* of a set of relations $\mathcal{Q} = \{R_1, R_2, ..., R_t\}$ $(t \geq 2)$ with mutually disjoint schemes. For convenience, define $CP(\mathcal{Q})$ as a short form for $R_1 \times R_2 \times ... \times R_t$. Note that $CP(\mathcal{Q})$ can also be regarded as the result $Join(\mathcal{Q})$ of the join query $\mathcal{Q}$.

Two results regarding cartesian products will be useful:

▶ **Lemma 3** ([3]). *Given $\mathcal{Q} = \{R_1, R_2, ..., R_t\}$, we can compute $CP(\mathcal{Q})$ in one round with load $\tilde{O}(|CP(\mathcal{Q})|^{\frac{1}{t}} / p^{\frac{1}{t}})$ using p machines.*

▶ **Lemma 4** ([9]). *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two join queries that have input sizes at most m, and satisfy the condition that attset$(\mathcal{Q}_1) \cap$ attset$(\mathcal{Q}_2) = \emptyset$. Suppose that Join$(\mathcal{Q}_1)$ can be computed in one round with load $\tilde{O}(m/p_1^{1/t_1})$ using p_1 machines, and similarly, Join$(\mathcal{Q}_1)$ can be computed in one round with load $\tilde{O}(m/p_2^{1/t_2})$ using p_2 machines. Then Join$(\mathcal{Q}_1) \times$ Join$(\mathcal{Q}_2)$ can be computed in one round with load $\tilde{O}(m/\min\{p^{1/t_1}, p^{1/t_2}\})$ using $p_1 p_2$ machines.*

Skew-Free Queries. Let $\mathcal{Q}$ be a join query on binary relations. Regardless of whether $\mathcal{Q}$ is simple, it can be solved in a single round with a small load if no value appears too often in its input relations. Denote by m the input size of $\mathcal{Q}$. Set $k = |attset(\mathcal{Q})|$, and list out the attributes in $attset(\mathcal{Q})$ as $X_1, ..., X_k$. Let p_i be a positive integer, $i \in [1, k]$, which is referred to as the *share* of X_i. A relation $R \in \mathcal{Q}$ with scheme $\{X_i, X_j\}$ is *skew-free* if every value $x \in \mathbf{dom}$ fulfills both conditions below:

- R has $O(m/p_i)$ tuples $\boldsymbol{u}$ with $\boldsymbol{u}(X_i) = x$;
- R has $O(m/p_j)$ tuples $\boldsymbol{u}$ with $\boldsymbol{u}(X_j) = x$.

Define $share(R) = p_i \cdot p_j$. If every $R \in \mathcal{Q}$ is skew-free, $\mathcal{Q}$ is *skew-free*, and can be solved with the following guarantee:

▶ **Lemma 5** ([5]). *A skew-free query $\mathcal{Q}$ can be answered in one round with load $\tilde{O}(m/\min_{R\in\mathcal{Q}} share(R))$ using $\prod_{i=1}^{k} p_i$ machines.*

One-Attribute Reduction. Let $X \in \mathbf{att}$ be an attribute. We have $a \geq 1$ unary relations $R_1, ..., R_a$ over $\{X\}$, and $b \geq 1$ binary relations $S_1, ..., S_b$ such that S_i $(1 \leq i \leq b)$ is a relation over $\{X, Y_i\}$ where Y_i is an attribute in $\mathbf{att}$ different from X. Here, both a and b are constants. Our objective is to compute $S_i^{\#}$ which includes all tuples $\boldsymbol{u} \in S_i$ satisfying the condition that $\boldsymbol{u}(X) \in \bigcap_{j=1}^{a} R_j$. We will refer to this operation as *one-attribute reduction*. Let $n = \sum_{j=1}^{a} |R_i| + \sum_{i=1}^{b} |S_i|$. A value $x \in \mathbf{dom}$ is a *heavy-hitter* if at least n/p tuples in *some* S_i $(1 \leq i \leq b)$ use x as their X-values, where p is the number of machines assigned to the operation.

▶ **Lemma 6.** *One-attribute reduction can be performed in one round with load $\tilde{O}(p + n/p)$ using p machines, provided that each machine knows all the heavy-hitters.*

Proof. See Appendix A. ◀

It is worth mentioning that the above lemma is an extension of a result in [10]. The p term in the load can actually be eliminated, if the machine knows also additional statistics of the heavy-hitters. We do not need to be bothered with such details because the term is affordable for our purposes.

3 A Taxonomy of the Join Result

Recall that Section 1 outlined a method to partition the join result based on heavy and light values. In this section, we formalize this method and establish some fundamental properties. Denote by $\mathcal{Q}$ a simple join query, by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the hypergraph defined by $\mathcal{Q}$, by m the input size of $\mathcal{Q}$, and by k the number of attributes in $attset(\mathcal{Q})$.

Heavy and Light Values. Fix an arbitrary integer $\lambda \in [1, m]$. A value $x \in \mathbf{dom}$ is

- *heavy* if there exists a relation $R \in \mathcal{Q}$ and some attribute $X \in \mathit{scheme}(R)$ such that $|\{\boldsymbol{u} \in R \mid \boldsymbol{u}(X) = x\}| \geq m/\lambda$;
- *light* if x is not heavy, but appears in at least one relation $R \in \mathcal{Q}$.

Since each relation has $O(1)$ attributes, the number of heavy values is $O(\lambda)$.

Configurations. Let $\mathcal{H}$ be an arbitrary (possibly empty) subset of $\mathit{attset}(\mathcal{Q})$. A *configuration* of $\mathcal{H}$ is a tuple $\boldsymbol{\eta}$ over $\mathcal{H}$ such that $\boldsymbol{\eta}(X)$ is heavy for every $X \in \mathcal{H}$. Obviously, each $\mathcal{H} \subseteq \mathit{attset}(\mathcal{Q})$ has at most $O(\lambda^{|\mathcal{H}|})$ configurations.

Residue Relations and Residue Queries. Now, let us fix a configuration $\boldsymbol{\eta}$ of $\mathcal{H}$, and aim to produce all the result tuples $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q})$ *consistent* with the configuration, namely, $\boldsymbol{u}$ satisfies

- $\boldsymbol{u}(X) = \boldsymbol{\eta}(X)$ for every $X \in \mathcal{H}$, and
- $\boldsymbol{u}(X)$ is light for every $X \in \mathit{attset}(\mathcal{Q}) \setminus \mathcal{H}$.

We will take a few steps to define what is the *residue query* under $\boldsymbol{\eta}$, which is denoted as $\mathcal{Q}'_{\boldsymbol{\eta}}$, whose result is precisely the set of all the qualifying $\boldsymbol{u}$.

Let e be a hyperedge in $\mathcal{E}$ that is not subsumed by $\mathcal{H}$, i.e., e has at least one attribute outside $\mathcal{H}$. This hyperedge is said to be *active* on $\boldsymbol{\eta}$. Define $e' = e \setminus \mathcal{H}$, namely, the set of attributes in e that are outside $\mathcal{H}$. The relation $R_e \in \mathcal{Q}$ defines a *residue relation* under $\boldsymbol{\eta}$ which

- is over e' and
- consists of every tuple $\boldsymbol{v}$ that is the projection of some tuple $\boldsymbol{w} \in R_e$ "consistent" with $\boldsymbol{\eta}$, namely: (i) $\boldsymbol{w}(X) = \boldsymbol{\eta}(X)$ for every $X \in e \cap \mathcal{H}$, (ii) $\boldsymbol{w}(Y)$ is light for every $Y \in e'$, and (iii) $\boldsymbol{v} = \boldsymbol{w}[e']$.

