18th International Conference on Database Theory

ICDT'15, March 23–27, 2015, Brussels, Belgium

Edited by

Marcelo Arenas

Martín Ugarte

LIPIcs – Vol. 31 – ICDT'15 www.dagstuhl.de/lipics

Editors

Marcelo Arenas
Pontificia Universidad Católica de Chile
Santiago, Chile
marenas@ing.puc.cl

Martín Ugarte
Pontificia Universidad Católica de Chile
Santiago, Chile
martinugarte@puc.cl

ACM Classification 1998
H.2: Database Management, H.2.1 Normal forms, H.2.2 Schema and subschema, H.2.3 Query languages, H.2.4 Query processing, H.2.4 Relational databases, H.2.4 Distributed databases, H.2.5 Heterogeneous Databases, H.3.5 Online Information Services, H.1: Miscellaneous – Privacy, H.4.1 Office Automation: Workflow management, B.4.4 Performance Analysis and Design Aids: Formal models, Verification, F.1.3 Complexity measures and classes, F.4.1 Computational Logic, Model Theory, G.2.2 Graph Theory – Hypergraphs

ISBN 978-3-939897-79-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-79-8.

Publication date
March, 2015

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICDT.2015.1

ISBN 978-3-939897-79-8 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

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

http://www.dagstuhl.de/lipics

Contents

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Graph Databases and Semantic Web

Algorithms and Workflows

Distributed Query Processing

Consistency and Repairs

Preface

The 18th International Conference on Database Theory (ICDT 2015) was held in Brussels, Belgium, March 23–27, 2015. Originally biennial, the ICDT conference has been held annually and jointly with EDBT ("Extending Database Technology") since 2009.

The proceedings of ICDT 2015 include a paper by Graham Cormode (University of Warwick) based on the keynote address by him, a paper by Rasmus Pagh (IT University of Copenhagen) based on the keynote address by him, an overview of an invited lecture by Nicole Schweikardt (Humboldt University of Berlin), a laudation concerning the ICDT 2015 Test of Time Award, and 22 research papers that were selected by the Program Committee from 50 submissions.

Out of the 22 accepted papers, the Program Committee selected the paper *A Declarative Framework for Linking Entities* by Douglas Burdick, Ronald Fagin, Phokion G. Kolaitis, Lucian Popa and Wang-Chiew Tan for the ICDT 2015 Best Paper Award. Furthermore, the Program Committee selected the paper *Asymptotic Determinacy of Path Queries using Union-of-Paths Views* by Nadime Francis for the ICDT 2015 Best Student Paper Award. The ICDT 2015 Test of Time Award is given to the paper *Efficient Computation of Frequent and Top-k Elements in Data Streams* by Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi, which originally appeared in the proceedings of ICDT 2005. Warmest congratulations to the authors of these award winning papers!

I thank all authors who submitted papers to ICDT 2015. I would also like to thank all members of the Program Committee, and the external reviewers, for the enormous amount of work they have done. The Program Committee did not meet in person, but carried out extensive discussions during the electronic PC meeting. I thank Andrei Voronkov for his EasyChair system, which made it easy to manage and coordinate the discussion.

I thank the ICDT Council members for their help in selecting the Program Committee and their advice on issues of policy during the conference. Special thanks also go to Martín Ugarte, the Proceedings Chair of EDBT/ICDT 2015. I thank many colleagues involved in the organisation of the conference for fruitful collaboration, in particular, Floris Geerts (EDBT/ICDT 2015 Conference Chair).

Marcelo Arenas
January 2015

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICDT 2015 Test of Time Award

In 2013, the International Conference on Database Theory (ICDT) began awarding the ICDT test-of-time award, with the goal of recognising one paper, or a small number of papers, presented at ICDT a decade earlier that have best met the "test of time". In 2015, the award will recognise a paper from the ICDT 2005 proceedings that has had the most impact in terms of research, methodology, conceptual contribution, or transfer to practise over the past decade. The award will be presented during the EDBT/ICDT 2015 Joint Conference, March 23–27, 2015 in Brussels, Belgium.

The committee consisting of Serge Abiteboul, Sudeepa Roy, and chaired by Leonid Libkin (2005 PC co-chair) has chosen the following recipient of the 2015 ICDT Test of Time Award:

"Efficient Computation of Frequent and Top-k Elements in Data Streams"

by Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi

The paper studies the problem of finding items which occur most frequently in a data stream. This is a basic algorithmic problem of great practical importance. The paper proposed a SpaceSaving algorithm which was shown to both provide theoretical guarantees and to perform significantly better than others in practical scenarios. Since its publication, the paper has made impact in both algorithms research and practical implementations of streaming algorithms. Several implementations of the algorithm have been made available, and they are used both in industry and as benchmarks to compare against other streaming algorithms. The paper has been highly cited: many papers have made use of the data structure, either directly, or to solve new problems. The algorithm itself is easy to motivate and state, and consequently it is taught in a number of algorithms courses.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Conference Chair

Floris Geerts (U. of Antwerp)

Program Chair

Marcelo Arenas (PUC Chile)

Program Committee

Marcelo Arenas (PUC Chile)
Pankaj Agarwal (Duke U.)
Angela Bonifati (U. of Lille 1 & Inria)
Edith Cohen (Microsoft Research)
Giuseppe De Giacomo (Sapienza U. di Roma)
Daniel Deutch (Tel Aviv U.)
Gaelle Fontaine (U. of Chile)
Todd Green (LogicBlox & UC Davis)
Sebastian Maneth (U. of Edinburgh)
Filip Murlak (U. of Warsaw)
S Muthukrishnan (Rutgers U.)
Reinhard Pichler (Vienna U. of Technology)
Christopher Re (Stanford U.)
Cristian Riveros (PUC Chile)
Sudeepa Roy (U. of Washington)
Cristina Sirangelo (LSV, ENS-Cachan)
Yufei Tao (Chinese U. of Hong Kong)
Balder Ten Cate (LogicBlox & UC Santa Cruz)
Jan Van Den Bussche (Hasselt U.)
Stijn Vansummeren (U. Libre de Bruxelles)
Victor Vianu (UC San Diego)
David Woodruff (IBM Almaden)

Proceedings Chair

Martín Ugarte (PUC Chile)

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Foto Afrati
Peter Alvaro
Tom Ameloot
Guillaume Bagan
Pablo Barceló
Manuel Bodirsky
Iovka Boneva
Pierre Bourhis
Paolo Ciaccia
Radu Ciucanu
Claire David
Alin Deutsch
Esther Ezra
Wenfei Fan
George H. L. Fletcher
Olivier Gauwin
Sam Haney
Piotr Hofman
Hossein Jowhari
Phokion G. Kolaitis
Paraschos Koutris
Kasper Larsen
Domenico Lembo
Maurizio Lenzerini
Jerry Li
Leonid Libkin
Katja Losemann
Wim Martens
Dániel Marx
Filip Mazowiecki
Andrew McGregor
Marco Montali
Jelani Nelson
Frank Neven
Jorge Pérez
Andreas Pieris
Chung Keung Poon
Miguel Romero
Riccardo Rosati
Emanuel Sallinger
Vadim Savenkov
Evgeny Sherkhonov
Mantas Simkus
Sebastian Skritek
Wang-Chiew Tan
Srikanta Tirthapura
Sophie Tison
Domagoj Vrgoc
Johannes Wallner
Adam Witkowski
Peter Wood
Zhilin Wu
You Wu
Xiaokui Xiao
Ke Yi
Marc Zeitoun
Thomas Zeume
Wuzhou Zhang

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Tom Ameloot
Pablo Barceló
Leopoldo Bertossi
Sudip Biswas
Iovka Boneva
Douglas Burdick
Balder ten Cate
Hubie Chen
Sara Cohen
Graham Cormode
Victor Dalmau
Ronald Fagin
Gaelle Fontaine
Nadime Francis
Piotr Hofman
Xiaocheng Hu
Samuel Hym
Bas Ketsman
Phokion G. Kolaitis
Egor V. Kostylev
Paraschos Koutris
Adrien Koutsos
Jose Emilio Labra Gayo
Leonid Libkin
Carsten Lutz
Wim Martens
Stefan Mengel
Tova Milo
Frank Neven
Rasmus Pagh
Manish Patil
Lucian Popa
Eric Prud'Hommeaux
Juan L. Reutter
Miguel Romero
Sudeepa Roy
Babak Salimi
Martin Schuster
Nicole Schweikardt
Thomas Schwentick
Frédéric Servais
Rahul Shah
Harold Solbrig
Sławek Staworko
Dan Suciu
Tony Tan
Wang-Chiew Tan
Yufei Tao
Sharma Thankachan
Martín Ugarte
Moshe Y. Vardi
Victor Vianu
Yaacov Y. Weiss
Piotr Wieczorek
Frank Wolter
Yi Yang
Shengyu Zhang
Shuigeng Zhou
Daniel Zinn

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Confounding Problem of Private Data Release

Graham Cormode

University of Warwick
Coventry, UK
G.Cormode@Warwick.ac.uk

Abstract

The demands to make data available are growing ever louder, including open data initiatives and "data monetization". But the problem of doing so without disclosing confidential information is a subtle and difficult one. Is "private data release" an oxymoron? This paper (accompanying an invited talk) aims to delve into the motivations of data release, explore the challenges, and outline some of the current statistical approaches developed in response to this confounding problem.

1998 ACM Subject Classification H.1 [Models and Principles]: Miscellaneous – Privacy

Keywords and phrases privacy, anonymization, data release

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.1

Category Invited Talk

1 Introduction

One can scarcely glance at the Internet these days without being overwhelmed with articles on the great promise of, and excitement surrounding, humanity's ability to collect, store and analyse data. Whether under the banner of "big data", "data science" or some other catchy phrase, data is the new sliced bread, and computer scientists are the new master bakers, in this half-baked metaphor. The list of activities that will benefit from this data-based perspective is lengthy, and has been enumerated enthusiastically in other venues.

The starting point for this article is that for this revolution to succeed, a vital component is the data itself, and in many cases this data derives from the activities of individuals who may be unaware of the manifold uses to which it is being put. Indeed, the cynic can view the era of big data as merely the second coming of data mining, rebranded due to the tarnish that this term carries with it. The original association of data mining, or data dredging as its detractors termed it, was of interrogating data in the pursuit of a purpose for which it was not originally collected.

Putting aside the statistical concerns around torturing the data long enough until it confesses, there are ethical and legal concerns that arise from this use of data. Data protection regulations mandate the ways in which data can and cannot be used, and specifically preclude some reuses of data beyond its original purpose. The individuals who have contributed to the data are naturally concerned about what can be learned about them from this process: either what information is revealed about them directly, or what can be inferred about them from the various metrics and measurements. Laws are increasingly being drawn to protect the data of an individual. The notion of 'privacy' is becoming recognized as a human right[1].

[1] http://www.un.org/ga/search/view_doc.asp?symbol=A/C.3/68/L.45/Rev.1

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This appears to create a tension between the bright future of a data-driven society, and the dark Orwellian nightmare of a world without privacy. Since it was computer scientists and statisticians who were most guilty of creating the hype, they should feel some responsibility for resolving this tension. This focuses attention on a solution that is primarily technical in nature, as opposed to one that is legal, social, or moral.

The solution that is most commonly suggested is seemingly simple: just anonymize the data before it is used. That is, make it impossible to determine the identity of individuals contributing to the data. Then it will be fine to go ahead with any and every subsequent piece of data mangling, since the association between people and their data has been broken.

The annoying flaw in this prescription is that the act of anonymization is far more intricate than one would ever imagine. The awkward history of attempts to anonymize data is littered with anecdotes of failure. These are sufficiently illuminating that they bear retelling, albeit with only light regard for historical accuracy.

I know what you tipped last summer

In 2014, the New York City Taxi and Limousine Commission released a dataset comprising information about taxi trips taken the preceding year[2]. This was in response to a Freedom of Information request. The data had the identifying information of the cabs "masked", so that the same cab had the same masked identifier throughout the data. But with a little effort, the masking function could be reversed, since it was performed by applying a standard hash function (MD5) to the number. A dictionary attack iterating over the moderate ($< 10^7$) number of possibilities was sufficient to reidentify the cabs. From this, various privacy breaches were possible: combining the pick-up time and location with the cab number obtained from press photographs, it was possible to identify where celebrities had traveled and how much they had tipped[3]; and by finding establishments of ill-repute frequented late at night, find trips from these to specific locations to identify the homes of their clients.

Mass Data Release

When Massachusetts Group Insurance Commission released data on hospital visits of state employees, they performed due diligence by removing the names, addresses and social security numbers from the data. However, Latanya Sweeney was able reidentify a large fraction of individuals in the data based on other signals: specifically, date of birth, sex, and postal (ZIP) code. These three attributes – none of them identifying in isolation – turn out to be uniquely identifying for a majority of Americans [9]. Moreover, data linking individuals to their demographics is quite widely available. The result is that the supposedly private detailed medical data for many people could be recovered from the released information.

Many other notable examples follow a similar pattern: the release of Internet search histories by AOL in 2006[4]; the extraction of individual movie viewing histories from Netflix data in 2008[5].

From these horror stories to chill the spines of researchers, certain patterns and themes can be derived:

---

[2] `https://archive.org/details/nycTaxiTripData2013`
[3] `http://research.neustar.biz/2014/09/15/`
[4] `http://www.nytimes.com/2006/08/09/technology/09aol.html`
[5] `http://www.nytimes.com/2009/10/18/business/18stream.html`

- Attempts to release data are usually done with the best of intentions. These can include attempts to create useful data sets for researchers and the public, response to freedom of information requests, and business purposes (attempts to "monetize" rich data sets).
- Those releasing data are not oblivious to the sensitivity of the data they are sharing: they make some efforts to remove or mask identifying data. However, these fail in what in retrospect appear to be obvious ways: the free availability of external information with which to join to reidentify, or trivial attacks on the masking function.
- The consequences vary: in some cases, a large fraction of individuals in the data can be reidentified, in others it is just an unlucky few. The current consensus seems to be that these are equally undesirable outcomes. Similarly, the nature of data does not affect the perceived severity of the breach. Even seemingly innocuous data sets (taxi trips or movie viewings) can inform on people's activities or beliefs that they would consider private.

It is worth noting that there is selection bias in these examples, and that there are many more releases of data which do not expose private information.

In response to these "surprising failures of anonymization" [7], there are a variety of possible responses. One is to despair of the difficulty of private data release, and to resolve to oppose any further releases. However, the various pressures, including the clarion calls from Governments and advocate groups to make data open, mean that data releases will continue to happen and grow as more data becomes available.

Equally pessimistic is to begin with the same premise, and instead to declare that privacy is an artifact of the past, which can no longer be attained in a world of Google and Facebook[6]. However, thus far society seems not to have abandoned its need for privacy.

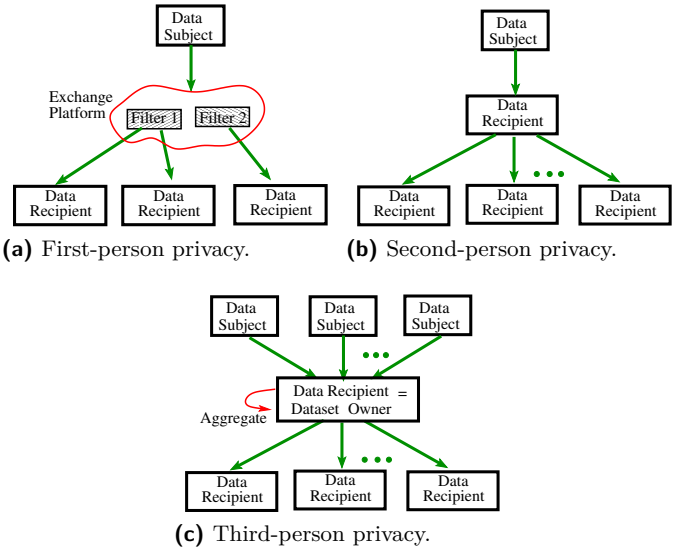
Legal responses are a valid option, but mostly seem to provide some attempt at recompense after the fact rather than prevent, and at best may provide a sufficient penalty that those releasing data do so with more caution and control over its spread. The scepticism with which the computing community has viewed efforts such as the "Right to be forgotten"[7] to put the genie back into the bottle show that information spreads too widely and too quickly for the law to be an effective information removal implement.

Thus, providing tools and mechanisms to understand and assist the release of private data remains the main viable option to respond to these challenges. The computer science and statistical research community has risen to this challenge over the past few decades, by providing a vast literature on the topic. Nevertheless, the problem remains a difficult and confounding one, that will occupy researchers for years to come.