The residue relation is denoted as $R'_{e'|\boldsymbol{\eta}[e \setminus e']}$, where $\boldsymbol{\eta}[e \setminus e']$ is the projection of $\boldsymbol{\eta}$ on $e \setminus e'$, as was introduced in Section 1.1.

We can now define the *residue query* as

$$\mathcal{Q}'_{\boldsymbol{\eta}} = \left\{ R_{e'|\boldsymbol{\eta}[e \setminus e']} \;\middle|\; e \in \mathcal{E},\ e \text{ active on } \boldsymbol{\eta} \right\}.$$

Example. Suppose that $\mathcal{Q}$ is the query discussed in Section 1.3 with its hypergraph $\mathcal{G}$ given in Figure 1a. Consider the configuration $\boldsymbol{\eta}$ of $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$ where $\boldsymbol{\eta}[\mathtt{D}] = \mathtt{d}$, $\boldsymbol{\eta}[\mathtt{E}] = \mathtt{e}$, $\boldsymbol{\eta}[\mathtt{F}] = \mathtt{f}$, and $\boldsymbol{\eta}[\mathtt{K}] = \mathtt{k}$. If e is the edge $\{\mathtt{A}, \mathtt{D}\}$, then $e' = \{\mathtt{A}\}$ and $\boldsymbol{\eta}[e \setminus e'] = \boldsymbol{\eta}[\{\mathtt{D}\}] = \mathtt{d}$, such that $R'_{e'|\boldsymbol{\eta}[e \setminus e']}$ is the relation $R'_{\{\mathtt{A}\}|\mathtt{d}}$ mentioned in Section 1.3. If e is the edge $\{\mathtt{A}, \mathtt{B}\}$, on the other hand, then $e' = \{\mathtt{A}, \mathtt{B}\}$ and $\boldsymbol{\eta}[e \setminus e'] = \emptyset$, so that $R'_{e'|\boldsymbol{\eta}[e \setminus e']}$ can be written as $R'_{\{\mathtt{A},\mathtt{B}\}|\emptyset}$, and is the relation $R'_{\{\mathtt{A},\mathtt{B}\}}$ in Section 1.3. The residue query $\mathcal{Q}'_{\boldsymbol{\eta}}$ is precisely the query $\mathcal{Q}'$ described in Section 1.3. *(to be continued)*▲

It is rudimentary to verify

$$\mathit{Join}(\mathcal{Q}) = \bigcup_{\mathcal{H}} \Big(\bigcup_{\text{config. } \boldsymbol{\eta} \text{ of } \mathcal{H}} \mathit{Join}(\mathcal{Q}'_{\boldsymbol{\eta}}) \times \{\boldsymbol{\eta}\} \Big). \tag{1}$$

Denote by $m_{\boldsymbol{\eta}}$ the input size of $\mathcal{Q}'_{\boldsymbol{\eta}}$. The next proposition says that total input size of all the residue queries is not too large:

► **Proposition 7.** *If $\mathcal{Q}$ is simple, $\sum_{\text{config. } \boldsymbol{\eta} \text{ of } \mathcal{H}} m_{\boldsymbol{\eta}} = O(m \cdot \lambda^{k-2})$ holds for every $\mathcal{H} \subseteq \mathit{attset}(\mathcal{Q})$.*

Proof. See Appendix B. ◄

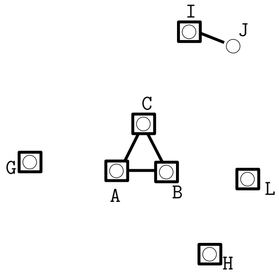


Figure 2 Subgraph induced by $\mathcal{L}$.

4 A Join Computation Framework

Given a simple join query $\mathcal{Q}$, we will concentrate on an arbitrary subset $\mathcal{H}$ of $attset(\mathcal{Q})$ in this section. In Sections 4.1-4.2, we will generalize the strategy illustrated in Section 1.3 into a formal framework for producing

$$\bigcup_{\text{config. } \boldsymbol{\eta} \text{ of } \mathcal{H}} Join(\mathcal{Q}'_{\boldsymbol{\eta}}). \tag{2}$$

Section 4.3 will then establish a theorem on this framework, which is the core of the techniques proposed in this paper.

4.1 Removing the Attributes in $\mathcal{H}$

We will refer to each attribute in $\mathcal{H}$ as a *heavy attribute*. Define $\mathcal{L} = scheme(\mathcal{Q}) \setminus \mathcal{H}$, where each attribute is called a *light attribute*. Denote by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the hypergraph defined by $\mathcal{Q}$. An edge $e \in \mathcal{E}$ is (i) a *light edge* if e contains two light attributes, or (ii) a *cross edge* if e contains a heavy attribute and a light attribute. A light attribute $X \in \mathcal{L}$ is a *border attribute* if it appears in at least one cross edge e of $\mathcal{G}$; note that this implies $e \setminus \mathcal{H} = \{X\}$. Denote by $\mathcal{G}' = (\mathcal{L}, \mathcal{E}')$ the subgraph of $\mathcal{G}$ induced by $\mathcal{L}$. A vertex $X \in \mathcal{L}$ is *isolated* if $\{X\}$ is the only edge in $\mathcal{E}'$ incident to X. Define $\mathcal{I}$ to be the set of isolated vertices in $\mathcal{G}'$.

Example (cont.). As before, let $\mathcal{Q}$ be the join query whose hypergraph $\mathcal{G}$ is shown in Figure 1a, and set $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$. $\mathcal{L}$ includes all the white vertices in Figure 1b. $\{\mathtt{A}, \mathtt{B}\}$ is a light edge, $\{\mathtt{A}, \mathtt{D}\}$ is a cross edge, while $\{\mathtt{D}, \mathtt{K}\}$ is neither a light edge nor a cross edge. All the vertices in $\mathcal{L}$ except J are border vertices. Figure 2 shows the subgraph of $\mathcal{G}$ induced by $\mathcal{L}$, where a unary edge is represented by a box and a binary edge by a segment. Notice that no unary edge covers J. Vertices G, H, and L are the only isolated vertices. *(to be continued)*▲

4.2 Semi-Join Reduction

Recall that every configuration $\boldsymbol{\eta}$ of $\mathcal{H}$ gives rise to a residue query $\mathcal{Q}'_{\boldsymbol{\eta}}$. Next, we will transform $\mathcal{Q}'_{\boldsymbol{\eta}}$ into an alternative join query $\mathcal{Q}''_{\boldsymbol{\eta}}$ which, as shown in the next section, can be processed in a single round in the MPC model.

First of all, observe that the hypergraph defined by $\mathcal{Q}'_{\boldsymbol{\eta}}$ is always $\mathcal{G}' = (\mathcal{L}, \mathcal{E}')$, regardless of $\boldsymbol{\eta}$. Consider a border attribute $X \in \mathcal{L}$, and a cross edge e of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ incident to X. As explained in Section 3, the input relation $R_e \in \mathcal{Q}$ defines a unary residue relation $R'_{e'|\boldsymbol{\eta}[e \setminus e']} \in \mathcal{Q}'_{\boldsymbol{\eta}}$ where $e' = e \setminus \mathcal{H}$. Since $e' = \{X\}$, we can as well write the relation as $R'_{\{X\}|\boldsymbol{\eta}[e \setminus \{X\}]}$. Now that every such relation has the same scheme $\{X\}$, we can define:

$$R''_{\{X\}|\boldsymbol{\eta}} = \bigcap_{\text{cross edge } e \text{ containing } X} R'_{\{X\}|\boldsymbol{\eta}[e \setminus \{X\}]}. \tag{3}$$

Example (cont.). Let $\mathcal{H}$ and $\boldsymbol{\eta}$ be the same as in the earlier description of this example. Set X to the border attribute $\mathtt{A}$. If e is the cross edge $\{\mathtt{A}, \mathtt{D}\}$, the example in Section 3 has shown that $R'_{e'|\boldsymbol{\eta}[e \setminus e']}$ is the relation $R'_{\{\mathtt{A}\}|\mathtt{d}}$ obtained in Section 1.3. Similarly, if e is the cross edge $\{\mathtt{A}, \mathtt{E}\}$, $R'_{e'|\boldsymbol{\eta}[e \setminus e']}$ is the relation $R'_{\{\mathtt{A}\}|\mathtt{e}}$ obtained in Section 1.3. As $\mathtt{A}$ is contained only in these two cross edges, $R''_{\{A\}|\boldsymbol{\eta}}$ is the intersection of $R'_{\{\mathtt{A}\}|\mathtt{d}}$ and $R'_{\{\mathtt{A}\}|\mathtt{e}}$, and corresponds to the relation $R''_{\{A\}}$ given in Section 1.3. *(to be continued)*▲

Consider a light edge $e = \{X, Y\}$ in $\mathcal{G}$. Recall that R_e defines a residue relation $R'_{e'|\boldsymbol{\eta}[e \setminus e']} \in \mathcal{Q}'_{\boldsymbol{\eta}}$, which can be written as $R'_{e|\emptyset}$ because $e' = e \setminus \mathcal{H} = e$. We define $R''_{e|\boldsymbol{\eta}}$ as a relation over e which consists of every tuple $\boldsymbol{u} \in R'_{e|\emptyset}$ satisfying both conditions below:

- (applicable only if X is a border attribute) $\boldsymbol{u}(X) \in R''_{X|\boldsymbol{\eta}}$;
- (applicable only if Y is a border attribute) $\boldsymbol{u}(Y) \in R''_{Y|\boldsymbol{\eta}}$.

Note that if neither X nor Y is a border attribute, then $R''_{e|\boldsymbol{\eta}} = R'_{e|\emptyset}$.

Example (cont.). Let us concentrate the light edge $e = \{\mathtt{A}, \mathtt{B}\}$. The example in Section 3 has explained that $R'_{e'|\boldsymbol{\eta}[e \setminus e']} = R'_{\{\mathtt{A},\mathtt{B}\}|\emptyset}$ is the relation $R'_{\{\mathtt{A},\mathtt{B}\}}$ obtained in Section 1.3. As $\mathtt{A}$ and $\mathtt{B}$ are both border attributes, $R''_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}}$ includes all the tuples in $R'_{\{\mathtt{A},\mathtt{B}\}}$ that take a value in $R''_{\{\mathtt{A}\}|\boldsymbol{\eta}}$ on attribute $\mathtt{A}$ and a value in $R''_{\{\mathtt{B}\}|\boldsymbol{\eta}}$ on attribute $\mathtt{B}$. Note that $R''_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}}$ corresponds to the relation $R''_{\{\mathtt{A},\mathtt{B}\}}$ given in Section 1.3. *(to be continued)*▲

Every vertex $X \in \mathcal{I}$ must be a border attribute, and thus must have $R''_{X|\boldsymbol{\eta}}$ defined. Therefore, we can legally define:

$$\mathcal{Q}''_{light|\boldsymbol{\eta}} = \{R''_{e|\boldsymbol{\eta}} \mid \text{light edge } e \in \mathcal{E}\}$$
$$\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}} = \{R''_{\{X\}|\boldsymbol{\eta}} \mid X \in \mathcal{I}\}$$
$$\mathcal{Q}''_{\boldsymbol{\eta}} = \mathcal{Q}''_{light|\boldsymbol{\eta}} \cup \mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}.$$

Notice that the join queries $\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}$, $\mathcal{Q}''_{light|\boldsymbol{\eta}}$, and $\mathcal{Q}''_{\boldsymbol{\eta}}$ are all scheme-clean.

Example (cont.). $\mathcal{Q}''_{light|\boldsymbol{\eta}}$ consists of $R''_{\{\mathtt{A},\mathtt{B}\}}$, $R''_{\{\mathtt{A},\mathtt{C}\}}$, $R_{\{\mathtt{B},\mathtt{C}\}}$, and $R_{\{\mathtt{I},\mathtt{J}\}}$, and $\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}$ consists of $R''_{\{\mathtt{G}\}}$, $R''_{\{\mathtt{H}\}}$, and $R''_{\{\mathtt{L}\}}$, where all the relation names follow those given in Section 1.3. ◀

▶ **Proposition 8.** *$Join(\mathcal{Q}'_{\boldsymbol{\eta}}) = Join(\mathcal{Q}''_{\boldsymbol{\eta}}) = CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}) \times Join(\mathcal{Q}''_{light|\boldsymbol{\eta}})$.*

Proof. See Appendix C. ◀

We will refer to the above process of converting $\mathcal{Q}'_{\boldsymbol{\eta}}$ to $\mathcal{Q}''_{\boldsymbol{\eta}}$ as *semi-join reduction*, and call $\mathcal{Q}''_{\boldsymbol{\eta}}$ the *reduced query* of $\boldsymbol{\eta}$.

4.3 The Isolated Cartesian Product Theorem

We are ready to present the first main result of this paper:

▶ **Theorem 9** (The Isolated Cartesian Product Theorem).

$$\sum_{\text{config. } \boldsymbol{\eta} \text{ of } \mathcal{H}} \left| CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}) \right| \quad = \quad O\Big(\lambda^{2(\rho - |\mathcal{I}|) - |\mathcal{L} \setminus \mathcal{I}|} \cdot m^{|\mathcal{I}|}\Big) \tag{4}$$

where ρ is the fractional edge covering number of $\mathcal{Q}$.

The rest of the section serves as a proof of the above theorem. To start with, define $\mathcal{F}$ to be the set of attributes in $\mathcal{H}$ that are adjacent to at least one isolated vertex in $\mathcal{G}$. The left hand side of (4) can be bounded by looking at only the configurations of $\mathcal{F}$:

▶ **Lemma 10.** $\sum_{config.\ \boldsymbol{\eta}\ of\ \mathcal{H}} \left|CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}})\right| = O\left(\lambda^{|\mathcal{H}|-|\mathcal{F}|}\right) \cdot \sum_{config.\ \boldsymbol{\eta}'\ of\ \mathcal{F}} \left|CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}'})\right|$.

Proof. See Appendix D. ◀

Now, let us take a fractional edge packing W of the hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that obeys the second bullet of Lemma 1. Denote by τ the total weight of W which, by definition of W, is the fractional edge packing number of $\mathcal{G}$. Define:

$$\mathcal{Z} \quad = \quad \Big\{X \in \mathcal{V} \mid \sum_{e \in \mathcal{E}: X \in e} W(e) = 0\Big\}$$

that is, $\mathcal{Z}$ is the set of vertices with weight 0 under W. Set $\mathcal{I}_0 = \mathcal{I} \cap \mathcal{Z}$ and $\mathcal{I}_1 = \mathcal{I} \setminus \mathcal{I}_0$. Since W satisfies Condition 1 of the second bullet in Lemma 1, we know that every vertex in $\mathcal{I}_1$ has weight 1, while every vertex in $\mathcal{I}_0$ has weight 0.

Example. Let $\mathcal{G}$ be the hypergraph in Figure 1a. As explained by the example in Section 2.2, the fractional edge packing number τ of $\mathcal{G}$ is achieved by the function W that maps {G, F}, {D, K}, {I, J}, and {E, H} to 1, {A, B}, {A, C}, and {B, C} to 1/2, and the other edges to 0; $\mathcal{Z}$ contains a single vertex L. Setting $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$ yields $\mathcal{I} = \{\mathtt{G}, \mathtt{H}, \mathtt{L}\}$, $\mathcal{I}_0 = \{\mathtt{L}\}$, and $\mathcal{I}_1 = \{\mathtt{G}, \mathtt{H}\}$. $\mathcal{F} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}\}$, noticing that K is not adjacent to any isolated vertex. *(to be continued)*▲

We now present a crucial lemma which is in fact a stronger version of Theorem 9:

▶ **Lemma 11.** $\sum_{config.\ \boldsymbol{\eta}'\ of\ \mathcal{F}} \left|CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}'})\right| = O\Big(\lambda^{|\mathcal{F}|-|\mathcal{I}_1|} \cdot m^{|\mathcal{I}|}\Big)$.