1.1 Outline

The remainder of this article attempts to touch on some of the technical issues surrounding privacy and data release. Section 2 delineates various privacy problems and their connection to other areas. Section 3 outlines some basic principles for working with private data and technical directions. Section 4 identifies some of the most interesting areas for research, and makes some initial suggestions for efforts here.

[6] https://www.eff.org/deeplinks/2009/12/google-ceo-eric-schmidt-dismisses-privacy
[7] http://www.stanfordlawreview.org/online/privacy-paradox/right-to-be-forgotten



(a) First-person privacy. **(b)** Second-person privacy.

(c) Third-person privacy.

Figure 1 Classification of scenarios raising privacy concerns.

2 Privacy Preliminaries

In a typical privacy setting, there are a number of players, with possibly competing interests. There are *data subjects*, whose information is the subject of privacy concern. A data subject can be an individual or user, but it can also be a device, e.g., a computer or cell phone. The subject's information may be released to specific *data recipients*. In some cases, this involves an exchange platform that typically provides both sharing and filtering capabilities. Examples include social networks and photo sharing sites. In other cases, it involves an intermediate entity that either re-shares the data directly (e.g., advertisement-supported sites), or aggregates it first and may then reshare it (e.g., stores, credit card companies, phone companies). The aggregator can end up collating large amounts of data about the demographics, habits, activities and interests of the data subjects, and becomes the *dataset owner* of this collection. Intermediate entities may choose to perform a data release and make some amount of information available to a data recipient, with or without notification to individual data subjects. However, such a release is still governed by legal obligations to data subjects (e.g., opt-in vs. opt-out), as well as good practice (a data release should not harm or upset the data subjects involved). Note that the information disclosed is based on the collected data, but may be modified in a number of ways. A data release can be addressed to a particular recipient, rather than being made generally available to all (a public release).

Within this scenario, there are three broad classes of privacy concerns. The classification is based on how close a data subject is to the final data recipient, and is illustrated in Figure 1. We distinguish between intermediate entities that re-share data at an individual level (Figure 1b) and those that aggregate it first (Figure 1c) because the privacy concerns are significantly different.

2.1 First-person privacy: Users (over)sharing their own data

First-person privacy issues concern the private data that a user shares with other entities (e.g., websites, social networks), and the rules and policies that determine who is able to see

this information. This is illustrated schematically in Figure 1a. There have been a large number of headlines about privacy in recent years which ultimately derive from the difficulty of ordinary users to appreciate the consequences of their data sharing. These include stories about people sacked for inadvertently sharing their feelings about their manager directly with their management chain[8]; and a fugitive captured after revealing his whereabouts to his 'friends' who included a justice department official[9]. Examples like this arise because users imagine that their information is being shared with their "true friends", but in fact it may be shared with all people they have marked as "friends" through a system. When these two groups differ, such unintended consequences can occur.

There are several opportunities for research into ways to help users cope with these privacy problems. Efforts to date have included the provision of tools and warnings to users (from service providers directly or from third-party plugins) about the extent of their (over)sharing, and encouraging them to check before they post information.

2.2 Second-person privacy: information spreads fast

Second-person privacy issues arise when the data that a user has shared with one entity is then passed on to other entities, as shown in Figure 1b. Oversharing of this data can lead to privacy concerns. For example, AT&T researchers identified that when MySpace was connecting to external advertisers to place ads on a page, they were passing detailed private data about the user viewing the page direct to the advertiser[10]. This ultimately led to a binding settlement with the FTC monitoring MySpace's activities for twenty years[11].

There is much research potential here. Focus so far has primarily been on detecting when this happens and tracking the flow of information. A natural approach is to provide languages for users to express their privacy preferences about how and with whom their information can be shared; P3P can be seen as an effort in this direction from the start of the century[12], and "do-not-track" a more recent example on the web. Adoption of such methods have been limited thus far, due to implicit opposition from users (lacking interest in expressing their privacy requirements and difficulty in doing so), and service providers (whose business interests may rely on allowing as much sharing of data as possible). Future directions may be to more actively track information sharing, and the development of "peer-to-peer" data networks where encryption tools are used to control access to information, such as the diaspora social network[13].

2.3 Third-person privacy: private data release

Third-person privacy issues surround the practice of collecting large amounts of data about many individuals together, and sharing this with other entities, illustrated in Figure 1c. This can be in the form of a static data set, a live data feed, or via exposing an API for interactive interrogation. The goal of this activity is to provide general information about a large user base without revealing detailed information about any one individual. However, there is the potential for such data sets to inadvertently reveal private information. The high-profile

[8] http://www.dailymail.co.uk/news/article-1206491/
[9] http://www.guardian.co.uk/technology/2009/oct/14/mexico-fugitive-facebook-arrest
[10] http://online.wsj.com/article/SB10001424052748704513104575256701215465596.html
[11] http://ftc.gov/opa/2012/05/myspace.shtm
[12] http://www.w3.org/P3P/
[13] https://github.com/diaspora/diaspora

failures of private data release outlined in the introduction all fall under the heading of third-person privacy.

2.4 Privacy, Utility and Trust

Guaranteeing the privacy and maintaining the utility of the released data are fundamentally opposing objectives. There are many possible compromise approaches, some favoring privacy over utility, and others the reverse. Where in this continuum one chooses to perform a data release is governed by the level of *trust* in the data recipient. Assessing trust is easier in some cases than others: For first-person privacy, data subjects assess the trustworthiness of their friends, who are the intended data recipients (however, pitfalls exist, as discussed above). For third-person privacy, data owners have various control levers over the data recipients (e.g., if the recipients are analysts employed by the data owner, they are subject to internal rules and regulations; if they are external partners, they are subject to contractual obligations). However, second-person privacy is less amenable to trust analysis, partly because of a lack of transparency in the data release process (see Section 4). Under all three models, understanding the data flow is essential to assessing trust.

2.5 Privacy versus Security

Philosophically, privacy and security (as understood within computer science) have many tenets in common. However, there are essential differences. Security is primarily concerned with a binary decision: is access granted to a particular resource? For example, if a user has the right key they can decrypt a file and access its contents in full, otherwise they learn nothing. In privacy, the issues are more subtle: a data owner has detailed information about a collection of users, and must decide which data items, and in what form, they should be revealed to another entity. As such, the foundations of privacy technology are less mature, and less widely deployed than security technologies such as encryption.

3 Privacy Principles

Some necessary (but not necessarily sufficient) conditions for ensuring private data release include the following:

3.1 Protecting Personally Identifiable Information

When sharing data, it is important to remove information which can uniquely identify the data subject, unless this is absolutely needed. Examples of such "personally identifiable information" (PII) are well-documented, and include names, account numbers, license plate number, and social security numbers. In the communications domain, attributes such as IP address, MAC address, and telephone number are also considered PII.

A US FTC report[14] advocates that data be "de-identified". That is, all PII is removed prior to sharing. However, in some cases, it is necessary to "join" two data sets to collate data on individuals from different sources. If this cannot be done prior to release (e.g., if the two data sets are owned by different organizations), then more complex technical solutions are needed (see below).

[14] `http://ftc.gov/os/2012/03/120326privacyreport.pdf`

3.2 Quasi-identifiers

A major subtlety of data release is that information which does not obviously qualify as PII may nevertheless be sufficient to identify an individual. For example, learning the zip (postal) code of an individual does not typically identify that person[15]: most zip codes contain around 10,000 households. However, if taken in combination with other attributes, this can become identifying, as discussed in the Massachusetts Group Insurance commission example.

There has been much work in the research world on how to anonymize data so that quasi-identifiers are not identifying. The work on k-anonymity tries to delete and reduce precision of information so that each individual matches against at least k entries in the released data [8]. The database community happily occupied itself for many years in generating new variations on k anonymity; the enthusiasm for this has waned since it was observed that k-anonymity/diversity does not necessarily provide very useful protection [5].

A different policy approach is to remove the "obviously" identifying fields (PII), and to ensure that what remains cannot be "reasonably linked" back to a specific individual or device[16]. The FTC's current standard for reasonability is qualitative, and lacking in examples, and so provides little actionable guidance for data release.