Before proving the above lemma, let us first see how it can be used to complete the proof of Theorem 9. By combining Lemmas 10 and 11, we know that the left hand side of (4) is $O(\lambda^{|\mathcal{H}|-|\mathcal{I}_1|} \cdot m^{|\mathcal{I}|})$. Hence, it suffices to prove

$$\begin{aligned}
|\mathcal{H}| - |\mathcal{I}_1| &\leq 2(\rho - |\mathcal{I}|) - |\mathcal{L} \setminus \mathcal{I}| \ \Leftrightarrow \\
|\mathcal{H}| + |\mathcal{L} \setminus \mathcal{I}| + |\mathcal{I}| + |\mathcal{I}| - |\mathcal{I}_1| &\leq 2\rho \ \Leftrightarrow \\
|\mathcal{V}| - \rho + |\mathcal{I}_0| &\leq \rho \text{ (note: } |\mathcal{V}| = |\mathcal{H}| + |\mathcal{L} \setminus \mathcal{I}| + |\mathcal{I}|) \Leftrightarrow \\
\tau + |\mathcal{I}_0| &\leq \rho \text{ (note: } \rho + \tau = |\mathcal{V}| \text{ by Lemma 1)}
\end{aligned}$$

which is true because $\rho - \tau = |\mathcal{Z}|$ by Condition 2 of the second bullet in Lemma 1, and $\mathcal{I}_0 \subseteq \mathcal{Z}$.

Proof of Lemma 11. Our idea is to construct a set $\mathcal{Q}^*$ of relations such that $Join(\mathcal{Q}^*)$ has a result size at least the left hand side of the inequality in Lemma 11. Then, we will prove that the hypergraph of $\mathcal{Q}^*$ has a certain fractional edge covering which, together with the AGM bound, yields an upper bound on $|Join(\mathcal{Q}^*)|$ which happens to be the right hand side of the inequality.

We construct $\mathcal{Q}^*$ as follows. Initially, set $\mathcal{Q}^*$ to $\emptyset$. For every cross edge $e \in \mathcal{E}$ incident to an isolated vertex, add to $\mathcal{Q}^*$ a relation $R^*_e = R_e$. For every $X \in \mathcal{F}$, add a unary relation $R^*_{\{X\}}$ to $\mathcal{Q}^*$ which consists of all the heavy values on attribute X. Note that $R^*_{\{X\}}$ has $O(\lambda)$ tuples. Finally, for every $Y \in \mathcal{I}_0$, add a unary relation $R^*_{\{Y\}}$ to $\mathcal{Q}^*$ which contains all the heavy and light values on attribute Y.

Define $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ as the hypergraph defined by $\mathcal{Q}^*$. Note that $\mathcal{V}^* = \mathcal{I} \cup \mathcal{F}$, while $\mathcal{E}^*$ consists of all the cross edges in $\mathcal{G}$, $|\mathcal{F}|$ unary edges $\{X\}$ for every $X \in \mathcal{F}$, and $|\mathcal{I}_0|$ unary edges $\{Y\}$ for every $Y \in \mathcal{I}_0$.

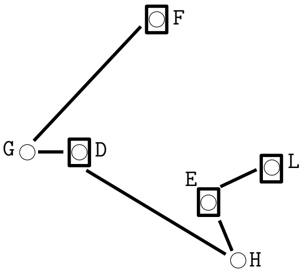


Figure 3 Illustration of $\mathcal{Q}^*$.

Example (cont.). Figure 3 shows the hypergraph of the $\mathcal{Q}^*$ constructed, where as before a box and a segment represent a unary and a binary edge, respectively. *(to be continued)*▲

▶ **Lemma 12.** $\sum_{config.\ \boldsymbol{\eta}'\ of\ \mathcal{F}} \left| CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}'}) \right| \le |Join(\mathcal{Q}^*)|$.

Proof. See Appendix E. ◀

▶ **Lemma 13.** *$\mathcal{G}^*$ admits a tight fractional edge covering $\mathcal{W}^*$ satisfying $\sum_{X \in F} W^*(\{X\}) = |\mathcal{F}| - |\mathcal{I}_1|$.*

Proof. Recall that our proof of Theorem 9 began with a fractional edge packing W of $\mathcal{G}$. We construct a desired function W^* from W as follows. First, for every cross edge $e \in \mathcal{E}$, set $W^*(e) = W(e)$. Observe that every edge in $\mathcal{E}$ incident to $Y \in \mathcal{I}$ must be a cross edge. Hence, $\sum_{\text{binary } e \in \mathcal{E}^*: Y \in e} W^*(e)$ is precisely the weight of Y under W. By definition of W, we thus have ensured $\sum_{\text{binary } e \in \mathcal{E}^*: Y \in e} W^*(e) = 1$ for each $Y \in \mathcal{I}_1$, and $\sum_{\text{binary } e \in \mathcal{E}^*: Y \in e} W^*(e) = 0$ for each $Y \in \mathcal{I}_0$. As a second step, we set $W^*(\{Y\}) = 1$ for each $Y \in \mathcal{I}_0$ so that the edges in $\mathcal{E}^*$ containing Y have a total weight of 1.

It remains to make sure that each attribute $X \in \mathcal{F}$ has a weight 1 under W^*. Since W is a fractional edge packing of $\mathcal{G}$, it must hold that $\sum_{\text{binary } e \in \mathcal{E}^*: X \in e} W(e) \le 1$. This permits us to assign the following weight to the unary edge $\{X\}$:

$$W^*(\{X\}) \quad = \quad 1 - \sum_{\text{binary } e \in \mathcal{E}^*: X \in e} W(e).$$

This finishes the design of W^* which is now a tight fractional edge covering of $\mathcal{G}^*$. Clearly:

$$\sum_{X \in \mathcal{F}} W^*(\{X\}) \quad = \quad |\mathcal{F}| - \sum_{X \in \mathcal{F}} \sum_{\text{binary } e \in \mathcal{E}^*: X \in e} W(e). \tag{5}$$

Every binary edge $e \in \mathcal{E}^*$ contains a vertex in $\mathcal{F}$ and a vertex in $\mathcal{I}$. Therefore:

$$\sum_{X \in \mathcal{F}} \sum_{\text{binary } e \in \mathcal{E}^*: X \in e} W(e) \quad = \quad \sum_{Y \in \mathcal{I}} \sum_{\text{binary } e \in \mathcal{E}^*: Y \in e} W(e) = |\mathcal{I}_1|.$$

Putting together the above equation with (5) completes the proof. ◀

Example (cont.). For the $\mathcal{G}^*$ in Figure 3, a fractional edge covering in Lemma 13 is given by the function W^* that maps $\{\mathtt{G}, \mathtt{F}\}$, $\{\mathtt{E}, \mathtt{H}\}$, $\{\mathtt{D}\}$, and $\{\mathtt{L}\}$ to 1, and the other edges to 0. Note that $\sum_{X \in F} W^*(\{X\}) = W^*(\{\mathtt{D}\}) = 1$, same as $|\mathcal{F}| - |\mathcal{I}_1| = 3 - 2 = 1$. ◀

The AGM bound in Lemma 2 tells us that

$$\begin{aligned} \mathit{Join}(\mathcal{Q}^*) &\leq \prod_{e \in \mathcal{E}^*} |R^*_e|^{W^*(e)} = \Big(\prod_{X \in \mathcal{F}} |R^*_{\{X\}}|^{W^*(\{X\})}\Big)\Big(\prod_{Y \in \mathcal{I}} \prod_{e \in \mathcal{E}^* : Y \in e} |R^*_e|^{W^*(e)}\Big) \\ &= \Big(\prod_{X \in \mathcal{F}} (O(\lambda))^{W^*(\{X\})}\Big)\Big(\prod_{Y \in \mathcal{I}} \prod_{e \in \mathcal{E}^* : Y \in e} m^{W^*(e)}\Big) \\ \text{(by Lemma 13)} &= O(\lambda^{|\mathcal{F}| - |\mathcal{I}_1|}) \cdot m^{|\mathcal{I}|} \end{aligned}$$

which completes the proof of Lemma 11.

5 A 5-Round MPC Algorithm

We now proceed to implement the strategy discussed in the previous section under the MPC model. Our objective in this section is to explain how the isolated cartesian product theorem can be utilized to answer a simple join query $\mathcal{Q}$ with the optimal load $O(m/p^{1/\rho})$ in a rather straightforward manner. Hence, we intentionally leave out the optimization tricks to reduce the number of rounds, but even so, our algorithm finishes in only 5 rounds. Those tricks are the topic of the next section.

A *statistical record* is defined as a tuple (R, X, x, cnt), where R is a relation in $\mathcal{Q}$, X an attribute in $\mathit{scheme}(R)$, x a value in **dom**, and cnt the number of tuples $\boldsymbol{u} \in R$ with $\boldsymbol{u}(X) = x$. Specially, $(R, \emptyset, \mathit{nil}, \mathit{cnt})$ is also regarded as a statistical record where cnt gives the number of tuples in R that use only light values. A *histogram* is defined as the set of statistical records for all possible R, X, and x satisfying (i) $\mathit{cnt} = \Omega(m/p^{1/\rho})$, or (ii) $X = \emptyset$ (and, hence $x = \mathit{nil}$); note that there are only $O(p^{1/\rho})$ such records. We assume that every machine has a local copy of the histogram. It is worth mentioning that all existing join algorithms [5,9], which strive to finish in a *specifically* small – rather than just asymptotically constant – number of rounds, demand that each machine should be preloaded with $p^{O(1)}$ statistical records.

Henceforth, the value of λ will be fixed to $\Theta(p^{1/(2\rho)})$. We focus on explaining how to compute (2) for an arbitrary subset $\mathcal{H}$ of $\mathit{attset}(\mathcal{Q})$. Set $k = |\mathit{attset}(\mathcal{Q})|$. As $\mathit{attset}(\mathcal{Q})$ has $2^k = O(1)$ subsets, processing all of them in parallel increases the load only by a constant factor, and definitely discovers the entire $\mathit{Join}(\mathcal{Q})$, as is guaranteed by (1). Our algorithm produces (2) in three steps:

1. Generate the input relations of the residue query $\mathcal{Q}'_{\boldsymbol{\eta}}$ of every configuration $\boldsymbol{\eta}$ of $\mathcal{H}$.
2. Generate the input relations of the reduced query $\mathcal{Q}''_{\boldsymbol{\eta}}$ of every $\boldsymbol{\eta}$.
3. Evaluate $\mathcal{Q}''_{\boldsymbol{\eta}}$ for every $\boldsymbol{\eta}$.

The number of configurations of $\mathcal{H}$ is $O(\lambda^{|\mathcal{H}|}) = O(\lambda^k) = O(p^{k/(2\rho)})$, which is $O(p)$ because $\rho \geq k/2$ by the first bullet of Lemma 1. Next, we elaborate on the details of each step.

Step 1. Proposition 7 tells us that the input relations of all the residue queries have $O(m \cdot \lambda^{k-2})$ tuples in total. We allocate $p'_{\boldsymbol{\eta}} = \lceil p \cdot \frac{m_{\boldsymbol{\eta}}}{\Theta(m \cdot \lambda^{k-2})} \rceil$ machines to store the relations of $\mathcal{Q}'_{\boldsymbol{\eta}}$, so that each machine assigned to $\mathcal{Q}'_{\boldsymbol{\eta}}$ keeps on average $O(m_{\boldsymbol{\eta}}/p'_{\boldsymbol{\eta}}) = O(m \cdot \lambda^{k-2}/p) = O(m/p^{1/\rho})$ tuples, where the last equality used $\rho \geq k/2$. Since each machine $i \in [1, p]$ can use the histogram to calculate $m_{\boldsymbol{\eta}}$ precisely for each $\boldsymbol{\eta}$, it can compute locally the id range of the $m_{\boldsymbol{\eta}}$ machines responsible for $\mathcal{Q}'_{\boldsymbol{\eta}}$. If a tuple $\boldsymbol{u}$ in the local storage of machine i belongs to $Q'_{\boldsymbol{\eta}}$, the machine sends $\boldsymbol{u}$ to a random machine within that id range. Standard analysis shows that each of the $m_{\boldsymbol{\eta}}$ machines receives roughly the same number of tuples, such that this step can be done in a single round with load $\tilde{O}(m/p^{1/\rho})$.

Step 2. Now that all the input relations of each $\mathcal{Q}'_{\boldsymbol{\eta}}$ have been stored on $p'_{\boldsymbol{\eta}}$ machines, the semi-join reduction that converts $\mathcal{Q}'_{\boldsymbol{\eta}}$ to $\mathcal{Q}''_{\boldsymbol{\eta}}$ becomes a standard process [9] that can be accomplished in 2 rounds with load $\tilde{O}(m_{\boldsymbol{\eta}}/p'_{\boldsymbol{\eta}}) = \tilde{O}(m/p^{1/\rho})$.

Step 3. This step starts by letting each machine know about the value of $|CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}})|$ for every $\boldsymbol{\eta}$. For this purpose, each machine can broadcast to all machines how many tuples it has in $R''_{\{X\}|\boldsymbol{\eta}}$ for every $X \in \mathcal{I}$ and every $\boldsymbol{\eta}$. Since there are $O(p)$ different $\boldsymbol{\eta}$, at most $O(p)$ numbers are broadcast by each machine, such that the load of this round is $O(p^2)$. With all these numbers, each machine can figure out independently the values of all $|CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}})|$. We will call this round the *statistical round* henceforth.

We allocate

$$p''_{\boldsymbol{\eta}} \quad = \quad \left\lceil p \cdot \frac{|CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}})|}{\Theta(\lambda^{2(\rho-|\mathcal{I}|)-|\mathcal{L}\setminus\mathcal{I}|} \cdot m^{|\mathcal{I}|})} \right\rceil \tag{6}$$

machines to computing $\mathcal{Q}''_{\boldsymbol{\eta}}$. Theorem 9 guarantees that the total number of machines needed by all the configurations is at most p. We complete the algorithm with the lemma below:

▶ **Lemma 14.** *$\mathcal{Q}''_{\boldsymbol{\eta}}$ can be answered in one round with load $O(m/p^{1/\rho})$ using $p''_{\boldsymbol{\eta}}$ machines.*