3.3 The data minimization principle

A basic concept in data sharing is the data minimization principle: it should reveal no more information than is needed for the task at hand. Clearly, this can guide how much information to delete or mask prior to release. However, putting this into practice can still be challenging. In particular, it is hard to fully anticipate what information could be of use for the data recipient, and it is too easy to fall into the trap of including data "just in case" it is needed. Moreover, the default option is often to allow data to remain in place: it takes an active decision to remove or modify the values. In these cases, it is helpful to remember the value of data to its collator: great effort is often expended in collecting and curating rich data. It is therefore incumbent on organizations to avoid freely giving such wealth to others without good reason.

3.4 Data Correlations

An additional risk to privacy arises from the accretion of data about individuals. That is, if a data set includes a lot of information about someone's activity collected over a long period of time, this can build up into a picture that is unique, and identifying. In the AOL example, it was the large collection of search terms that helped to identify certain users, and hence learn about other searches that they had made. Modern communications and internet applications are a particularly rich source of information about individual's interests and views. Relevant data can include internet browsing history, phone call activities, and set-top box TV data. Even if attempts are made to mask these (e.g., by suppressing or hashing phone numbers), these patterns can quickly become unique for most users; moreover, a determined entity could observe a targeted individual in order to find their entries in the data. As with quasi-identifiers, there may be no reasonable technical provision which can preclude such a determined effort to re-identify while still providing the desired functionality. However, it is possible to ensure that such efforts become costly to enact, and that casual

[15] With some exceptions, e.g., 20252
[16] `http://ftc.gov/os/2012/03/120326privacyreport.pdf`

inspection of the released data does not allow easy identification: a far weaker standard than the goal of perfect privacy, but perhaps a more reasonable one for data that is released to a single party rather than to the world at large.

3.5 Aggregations and Differential Privacy: Safety in Numbers

A natural way to improve the privacy of data is to provide it in aggregated form. That is, instead of reporting the raw data, just provide statistics on groups of the data. For example, instead of releasing full lists of phone calls made on a mobile network, one could compute just the number of calls and average call length (etc.) per account. Or, customer information could be aggregated up to the neighborhood level, rather than at the household level. Great care and thought is still required: information still leaks when some groups are allowed to be small, or when the behavior within a group is uniform.

Within the privacy research community, the concept of "Differential Privacy" is close to such aggregation in spirit [4]. In its most common form, differential privacy typically computes aggregate statistics over grouped data, and adds statistical noise to further perturb the result. In practice, it may be sufficient to rely on aggregation alone, but augmented with suppression of small groups: the uncertainty in which individuals contribute to the data is sufficient to provide the perturbation necessary. Use of aggregation is one technique specifically mentioned in the FTC report (see footnote 14).

3.6 Privacy Checklist

The following checklist is an attempt to articulate a set of questions that should be answered about any planned use of private data:

- What is the data that will be used?
- What are the different fields in the data?
- Which are the uniquely identifying fields?
- How was the data obtained? What control did users have over the inclusion of their data?
- How much data is there? At what rate is it produced?
- Who is the intended recipient of the data?
- What is their intended use for the data?
- What other (linkable) data sets would they have access to?
- What contractual obligations will the data recipient be placed under?
- What transformations will be applied to the data to ensure privacy? Who will perform each step?
- Can each of the data elements be justified or can the list be shortened?
- Will all PII be removed from the data prior to release?
- What partially identifying information will remain in the data? How easy would it be to identify an individual from this?
- How much data will there be on each individual? Will this allow correlation attacks?
- What are the consequences of re-identification? What are the possible harms that could result?
- What are the benefits of the use of the data contrasted against the potential risk of re-identification?
- What are the benefits to the user to share the data?
- Who will ensure that the privacy procedure will be adhered to? Can this procedure be audited?

4 Privacy Pinch-points

The discussion thus far has ranged broadly over different types of data and different data release settings. The subsequent sections consider some specific applications and techniques in more detail. The cited work here is heavily biased to the author's recent research.

4.1 Location and Mobility Data

Data about people's location, gathered from GPS devices and mobile phones, is increasingly available. This gives insight into the distribution of people, but also their movements. The possible applications suggested for mobility data are many and varied: urban planning, dynamic advertising, road traffic analysis, emergency management and more. At the same time, it is understood that the detailed location of an individual is very sensitive: their presence at a particular medical facility, say, may be very private. Even coarse location data is sensitive: an individual may not wish it to be learned that they were far from where they said they would be. Longitudinal location data is also particularly susceptible to correlation attacks of the kind described above: observing someone's location late at night typically identifies a "home" location, while location in the middle of the day identifies a "work" location. This can isolate an individual, and then reveal everywhere else they go.

Consequently, great care is required in releasing location data. Raw trajectories of movements over extended periods reveal too much. Instead, different approaches are needed. These can include: (1) Demographic snapshots. Describe the demographic occupancy of grid-cells of sufficient size, e.g., the (approximate) number of people there; the gender and age breakdowns, etc. [2]. (2) Short trajectories. Describe the detailed movements of (anonymous) individuals for short periods of time. It must be made difficult or impossible to "sew these back together". (3) Density maps. The approximate locations of an identified sub-population can be revealed at regular (e.g., hourly) intervals. Each of these brief outlines needs further research to refine into a specific, robust, procedure.

4.2 Joining private data sets

Much value in working with data comes from the ability to join together multiple data sets, and hence to learn from the combination. For example, a telecoms provider might wish to study the impact of call drops on customers' usage by joining logging data on call drops with billing data. This becomes problematic to achieve under privacy, since such joins are best performed making use of a unique identifier to isolate records corresponding to a particular entity ("the key"); however, such unique identifiers are typically considered PII. Moreover, several attacks on privacy have occurred due to the possibility of joining a private data set with a public one.

There are several natural approaches to dealing with joins over private data: (1) The (trusted) data owner performs the linkage, and then drops the uniquely identifying attributes from the resulting joined data set, before releasing it. (2) Appropriate "hashing" (using a secret 'salt' value[17]) to replace occurrences of the key in both data sets. Then they can be joined using the hashed key value, rather than the true key value. (3) Both data sets can be entrusted to a trusted third party, who will perform the linkage, and return the results to the data user. It is important that the data recipient cannot easily compare the joined output to the input and so reidentify the source of some data items. Each one of these approaches

[17] It was a lack of salt that made the NYC taxi data so easy to reidentify.

necessitates some amount of trust between the parties. There are cryptographic protocols for performing joins without revealing which items matched, but these are considered slow and costly to put into practice.

4.3 Synthetic data sets

One approach that can significantly enhance the privacy of a dataset is to generate a synthetic dataset that mimics certain properties of the original, but contains made-up entries that are generated according to some model [6]. A synthetic dataset is designed such that specific tasks can be performed over it with sufficient accuracy (e.g., analyzing traffic patterns), but will most likely introduce large errors in other, unrelated types of analysis.

Generating synthetic data may seem very different in nature to anonymization of data, as they start from opposite extremes. Data anonymization is often viewed as starting with the original private data in full, and chipping away at it by removal and coarsening of information, whereas synthetic data generation may be seen as starting with nothing, and creating a new data set by sampling from an appropriate statistical model whose parameters are derived from the full data. However, this can also be viewed as a spectrum. One perspective on private data release is that it should be viewed as designing an appropriate model for the data, the parameters of which are learned from the data, and which is rich enough to generate faithful data.

This approach is of particular value when combined with models such as differential privacy. Applying differential privacy to the function $f(x) = x$, i.e. trying to simply release the input data in full, can be seen as a trivially complex model, where the parameters describe the data in full. The effect of differential privacy in this setting is simply to add noise that drowns out all signal in the data. At the other extreme, a simple model of the data, described in terms of sums and averages across all individuals (say) can be obtained very accurately through differential privacy, but may only describe the data poorly. Abstracting from these extremes, the difference between the input data and the released data can be broken into two pieces: the model error (the noise introduced by fitting the data to a model) and the privacy error (the additional noise added to the parameters of the model to provide a privacy guarantee). Much recent work in private data release can then be viewed as trying to find appropriate models for data so that the model error and noise error can both be contained [3].

4.4 Graph Structured Data

One aspect of "big data" is the variety of forms that the data can arrive in. Different types of data require different approaches to allow private data release. An important class of data is that which can be represented in the form of a graph, such as the pattern of interactions between individuals. This provides a suitable target: a problem simple enough to state, yet complex enough to give pause, and flexible enough to model a number of different settings. The reasons that graph data presents a challenge for data release hinges on the fact that typically an individual will correspond to a node in the graph, and the associated information (edges) can be quite substantial. Finding a suitable representation of the graph data so that appropriate statistical noise (say) can be added to mask the presence of an individual while preserving properties of the graph has so far eluded researchers.

4.5 Inference and Privacy

One of the reasons that private data release remains a confounding problem is the difficulty in pinning down a suitable definition. Lacking a precise definition of what properties a private data release should satisfy, it is possible to be fooled into believing that stronger guarantees result. A case in point is the ability to draw strong conclusions about individuals from the released data. One might assume that if data is released under an appropriate privacy model, then it should not be possible to infer supposedly private information about individuals in the data. However, this is often the case, under a variety of privacy models [5, 1].

The reason is that effective classifiers can be built for data where the parameters of the classifier depend not on the behaviour of any one individual, but collectively on large groups within the population. Data released under privacy often preserves statistics on large groups – indeed, this is very much a requirement for utility. Consequently, it is possible to build accurate classifiers for seemingly private information. Applying the classifier to individuals (either from within the data or drawn from a similar population) leads to accurate inferences about them.

4.6 Data-as-a-Service

The concept of data-as-a-service (DaaS) is a powerful one: since companies have access to much data of interest, it should be possible to monetize this, and license access to other organizations who would like to make use of it. Here, privacy concerns come to the fore. It is vital to ensure that detailed customer data is not released as part of DaaS: revealing who a business's customers are, let alone what they are doing, would be viewed as a serious privacy breach. Multiple privacy techniques may be needed to ensure that such transactions can proceed effectively. Specifically, all identifiers should be removed, and it should be ensured that there are no shortcuts that would allow identifiers to be restored. The minimal amount of information should be included, and the contribution of one individual to the data should be limited, to reduce the chances of exploiting data correlations to re-identify a person. Ideally, data would be provided in an aggregate form, possibly with some random noise added and small counts suppressed. Even then, the privacy analysis should take into account the possibility of linking the data to other external sources, and thus establish the potential risks of this.

5 Conclusion

Enabling the release of private data remains a fundamental challenge. Guaranteeing privacy and being able to share useful data stand in fundamental opposition: the only way to provide perfect privacy is to entirely prevent all access to data, and the only way to ensure full use of the data is to make no attempt to address privacy concerns. Nevertheless, there can be workable compromises that provide a reasonable level of privacy against re-identification while enabling legitimate data uses. This article has attempted to outline the different ways in which privacy can be at risk, and discussed principles and ongoing efforts to find workable solutions. Despite the many horror stories and conceptual challenges, there remains optimism that suitable technical solutions can be found to all the promise of big data to be realized while providing strong and effective privacy protections for all.

Acknowledgments. I am indebted to Balachander Krishnamurthy, Magda Procopiuc and Divesh Srivastava for many lengthy and detailed discussions at AT&T Labs–Research

around the topic of privacy. These discussions developed many of the perspectives on privacy promulgated in this paper, and led to notes on which this article is based (perhaps explaining the telecoms bias in some of the examples chosen). I similarly thank my many other collaborators with whom I have worked on topics in privacy over the years. These include Smriti Bhagat, Xi Gong, Xi He, Zach Jorgensen, Ninghui Li, Tiancheng Li, Ashwin Machanavajjhala, Entong Shen, Tanh Tran, Xiaokui Xiao, Ting Yu, and Jun Zhang. I also thank the many other researchers in the privacy community with whom I have discussed ideas and algorithms over the years.

My work is supported by a Royal Society Wolfson Research Merit Award, and European Commission Marie Curie Integration Grant PCIG13-GA-2013-618202.

References

1 Graham Cormode. Personal privacy vs population privacy: Learning to attack anonymization. In *ACM SIGKDD*, August 2011.

2 Graham Cormode, Magda Procopiuc, Divesh Srivastava, and Thanh Tran. Differentially private publication of sparse data. In *International Conference on Database Theory*, 2012.

3 Graham Cormode, Magda Procopiuc, Divesh Srivastava, Xiaokui Xiao, and Jun Zhang. Privbayes: Private data release via bayesian networks. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2014.

4 Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

5 Dan Kifer. Attacks on privacy and deFinetti's theorem. In *ACM SIGMOD International Conference on Management of Data*, 2009.

6 Ashwin Machanavajjhala, Daniel Kifer, John M. Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *IEEE International Conference on Data Engineering*, 2008.

7 Paul Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA Law Review*, 57(6):1701–1778, August 2010.

8 Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-04, SRI, 1998.

9 L. Sweeney. Simple demographics often identify people uniquely. Technical Report Data Privacy Working Paper 3, Carnegie Mellon University, 2000.

Using Locality for Efficient Query Evaluation in Various Computation Models

Nicole Schweikardt

Department of Computer Science
Humboldt-Universität zu Berlin
schweikn@informatik.hu-berlin.de

Abstract

In the database theory and logic literature, different notions of *locality* of queries have been studied, the most prominent being Hanf locality [6, 4, 12] and Gaifman locality [5, 8]. These notions are designed so that, in order to evaluate a local query in a given database, it suffices to look only at small neighbourhoods around tuples of elements that belong to the database.

In this talk I want to give a survey of how to use locality for efficient query evaluation in various computation models. In particular, we will take a closer look at how to enumerate query results with constant delay [2, 9, 3], and at how to evaluate queries in a map-reduce like setting [11] or in Pregel [10]. Also, we will have a closer look at how to transform a given local query into a form suitable for exploiting its locality [1, 7].

1998 ACM Subject Classification H.2.4 [Database Management]: Systems – *Query Processing*, H.2.3 [Database Management]: Languages – *Query Languages*, F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic – *Computational Logic, Model Theory*

Keywords and phrases query evaluation, locality

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.13

Category Invited Talk

References

1 B. Bollig and D. Kuske. An optimal construction of Hanf sentences. *J. Applied Logic*, 10(2):179–186, 2012.

2 A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007.

3 A. Durand, N. Schweikardt, and L. Segoufin. Enumerating answers to first-order queries over databases of low degree. *Proc. PODS 2014*, pages 121–131, 2014.

4 R. Fagin, L. Stockmeyer, and M. Vardi. On Monadic NP vs Monadic co-NP. *Inf. Comput.*, 120(1):78–92, 1995.

5 H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium'81*, pages 105–135. North-Holland, 1982.

6 W. Hanf. Model-theoretic methods in the study of elementary logic. In *The Theory of Models*, J. Addison, L. Henkin, and A. Tarski, Eds. North Holland, 1965, pp. 132–145.

7 L. Heimberg, D. Kuske, and N. Schweikardt. An Optimal Gaifman Normal Form Construction for Structures of Bounded Degree. *Proc. LICS 2013*, pages 63–72, 2013.

8 L. Hella, L. Libkin, and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *J. Symb. Log.*, 64(4):1751–1773, 1999.

9 W. Kazana and L. Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7(2), 2011.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 13–14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

10 G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, G. Czajkowski Pregel: a system for large-scale graph processing. *Proc. SIGMOD 2010*, pages 135–146, 2010.

11 F. Neven, N. Schweikardt, F. Servais, and T. Tan. Distributed Streaming with Finite Memory. *Proc. ICDT 2015*.

12 J. Nurmonen. Counting Modulo Quantifiers on Finite Structures. *Inf. Comput.*, 160(1–2):62–87, 2000.

Large-Scale Similarity Joins With Guarantees

Rasmus Pagh*

IT University of Copenhagen, Denmark
pagh@itu.dk

Abstract

The ability to handle noisy or imprecise data is becoming increasingly important in computing. In the database community the notion of similarity join has been studied extensively, yet existing solutions have offered weak performance guarantees. Either they are based on deterministic filtering techniques that often, but not always, succeed in reducing computational costs, or they are based on randomized techniques that have improved guarantees on computational cost but come with a probability of not returning the correct result. The aim of this paper is to give an overview of randomized techniques for high-dimensional similarity search, and discuss recent advances towards making these techniques more widely applicable by eliminating probability of error and improving the locality of data access.

1998 ACM Subject Classification H.2.4 Database Management – Systems

Keywords and phrases Similarity join, filtering, locality-sensitive hashing, recall

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.15

Category Invited Talk

If I am not me, then who the hell am I?