Proof. *Join*$(\mathcal{Q}''_{\boldsymbol{\eta}})$ is the cartesian product of $CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}})$ and *Join*$(\mathcal{Q}''_{light|\boldsymbol{\eta}})$, as shown Proposition 8. By Lemma 3, if we deploy $p''_{\boldsymbol{\eta}}/\lambda^{\mathcal{L}\setminus\mathcal{I}}$ machines to compute $CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}})$ in one round, the load is

$$\tilde{O}\left(\frac{CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}})^{1/|\mathcal{I}|}}{\left(\frac{p''_{\boldsymbol{\eta}}}{\lambda^{\mathcal{L}\setminus\mathcal{I}}}\right)^{1/|\mathcal{I}|}}\right) = \tilde{O}\left(\frac{m \cdot \lambda^{\frac{2(\rho-|\mathcal{I}|)}{|\mathcal{I}|}}}{p^{1/|\mathcal{I}|}}\right) = \tilde{O}\left(\frac{m \cdot p^{\frac{2(\rho-|\mathcal{I}|)}{2\rho|\mathcal{I}|}}}{p^{1/|\mathcal{I}|}}\right) = \tilde{O}\left(\frac{m}{p^{1/\rho}}\right).$$

Regarding $\mathcal{Q}''_{light|\boldsymbol{\eta}}$, first verify that $attset(\mathcal{Q}''_{light|\boldsymbol{\eta}}) = \mathcal{L}\setminus\mathcal{I}$. Recall that the input relations of $\mathcal{Q}''_{light|\boldsymbol{\eta}}$ contain only light values. Hence, this join query is skew-free if we assign a share of λ to each attribute in $\mathcal{L} \setminus \mathcal{I}$. By Lemma 5, we can solve it in one round with load $\tilde{O}(m/\lambda^2) = \tilde{O}(m/p^{1/\rho})$ using $\lambda^{|\mathcal{L}\setminus\mathcal{I}|}$ machines.

Lemma 4 now tells us that *Join*$(\mathcal{Q}''_{\boldsymbol{\eta}})$ can be computed in one round with load $\tilde{O}(m/p^{1/\rho})$ using $(p''_{\boldsymbol{\eta}}/\lambda^{|\mathcal{L}\setminus\mathcal{I}|}) \cdot \lambda^{|\mathcal{L}\setminus\mathcal{I}|} = p''_{\boldsymbol{\eta}}$ machines. ◀

6 A 3-Round MPC Algorithm

Next, we will improve the algorithm in Section 5 by reducing the number of rounds to 3.

6.1 A New Approach to Handle Light Edges

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the hypergraph defined by the simple join query $\mathcal{Q}$ given. Fix an arbitrary subset $\mathcal{H}$ of *attset*$(\mathcal{Q})$. Recall that, for every light edge $e \in \mathcal{E}$, our 5-round algorithm needed to generate $R''_{e|\boldsymbol{\eta}}$ for every configuration $\boldsymbol{\eta}$ of $\mathcal{H}$. Next, we will describe an alternative approach to perform the join without explicitly computing $R''_{e|\boldsymbol{\eta}}$, which is crucial for obtaining a 3-round algorithm (to be presented in the next subsection).

Fix a configuration $\boldsymbol{\eta}$ of $\mathcal{H}$. Consider a light edge e of $\mathcal{G}$, and an attribute $X \in e$. Define $R^{\#X}_{e|\boldsymbol{\eta}}$ as follows:

- if X is not a border attribute, $R^{\#X}_{e|\boldsymbol{\eta}} = R'_{e|\emptyset}$;
- otherwise, $R^{\#X}_{e|\boldsymbol{\eta}}$ is a relation over e that consists of every tuple $\boldsymbol{u} \in R'_{e|\emptyset}$ satisfying $\boldsymbol{u}(X) \in R''_{X|\boldsymbol{\eta}}$.

▶ **Proposition 15.** $R''_{e|\boldsymbol{\eta}} = R^{\#X}_{e|\boldsymbol{\eta}} \bowtie R^{\#Y}_{e|\boldsymbol{\eta}}$.

Proof. See Appendix F. ◀

Example. Returning to the query $\mathcal{Q}$ in Figure 1a, consider again $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$, and the configuration $\boldsymbol{\eta}$ of $\mathcal{H}$ with $\boldsymbol{\eta}(\mathtt{D}) = \mathtt{d}$, $\boldsymbol{\eta}(\mathtt{E}) = \mathtt{e}$, $\boldsymbol{\eta}(\mathtt{F}) = \mathtt{f}$, and $\boldsymbol{\eta}(\mathtt{K}) = \mathtt{k}$. We will illustrate the above definitions by showing how to avoid explicitly computing $R''_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}}$ and $R''_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}}$ (namely, $R''_{\{\mathtt{A},\mathtt{B}\}}$ and $R''_{\{\mathtt{I},\mathtt{J}\}}$ in the description of Section 1.3). We will generate $R^{\#\mathtt{A}}_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}}$, $R^{\#\mathtt{B}}_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}}$, $R^{\#\mathtt{I}}_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}}$, and $R^{\#\mathtt{J}}_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}}$ such that $R''_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}} = R^{\#\mathtt{A}}_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}} \bowtie R^{\#\mathtt{B}}_{\{\mathtt{A},\mathtt{B}\}|\boldsymbol{\eta}}$ and $R''_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}} = R^{\#\mathtt{I}}_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}} \bowtie R^{\#\mathtt{J}}_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}}$.

Among the four relations to compute, $R^{\#\mathtt{J}}_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}}$ is the simplest because $\mathtt{J}$ is not a border attribute; hence, $R^{\#\mathtt{J}}_{\{\mathtt{I},\mathtt{J}\}|\boldsymbol{\eta}}$ equals $R'_{\{\mathtt{I},\mathtt{J}\}|\emptyset}$ (i.e., $R'_{\{\mathtt{I},\mathtt{J}\}}$ in Section 1.3). Regarding the other three relations, we will elaborate only on the generation of $R^{\#\mathtt{A}}_{\{\mathtt{A},\mathtt{B}\}}$ because the same ideas apply to $R^{\#\mathtt{B}}_{\{\mathtt{A},\mathtt{B}\}}$ and $R^{\#\mathtt{I}}_{\{\mathtt{I},\mathtt{J}\}}$.

Under $\boldsymbol{\eta}$, there are two unary residue relations defined over $\{\mathtt{A}\}$, namely, $R'_{\{\mathtt{A}\}|\mathtt{d}}$ and $R'_{\{\mathtt{A}\}|\mathtt{e}}$. The intersection of those two relations yields the unary relation $R''_{\{\mathtt{A}\}|\boldsymbol{\eta}}$ (i.e., $R''_{\{\mathtt{A}\}}$ in Section 1.3). Then, $R^{\#\mathtt{A}}_{\{\mathtt{A},\mathtt{B}\}}$ consists of every tuple $\boldsymbol{u}$ in the residue relation $R'_{\{\mathtt{A},\mathtt{B}\}|\emptyset}$ (i.e., $R'_{\{\mathtt{A},\mathtt{B}\}}$ in Section 1.3) whose $\boldsymbol{u}(A)$ appears in $R''_{\{\mathtt{A}\}|\boldsymbol{\eta}}$. ◀

Define:

$$\mathcal{Q}^{\#}_{light|\boldsymbol{\eta}} = \{R^{\#X}_{e|\boldsymbol{\eta}} \mid \text{every light edge } e \in \mathcal{E}, \text{ every attribute } X \in e\}$$
$$\mathcal{Q}^{\#}_{\boldsymbol{\eta}} = \mathcal{Q}^{\#}_{light|\boldsymbol{\eta}} \cup \mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}.$$

Proposition 15 immediately implies $Join(Q''_{\boldsymbol{\eta}}) = Join(Q^{\#}_{\boldsymbol{\eta}}) = CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}) \times Join(\mathcal{Q}^{\#}_{light|\boldsymbol{\eta}})$.