(Douglas Quaid (Arnold Schwarzenegger) in *Total Recall*)

1 Introduction

In their famous book *Perceptrons* [23] from 1969, Minsky and Papert formulated a simple indexing problem: "Store a set S of D-dimensional binary vectors, such that for a query vector q, and a tolerance r, we can quickly answer if there exists a vector in S that differs from q in at most r positions." To this day the achievable space/time trade-offs for this indexing problem remain unknown, and even the best solutions offer only weak performance guarantees for large r and D compared to the guarantees that are known for exact or 1-dimensional search (hash tables, B-trees). At the same time "tolerance" in data processing and analysis is gaining importance, as systems are increasingly dealing with imprecise data such as: different textual representations of the same object (e.g., in data reconciliation), slightly modified versions of a string (e.g., in plagiarism detection), or feature vectors whose similarity tells us about the affinity of two objects (e.g., in recommender systems).

The reporting version of Minsky and Papert's problem, where the task is to search for all vectors in S within distance r from q, is usually referred to as the *near neighbor* problem, and has been considered as an approach to tolerant indexing for many different spaces and

* Supported by the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 15–24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

metrics. In database settings it will often be the case that there is a *set* Q of vectors[1] for which we want to find all near neighbors in S. If we let $d(q, x)$ denote the distance between vectors q and x, the *similarity join* of Q and S with tolerance r is the set

$$Q \bowtie_r S = \{(q, x) \in Q \times S \mid d(q, x) \leq r\} \ .$$

The term "similarity join" suggests that we join vectors that are similar, defined as having distance smaller than r. Sometimes it is more natural to work with a measure of similarity rather than distance, for example *cosine* or *Jaccard* similarity. This does not fundamentally change the problem since for each similarity measure we can define a distance measure that is a decreasing function of similarity.

Some applications do not specify a fixed search radius r, but rather ask to find the k closest vectors in S for each $q \in Q$. This variant is referred to as a *k-nearest-neighbor* (kNN) similarity join. Here we do not consider kNN similarity joins, but note that there are asymptotically efficient reductions for answering kNN queries using near neighbor queries [16].

While our focus is on theoretical aspects of computing similarity joins, we note that the problem (and its kNN variation) is of broad practical importance and has been a subject of extensive study in recent years in several application-oriented fields:

- databases (e.g. [9, 10, 11, 18, 19, 20, 22, 28, 31, 32, 34]),
- information retrieval (e.g. [6, 12, 35]), and
- knowledge discovery (e.g. [1, 5, 33, 36, 37]).

Scope of this paper. The aim of this paper is to give an overview of the main techniques used for high-dimensional similarity search, and discuss recent advances towards making these techniques more widely applicable by eliminating probability of error and improving the locality of data access.

We focus on *randomized* techniques that scale well to *high-dimensional* data. (In low dimensions there exist efficient indexing methods for similarity search, and these translate into efficient similarity join methods.) For similarity joins it is generally better to take advantage of the fact that many queries have to be answered, and avoid creating a standard index, so we omit discussion of the extensive body of work dealing with *indexes* for similarity search.

We refer to the recent book by Augsten and Böhlen [4] for a survey of a broader spectrum of algorithms for similarity join than we are able to cover here.

2 Techniques for high-dimensional similarity joins

To focus on basic techniques we mainly consider the simple case of D-dimensional binary vectors and Hamming distances. Also, we assume for simplicity that the binary vectors are explicitly stored, but binary vectors with Hamming weight much smaller than D could advantageously be stored using a sparse representation.

2.1 Approximation

Following the seminal papers of Gionis, Har-Peled, Indyk, and Motwani [16, 14] a large literature has focused on *approximate* similarity join algorithms that may have *false negatives*,

[1] While the near neighbor problem can be defined for arbitrary spaces we will refer to elements of the space as "vectors".

i.e., pairs that have distance at most r but fail to be reported by the algorithm. In this setting it is of course desirable to achieve the highest possible *recall*, defined as the fraction of the join result returned by the algorithm.

The possibility of false negatives should not be confused with *over*-approximation where the join result is likely to contain "false positive" pairs at distance up to cr, for some approximation factor c. False positives can be eliminated by checking the distances of all pairs, which is relatively cheap if the *precision* is not too low, i.e., when there is not a large fraction of such false positives. It has been observed that many real-life data sets have low intrinsic dimension in the sense that the number of vectors within distance cr is within a reasonably small factor of the number of vectors within distance r when c is a small constant.

2.2 Candidate set generation

Many approximate similarity join algorithms work by first generating a set of *candidate pairs* $C \subseteq Q \times S$. For each $(q, x) \in C$ the distance $d(q, x)$ is then either computed exactly or estimated with sufficient precision to determine whether or not $d(q, x) \leq r$. More generally one can think about a gradual process where a distance estimate is refined for each pair until it is possible to determine whether $d(q, x) \leq r$. Many results on similarity join can be seen as methods for computing good estimates of distances based on a small amount of information about vectors. Lower bounds from communication complexity (see e.g. [26] and its references) tell us that it is in general not possible to efficiently and accurately estimate distances between two vectors based on a small amount of information about each. But in many settings even crude estimates can give rise to substantial savings compared to a naïve solution that computes all distances precisely. For example, the Hamming distance between vectors q and x always lies between $|||q|| - ||x|||$ and $||q|| + ||x||$, where $||\cdot||$ denotes the Hamming weight, i.e., number of 1s.

Subquadratic algorithms. Ideally, to be able to scale to large set sizes we would like to not spend time on every pair in $Q \times S$. For simplicity of discussion suppose that $|Q| = |S| = n$ such that $|Q \times S| = n^2$. Of course, if the join result has size close to n^2 we cannot hope to use subquadratic time. But for large n it will normally only be of interest to compute $Q \bowtie_r S$ when $|Q \bowtie_r S| \ll n^2$, so we focus on this case.

Some algorithms achieve subquadratic performance for certain data distributions. For example Bayardo et al. [6] exploit that cosine similarity of most pairs of vectors (in information retrieval data sets) is small unless they share high weight features. For data sets where many pairs do not share such high weight features the processing time can be substantially subquadratic. Another example comes from association mining in genetic data where Achlioptas et al. [1], building upon seminal work of Paturi et al. [27], obtained running time empirically proportional to $n^{3/2}$.

Curse of dimensionality. To obtain the widest possible applicability we would ideally like to guarantee subquadratic running time *regardless* of data distribution. It is commonly believed that such a general result is not possible due to the *curse of dimensionality*. Indeed, no similarity join algorithms with good time bounds in terms of parameters n and D are known. This means that, unless the general belief is wrong, we need to settle for results that take some aspect of the data distribution into account. Intuitively, the most difficult case occurs when all distances are close to r, since it is hard to determine whether or not to include a pair without computing its distance exactly. Most data sets have a *distribution of distances* that is much more well-behaved, at least up to a certain radius $r_{\max}$ above which

the similarity join size explodes. Some of the strongest theoretical results that guarantee subquadratic performance (e.g. [16, 14]) rest on the assumption that we are considering r that is not too close to $r_{\max}$. In other words, there is not a huge number of pairs with distance slightly above r. Often the ratio $r_{\max}/r$ is thought of as a parameter c that specifies how well distances need to be approximated to distinguish pairs at distance r from pairs at distance $r_{\max}$.

2.3 Locality-sensitive hashing.

Paturi et al. [27] pioneered an approach to candidate set generation that results in subquadratic time for similarity join[2] as soon as the above-mentioned approximation ratio c is larger than 1.

Their technique can be seems as an early instance of *locality sensitive hashing* (LSH), which creates a candidate set using a sequence of specially constructed hash functions $h_1, h_2, \ldots, h_t$ chosen from a family of functions where the collision probability $\Pr[h_i(q) = h_i(x)]$ is a non-increasing function of the distance $d(q, x)$. For each hash function h_i, every pair in $Q \times S$ with colliding hash values is included as a candidate pair, i.e.:

$$C = \bigcup_{i=1}^{t} \{(q, x) \in Q \times S \mid h_i(q) = h_i(x)\} \ . \tag{1}$$

Computing C can be seen as a sequence of t equi-joins with hash value as join attribute. If we are not concerned with the possibility of generating a candidate pair more than once, these joins can be processed independently, and the space required for each join is linear in the input size. While these joins can of course individually be efficiently computed, the number of joins needed can be large, so it is not obvious that computing them one by one is the best solution. Indeed, in Section 4 we give an overview of recent work highlighting how other approaches can give rise to better temporal locality of memory accesses, resulting in more efficient algorithms in memory hierarchies.

LSH construction. For binary vectors and Hamming distances, the currently best LSH construction (from [27, 14]) is remarkably simple. It works by selecting (or sampling) a random set of bits from the vector, i.e., for some randomly chosen $a_i \in \{0, 1\}^D$ we have:

$$h_i(x) = x \wedge a_i, \tag{2}$$

where $\wedge$ denotes bitwise conjunction, i.e., the `&` operator in many programming languages. The Hamming weight of a_i is chosen to ensure "small" collision probability for the large majority of pairs in $Q \times S$ that have distance above cr, for some c. If the aim is to minimize the total number of vector comparisons plus hash function evaluations the best choice of a_i (for large D) is a random vector with $||a_i|| \approx D \ln(n)/(cr)$. This results in $\Pr[h_i(q) = h_i(x)] < 1/n$ when $d(q, x) > cr$, meaning that for each h_i there will in expectation be $\mathcal{O}(n)$ candidate pairs that have distance above cr. A common way to think about this is as a *filter* that is able to remove all but a few of the up to n^2 pairs in $Q \times S$ that are not "close to being in the join result". If we have $d(q, x) = r$ the same choice of a_i yields $\Pr[h_i(q) = h_i(x)] \approx n^{-1/c}$. So for each hash function h_i the probability that (q, x) is added

[2] In fact, they considered a simplified setting where the task is to find a *single* pair at distance less than r in a binary data set that is otherwise random. But their technique extends to the more general setting of similarity join computation.

to C is quite small. This can addressed by using $t > n^{1/c}$ hash functions, which for an independent sequence of hash functions reduces the probability that $(q, x) \notin C$ (such that the pair becomes a false negative) to $(1 - n^{-1/c})^t < 1/e$. Further increasing the parameter t reduces, but does not eliminate, the probability of false negatives.

We note that LSH families for many other distance (and similarity) measures have been developed. The 2012 Paris Kanellakis Theory And Practice Award went to Andrei Broder, Moses Charikar, and Piotr Indyk for their contributions to work on locality sensitive hashing for fundamental measures, e.g. [14, 7, 8, 13]. LSH methods are by definition randomized and have often been seen as inherently approximate in the sense that they must have a nonzero false negative probability. However, as we will see in Section 3 there are randomized filtering techniques that guarantee *total recall*, i.e., no false negatives. As such, the borderline between LSH techniques and other filtering technique is blurred, or even meaningless.

2.4 Aggregation

In some cases it is possible to gather information on a *set* of vectors that allows filtering of pairs involving any element of the set. One such technique is based on *aggregation*. As an example, Valiant [29] showed that if we consider $Q' \subset Q$ and $S' \subset S$ there are situations in which we can show emptiness of $Q' \bowtie_r S'$ based only on aggregate vectors $\sigma_{Q'} = \sum_{q \in Q'} (2q - 1)s(q)$ and $\sigma_{S'} = \sum_{x \in S'} (2x - 1)s(x)$, where $s : \{0, 1\}^D \to \{-1, +1\}$ is a random function and the arithmetic is over the integers. We consider the dot product

$$\sigma_{Q'} \cdot \sigma_{S'} = \sum_{q \in Q'} \sum_{x \in S'} (s(q)(2q - 1^D)) \cdot (s(x)(2x - 1^D)) \ . \tag{3}$$

The value of (3) is useful for example if the dot product $(2q - 1^D) \cdot (2x - 1^D)$ is close to 0 for all but at most one pair $(q, x) \in Q \times S$. Since $(2q - 1^D) \cdot (2x - 1^D) = D - 2d(q, x)$ a pair with small Hamming distance has a contribution to (3) of either close to $-D$ or close to $+D$. This means that the presence of a close pair can be detected (with a certain probability of error) by considering whether $|\sigma_{Q'} \cdot \sigma_{S'}|$ is close to 0 or close to D. In a theoretical breakthrough [29], Valiant showed how to combine this technique with fast matrix multiplication (for computing all dot products of aggregates) to achieve sub-quadratic running time even in cases where most distances are very close to r. It remains to be seen if aggregation techniques can yield algorithms that are good in both theory and practice. Promising experimental work on a different aggregation technique was carried out by Low and Zheng [21], independently of Valiant, in the context of indexing.

3 Addressing the issue of false negatives

We now consider ways of eliminating false negatives, or in other words achieving total recall. As a starting observation, note that the non-zero probability that q and x do not collide under any h_i, as defined in (2), is *not* due to the fact that each a_i is random. Indeed, if we choose $a_1, \ldots, a_t$ as a random permutation of the set of all vectors of Hamming weight $D - r$ we will be guaranteed that $h_i(q) = h_i(x)$ for some i, while vectors at distance above r will have no collisions. We say that such a sequence of vectors is *covering* for distance r. The reason that traditional LSH constructions are not covering is that the hash functions are chosen *independently*, which inherently means that there will be a probability of error. Though the example above is extreme — it requires t to be very large — it does show that suitably *correlating* the hash functions in the sequence can ensure that all pairs within distance r are included in the candidate set.

More efficient constructions. Following an early work of Greene et al. [15], Arasu et al. [3] presented a much more efficient construction of a covering vector sequence. Their first observation is that if $t = r + 1$ and we choose $a_1, \ldots, a_t$ that are disjoint ($a_i \wedge a_j = 0^D$ for all $i \neq j$) and covering all dimensions ($\bigvee_i a_i = 1^D$) then vectors at distance at most r will be identical in the bits indicated by at least one vector a_i, such that $h_i(q) = h_i(x)$.

While this approach ensures that we generate every pair in $Q \bowtie_r S$ as a candidate, it is not clear how well this sequence of functions is able to filter away pairs at distances larger than r from the candidate set. Norouzi et al. [24] recently considered, independently of Arasu et al. [3], the special case where the binary vectors are random. But for general (worst case) data it could be that all differences between q and x appear in the positions indicated by a_1, in which case there will be hash collisions for all functions except h_1. To address this, Arasu et al. propose to initially apply a random permutation $\pi : \{1, \ldots, D\} \to \{1, \ldots, D\}$ to the dimensions. Abusing notation we let $\pi(x)$ denote the vector where $\pi(x)_i = x_{\pi(i)}$, so that the hash functions considered are defined by:

$$h'_i(x) = \pi(x) \wedge a_i \ . \tag{4}$$

As a practical consideration it may be advantageous to instead compute $\pi^{-1}(h'(x)) = x \wedge \pi^{-1}(a_i)$, since we can precompute the vectors $\pi^{-1}(a_i)$, and this change preserves collisions.

Analysis. We claim that including the permutation makes $\Pr[h'_i(q) = h'_i(x)]$, where the probability is over the choice of random permutation, depend only on the Hamming weight $||a_i||$ and the distance $d(q, x)$. Essentially, $h'_i(x)$ is sampling a set of $||a_i||$ positions from $\{1, \ldots, D\}$, without replacement, and we have a collision if none of these positions differ:

$$\Pr[h'_i(q) = h'_i(x)] = \binom{D - d(q,x)}{||a_i||} \Big/ \binom{D}{||a_i||} \ . \tag{5}$$

When D is large (and $D \gg d(q, x)$) this probability is well approximated by the probability when sampling with replacement, i.e.,

$$\Pr[h'_i(q) = h'_i(x)] \approx (1 - d(q,x)/D)^{||a_i||} \approx \exp(-||a_i||\, d(q,x)/D) \ . \tag{6}$$

As can be seen the collision probability decreases exponentially with $||a_i||$, so increasing the Hamming weights of $a_1, \ldots, a_t$ will increase filtering efficiency. Arasu et al. present a method for getting a covering sequence $a^*_1, \ldots, a^*_t$ of larger Hamming weight (no longer satisfying the disjointness condition). We refer to their paper for a precise description, and simply use $h^*_1, \ldots, h^*_t$ to denote the corresponding hash functions. Arasu et al. do not provide a general analysis, but it appears that the Hamming weight that their method can achieve with t functions is asymptotically (for large t and r) around $D \log_2(t)/(4r)$. This corresponds to a collision probability for each hash function h^*_i of roughly:

$$\Pr[h^*_i(q) = h^*_i(x)] \approx \exp(-\log_2(t) d(q,x)/(4r)) \approx t^{-0.36 d(q,x)/r} \ . \tag{7}$$

The probability that (q, x) is included in the candidate set can be upper bounded by a union bound over all t functions, as $t^{1-0.36\, d(q,x)/r}$. This means that for $d(q, x) > 3r$ we are likely not to include (q, x) as a candidate, and the probability that it becomes a candidate decreases polynomially with t. Just as for standard LSH, the best choice of t balances the cost of false positives in the candidate set with the cost of identifying candidate pairs more than once.