6.2 The Algorithm

We are now ready to clarify how to solve $\mathcal{Q}$ in 3 rounds, concentrating on a specific subset $\mathcal{H}$ of $attset(\mathcal{Q})$ (set $k = |attset(\mathcal{Q})|$):

- Round 1: Generate the input relations of $\mathcal{Q}^{\#}_{\boldsymbol{\eta}}$ for every configuration $\boldsymbol{\eta}$ of $\mathcal{H}$.
- Round 2: Same as the statistical round in Section 5.
- Round 3: Evaluate $\mathcal{Q}^{\#}_{\boldsymbol{\eta}}$ for every $\boldsymbol{\eta}$.

It remains to elaborate on the details of Round 1 and 3.

Round 1. Allocate $p'_{\boldsymbol{\eta}} = \lceil p \cdot \frac{m_{\boldsymbol{\eta}}}{\Theta(m \cdot \lambda^{k-2})} \rceil$ machines to computing the relations of $\mathcal{Q}^{\#}_{\boldsymbol{\eta}}$. Let us focus on a specific configuration $\boldsymbol{\eta}$. To generate the relations in $\mathcal{Q}^{\#}_{light|\boldsymbol{\eta}}$, we carry out one-attribute reduction (see Section 2.4) for every border attribute $X \in \mathcal{L}$. Specifically, this operation is performed on

- the unary relations $R'_{\{X\}|\boldsymbol{\eta}[e \setminus \{X\}]}$ for all cross edges e of $\mathcal{G}$ incident to X, and
- the binary relations $R'_{e|\emptyset}$ for all light edges e of $\mathcal{G}$ incident to X.

It generates the $R^{\#X}_{e|\boldsymbol{\eta}}$ for every light edge e incident to X. Note that a value $x \in \mathbf{dom}$ is a heavy-hitter for this operation only if x appears in some input relation of $\mathcal{Q}$ $m_{\boldsymbol{\eta}}/p'_{\boldsymbol{\eta}} = \Omega(\frac{m \cdot \lambda^{k-2}}{p}) = \Omega(m/p^{1/\rho})$ times. Therefore, every machine can independently figure out the heavy-hitters from its histogram, and send each tuple in its local storage directly to the corresponding machines where the tuple is needed to perform one-attribute reductions. By Lemma 6, all the one-attribute reductions entail an overall load of $\tilde{O}(p + m_{\boldsymbol{\eta}}/p'_{\boldsymbol{\eta}}) = \tilde{O}(p + m/p^{1/\rho})$.

The relations in $\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}$ can be easily produced by set intersection. Specifically, for every isolated attribute $X \in \mathcal{I}$, we obtain $R''_{\{X\}|\boldsymbol{\eta}}$ as the intersection of all the unary relations $R'_{\{X\}|\boldsymbol{\eta}[e \setminus \{X\}]}$, where e ranges over all cross edges of $\mathcal{G}$ incident to X. This can be done by standard hashing in one round with load $\tilde{O}(m_{\boldsymbol{\eta}}/p'_{\boldsymbol{\eta}}) = \tilde{O}(m/p^{1/\rho})$.

Round 3. Allocating $p''_{\boldsymbol{\eta}}$, as is given in (6), machines to each configuration $\boldsymbol{\eta}$ of $\mathcal{H}$, we compute $\mathcal{Q}^{\#}_{\boldsymbol{\eta}}$ in exactly the same way Lemma 14 computes $\mathcal{Q}''_{\boldsymbol{\eta}}$. In fact, the statement of Lemma 14, as well as the proof, holds verbatim by replacing every $\mathcal{Q}''_{\boldsymbol{\eta}}$ with $\mathcal{Q}^{\#}_{\boldsymbol{\eta}}$ and every $\mathcal{Q}''_{light|\boldsymbol{\eta}}$ with $\mathcal{Q}^{\#}_{light|\boldsymbol{\eta}}$.

We thus have obtained a 3-round algorithm for answering a simple join query with load $\tilde{O}(p^2 + m/p^{1/\rho})$ which is $\tilde{O}(m/p^{1/\rho})$ under our assumption $m \geq p^3$. This establishes the second main result of this paper:

▶ **Theorem 16.** *Given a simple join query with input size m and a fractional edge covering number ρ, we can answer it in the MPC model using p machines in two rounds with load $O(m/p^{1/\rho})$, assuming that $m \geq p^3$, and that each machine has been preloaded with a histogram as is prescribed in Section 5.*

It is worth mentioning that Round 2 of our algorithm (i.e., the statistical round) has a load of $O(p^2)$ such that only the first and third rounds of the algorithm entail a load sensitive to m.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):922–933, 2009.

3 Foto N. Afrati and Jeffrey D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(9):1282–1298, 2011.

4 Albert Atserias, Martin Grohe, and Daniel Marx. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.

5 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *Journal of the ACM (JACM)*, 64(6):40:1–40:58, 2017.

6 Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 137–150, 2004.

7 Xiao Hu, Paraschos Koutris, and Ke Yi. An External-Memory Work-Depth Model and Its Applications to Massively Parallel Join Algorithms. *Manuscript*, 2018.

8 Xiaocheng Hu, Miao Qiao, and Yufei Tao. I/O-efficient join dependency testing, loomis-whitney join, and triangle enumeration. *Journal of Computer and System Sciences (JCSS)*, 82(8):1300–1315, 2016.

9 Bas Ketsman and Dan Suciu. A Worst-Case Optimal Multi-Round Algorithm for Parallel Computation of Conjunctive Queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 417–428, 2017.

10 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-Case Optimal Algorithms for Parallel Query Processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 8:1–8:18, 2016.

11 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case Optimal Join Algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018.

12 Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 224–233, 2014.

13 Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Wiley, New York, 1997.

14 Todd L. Veldhuizen. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

15 Mihalis Yannakakis. Algorithms for Acyclic Database Schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94, 1981.

A Proof of Lemma 6

For each $i \in [1, b]$, divide S_i into (i) S_i^1, which includes the tuples of $\boldsymbol{u} \in S_i$ where $\boldsymbol{u}(X)$ is a heavy-hitter, and (ii) $S_i^2 = S_i \setminus S_i^1$. Accordingly, divide $S_i^\#$ into (i) $S_i^{\#1}$, which includes the tuples of $\boldsymbol{u} \in S_i^\#$ where $\boldsymbol{u}(X)$ is a heavy-hitter, and (ii) $S_i^{\#2} = S_i^\# \setminus S_i^{\#1}$. We will compute $S_i^{\#1}$ and $S_i^{\#2}$, separately.

The computation of $S_1^{\#1}, ..., S_b^{\#1}$ is trivial. Since there are at most p heavy-hitters, each machine storing a heavy-hitter x in some R_j ($j \in [1, a]$) simply broadcasts the pair (x, j) to all machines. This takes one round with load $O(p)$. A machine holding a tuple $\boldsymbol{u}$ with $\boldsymbol{u}(X) = x$ in some S_i ($i \in [1, a]$) adds $\boldsymbol{u}$ to $S_i^{\#1}$ only if it has received (x, j) for all $j \in [1, a]$.

$S_1^{\#2}, ..., S_b^{\#2}$, on the other hand, can be produced using Lemma 5. Assign a share of p to X and a share of 1 to every other attribute. By definition, the join query $\{R_1, ..., R_a, S_1^2, ..., S_b^2\}$ is skew-free, and therefore, can be solved in round round with load $\tilde{O}(n/p)$. For each $i \in [1, b]$, $S_i^{\#2}$ can then be easily obtained from the result of this query.

B Proof of Proposition 7

Let us first introduce a definition. Suppose that $\mathcal{S}$ is a subset of $\mathcal{H}$. We say that a configuration $\boldsymbol{\eta}_\mathcal{H}$ of $\mathcal{H}$ *extends* a configuration $\boldsymbol{\eta}_\mathcal{S}$ of $\mathcal{S}$ if $\boldsymbol{\eta}_\mathcal{S} = \boldsymbol{\eta}_\mathcal{H}[\mathcal{S}]$. $O(\lambda^{|\mathcal{H}|-|\mathcal{S}|})$ configurations $\boldsymbol{\eta}_\mathcal{H}$ of $\mathcal{H}$ can extend $\boldsymbol{\eta}_\mathcal{S}$, because every attribute in $\mathcal{H} \setminus \mathcal{S}$ has $O(\lambda)$ heavy values.

Returning to the proof of the proposition, Let $\boldsymbol{\eta}$ be an arbitrary configuration of $\mathcal{H}$, and $e \in \mathcal{E}$ an arbitrary hyperedge that is active on $\boldsymbol{\eta}$. Define $\mathcal{H}' = e \cap \mathcal{H}$ and $e' = e \setminus \mathcal{H}$. A tuple $\boldsymbol{u} \in R_e$ belongs to $R_{e|\boldsymbol{\eta}[e \setminus e']}$ only if $\boldsymbol{\eta}$ extends the configuration $\boldsymbol{u}[\mathcal{H}']$ of $\mathcal{H}'$. There are $O(\lambda^{|\mathcal{H}|-|\mathcal{H}'|})$ such $\boldsymbol{\eta}$. As $|\mathcal{E}| = O(1)$, $\boldsymbol{u}$ can contribute $O(1)$ to the term $m_{\boldsymbol{\eta}}$ for at most $O(\lambda^{|\mathcal{H}|-|\mathcal{H}'|})$ different $\boldsymbol{\eta}$.

It remains to prove that $|\mathcal{H}| - |\mathcal{H}'| \le k - 2$. Observe that $|\mathcal{H}| - |\mathcal{H}'|$ is the number of attributes in $\mathcal{H}$ that do *not* belong to e. This number is at most $k - 2$ because e has two attributes.

C Proof of Proposition 8

The second equality follows directly from the fact that the scheme of each relation in $\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}$ is disjoint with the scheme of any other relation in $\mathcal{Q}''_{\boldsymbol{\eta}}$. Next we focus on proving the first equality.

We first show $Join(\mathcal{Q}'_{\boldsymbol{\eta}}) \subseteq Join(\mathcal{Q}''_{\boldsymbol{\eta}})$. Consider an arbitrary tuple $\boldsymbol{u} \in Join(\mathcal{Q}'_{\boldsymbol{\eta}})$. For any attribute $X \in \mathcal{L}$ and any cross edge e of $\mathcal{G}$ containing X, since $\boldsymbol{u}(X) \in R_{\{X\}|\boldsymbol{\eta}[e \setminus \{X\}]}$, it must hold that $\boldsymbol{u}(X) \in R''_{\{X\}|\boldsymbol{\eta}}$. For any light edge $e = \{X, Y\} \in \mathcal{E}$, since $\boldsymbol{u}[e] \in R'_{e|\emptyset}$, it must hold that $\boldsymbol{u}[e] \in R''_{e|\boldsymbol{\eta}}$. It thus follows that $\boldsymbol{u} \in Join(\mathcal{Q}''_{\boldsymbol{\eta}})$.

Next, we show $Join(\mathcal{Q}''_{\boldsymbol{\eta}}) \subseteq Join(\mathcal{Q}'_{\boldsymbol{\eta}})$. Consider an arbitrary tuple $\boldsymbol{u} \in Join(\mathcal{Q}''_{\boldsymbol{\eta}})$. For any attribute $X \in \mathcal{L}$, since $\boldsymbol{u}(X) \in R''_{\{X\}|\boldsymbol{\eta}}$, it must hold that $\boldsymbol{u}(X) \in R_{\{X\}|\boldsymbol{\eta}[e \setminus \{X\}]}$ for any cross edge e of $\mathcal{G}$ containing X. For any light edge $e = \{X, Y\} \in \mathcal{E}$, since $\boldsymbol{u}[e] \in R''_{e|\boldsymbol{\eta}}$, it must hold that $\boldsymbol{u}[e] \in R'_{e|\emptyset}$. It thus follows that $\boldsymbol{u} \in Join(\mathcal{Q}'_{\boldsymbol{\eta}})$.

D Proof of Lemma 10

Consider any $R''_{\{X\}|\boldsymbol{\eta}} \in \mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta}}$. Observe that the content of $R''_{\{X\}|\boldsymbol{\eta}}$ does not depend on $\boldsymbol{\eta}(Y)$ for any $Y \in \mathcal{H} \setminus \mathcal{F}$. In other words, if we set $\boldsymbol{\eta'} = \boldsymbol{\eta}[\mathcal{F}]$, then $R''_{\{X\}|\boldsymbol{\eta}}$ is precisely the same as $R''_{\{X\}|\boldsymbol{\eta'}}$. Notice that $\boldsymbol{\eta'}$ is a configuration of $\mathcal{F}$ that is extended by $\boldsymbol{\eta}$ (see the proof of Proposition 7 for the definition of *extension*). The lemma follows from the fact that a configuration $\boldsymbol{\eta'}$ of $\mathcal{F}$ can be extended by $O(\lambda^{|\mathcal{H}|-|\mathcal{F}|})$ configurations $\boldsymbol{\eta}$ of $\mathcal{H}$.

E Proof of Lemma 12

We will prove

$$\bigcup_{\text{config. } \boldsymbol{\eta'} \text{ of } \mathcal{F}} CP(\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta'}}) \times \{\boldsymbol{\eta'}\} \quad \subseteq \quad Join(\mathcal{Q}^*). \tag{7}$$

from which the lemma follows.

Take a tuple $\boldsymbol{u}$ from the left hand side of (7), and set $\boldsymbol{\eta'} = \boldsymbol{u}[\mathcal{F}]$. Based on the definition of $\mathcal{Q}''_{\mathcal{I}|\boldsymbol{\eta'}}$, it is easy to verify that $\boldsymbol{u}[e] \in R_e$ for every cross edge $e \in \mathcal{E}$, and hence, $\boldsymbol{u}[e] \in R^*_e$. Furthermore, $\boldsymbol{u}(X) \in R^*_{\{X\}}$ for every $X \in \mathcal{F}$ because $\boldsymbol{u}(X) = \boldsymbol{\eta'}(X)$ is a heavy value. Finally, obviously $\boldsymbol{u}(Y) \in R^*_{\{Y\}}$ for every $Y \in \mathcal{I}_0$. All these facts together ensure that $\boldsymbol{u} \in Join(\mathcal{Q}^*)$.

F Proof of Proposition 15

Consider first the case where X and Y are both border attributes. We have

$$\begin{aligned} R''_{e|\boldsymbol{\eta}} &= R''_{X|\boldsymbol{\eta}} \bowtie R'_{e|\emptyset} \bowtie R''_{Y|\boldsymbol{\eta}} \\ &= (R''_{X|\boldsymbol{\eta}} \bowtie R'_{e|\emptyset}) \bowtie (R'_{e|\emptyset} \bowtie R''_{Y|\boldsymbol{\eta}}) \\ &= R^{\#X}_{e|\boldsymbol{\eta}} \bowtie R^{\#Y}_{e|\boldsymbol{\eta}}. \end{aligned}$$

If X is a border attribute but Y is not, then:

$$\begin{aligned} R''_{e|\boldsymbol{\eta}} &= R''_{X|\boldsymbol{\eta}} \bowtie R'_{e|\emptyset} \\ &= (R''_{X|\boldsymbol{\eta}} \bowtie R'_{e|\emptyset}) \bowtie R'_{e|\emptyset} \\ &= R^{\#X}_{e|\boldsymbol{\eta}} \bowtie R^{\#Y}_{e|\boldsymbol{\eta}}. \end{aligned}$$

If neither X nor Y is a border attribute, the proposition is trivial.