Discussion of optimality. If we consider (q, x) with $d(q, x) = r$ any covering sequence of vectors needs to ensure a collision, which requires, at least, that the expected number of collisions is 1. So if we use hash functions $h_1, \ldots, h_t$ then we must have $\sum_{i=1}^{t} \Pr[h_i(q) = h_i(x) \mid d(q, x) = r] \geq 1$. If we assume that there is a non-increasing function of $f : \mathbf{R} \to \mathbf{R}$ such that for each i, $\Pr[h_i(q) = h_i(x) \mid d(q, x) = cr] = f(c)$, then:

$$t \geq 1/f(1) \ . \tag{8}$$

According to (7) the construction of Arasu et al. [3] obtains $f(1) \approx t^{-0.36}$, i.e., $t \approx (1/f(1))^{2.77}$. If it is possible to obtain a sequence of hash functions of length close to the lower bound (8) it will be possible to match the performance of the classical LSH method of Gionis et al. [14]. A nonconstructive argument by the probabilistic method, using a sequence of randomly chosen vectors a_i, shows that $t = \mathcal{O}(r \log(D)/f(1))$ suffices to ensure the existence of a covering sequence of vectors. However, this fact is of little help since we have no effective way of constructing such a sequence, or even checking its correctness.

4 Addressing the issue of data locality

An issue with the candidate generation method that we have outlined is that it makes poor use of the memory hierarchy. If the data set does not fit in internal memory it is likely that we will need to read every vector from external memory once for each hash function. In contrast, methods that explicitly consider all pairs of vectors can be implemented with good data locality by handling subsets of Q and S that fit in fast memory.

Bahmani et al. [5] suggested to use two levels of LSH to compute similarity joins more efficiently in a distributed setting, where the goal is to distribute the data such that most computations can be done separately on each node. Roughly, the top-level LSH distributes the data to nodes such that it is reasonably evenly distributed, and the second-level LSH improves the filtering to decrease the computational cost.

In a recent paper [25] we take this idea further and analyze the resulting algorithm in the *I/O model* (see e.g. [30]), where the aim is to minimize the number of block transfers between slow and fast memory. Consider a setting where standard LSH-based similarity joins use n^ρ hash functions, for a parameter $\rho \in (0; 1)$. If n significantly exceeds the number M of vectors that fit in fast memory, this means that the join algorithm will need to transfer each element n^ρ times between fast and slow memory. This is not always better than a naïve block nested loop join algorithm, which transfers each vector n/M times. Our main finding is that both of these bounds can be improved for large n, to $\mathcal{O}((n/M)^\rho \operatorname{poly} \log n)$ transfers of each vector, under a reasonable assumption on the distance distribution (essentially that the number of pairs in $Q \times S$ at distance close to r is not huge).

A recursive similarity join algorithm. The algorithm described in our paper [25] is designed with ease of analysis in mind, so it may not be fully practical — in particular due to the $\operatorname{poly} \log n$ factor. Here we will consider a simple variant that highlights the main idea of [25] and is potentially useful. The idea is to use a shorter sequence of LSH functions $h_1, \ldots, h_L$ that is just powerful enough to filter away a *constant fraction* of the candidate pairs in $Q \times S$. This step alone will leave many false positives, but it can be applied *recursively* to each bucket of each hash function to increase filtering efficiency. We will soon describe the base case of the recursion, but first state the general case, where $C(Q, S)$ denotes our recursive

function for computing a candidate set. Let V denote the set of possible hash values; then:

$$C(Q,S) = \bigcup_{i=1}^{L} \bigcup_{v \in V} C(\{q \in Q \mid h_i(q) = v\}, \{x \in S \mid h_i(x) = v\}), \tag{9}$$

where the sequence of hash functions is chosen randomly and independently. Since a pair may collide more than once the candidate set should be thought of as a *multiset*, and the number of candidates proportional to the size of this multiset. The recursion in (9) effectively replaces the naïve candidate set $Q \times S$ by a multiset of colliding pairs of size:

$$\text{coll}(Q,S) = \sum_{i=1}^{L} \sum_{v \in V} |\{q \in Q \mid h_i(q) = v\} \times \{x \in S \mid h_i(x) = v\}| \ . \tag{10}$$

We would like to apply (9) when it at least halves the number $\text{coll}(Q,S)$ of candidate pairs compared to $|Q \times S|$. That is, as a base case we take:

$$C(Q,S) = Q \times S \text{ if } |Q \times S| \leq 2\,\text{coll}(Q,S) \ . \tag{11}$$

Discussion. If we consider the sets $Q' \subseteq Q$ and $S' \subseteq S$ in a particular base case, all vectors will have identical values for a sequence of hash functions. This is in some ways similar to the common technique of composing several LSH functions to increase filtering efficiency. The essential difference is that many subproblems can share each hash function, so much of the work on distributing vectors into subproblems is shared. Since applying the LSH family to the base case fails to decrease the number of candidate pairs substantially, it must be the case that the majority of pairs in $Q' \times S'$ have distance not much above r, so the complexity of checking candidates can be bounded in terms of the number of pairs that have distance not much above r.

We outline the main idea in the analysis from our paper [25] as it applies to the simplified algorithm. If we restrict attention to a particular pair $(q,x) \in Q \times S$, the recursion can be seen as a *branching process* where the presence of (q,x) in a subproblem may lead to the pair being repeated in one or more recursive subproblems. The pair is ultimately considered if and only if it "survives" on some path in the recursion tree, all the way down to a base case (where all pairs are considered as candidates). We would like to ensure that if $d(q,x) \leq r$ then (q,x) survives with certainty (or with good probability, if false negatives can be tolerated), while if $d(q,x) > cr$ for $c > 1$ the probability that (q,x) becomes a candidate decreases rapidly with c. The theory of branching processes (see e.g. [17]) says that this will be the case if the expected number of recursive calls involving (q,x) is at least 1 for $d(q,x) \leq r$ and less than 1 for $d(q,x) > cr$.

An advantage of a recursive algorithm is that once subproblems are small enough to fit in fast memory they will produce no further transfers between fast and slow memory. When solving the base case we may also use a recursive approach by splitting $Q \times S$ into four candidate sets each involving half of Q and S.

We stress that an arbitrary sequence of hash functions can be used, including those described in Section 3. In this case, the similarity join algorithm will achieve total recall.

5 Conclusion and open problems

We have argued that randomized techniques related to those traditionally used in LSH are applicable to similarity joins in which false negatives do not occur. It remains to be seen to

what extent such techniques can be extended to other spaces and metrics, such as Euclidean or Manhattan distances, but non-constructive arguments suggest that at least some nontrivial results are possible. Even in the case of binary vectors it is an interesting question if the construction of Arasu et al. [3] can be improved, and in that case how close to the lower bound (8) it is possible to get.

Finally, recent developments on indexes for approximate similarity search [2] suggest that it may be possible to further improve LSH-based similarity join algorithms.

Acknowledgement. The author thanks Ninh Pham, Francesco Silvestri, and Matthew Skala for useful feedback on a preliminary version of this paper.

References

1 Panagiotis Achlioptas, Bernhard Schölkopf, and Karsten Borgwardt. Two-locus association mapping in subquadratic time. In *Proceedings of KDD*, pages 726–734. ACM, 2011.

2 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. *arXiv preprint arXiv:1501.01062*, 2015.

3 Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *Proceedings of VLDB*, pages 918–929, 2006.

4 Nikolaus Augsten and Michael H Böhlen. Similarity joins in relational database systems. *Synthesis Lectures on Data Management*, 5(5):1–124, 2013.

5 Bahman Bahmani, Ashish Goel, and Rajendra Shinde. Efficient distributed locality sensitive hashing. In *Proceedings of CIKM*, pages 2174–2178, 2012.

6 Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proceedings of WWW*, pages 131–140, 2007.

7 Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

8 Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of STOC*, pages 380–388, 2002.

9 Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of ICDE*, page 5, 2006.

10 Yun Chen and Jignesh M Patel. Efficient evaluation of all-nearest-neighbor queries. In *Proceedings of ICDE*, pages 1056–1065. IEEE, 2007.

11 Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. *IEEE Trans. Knowl. Data Eng.*, 13(1):64–78, 2001.

12 Abhinandan Das, Mayur Datar, Ashutosh Garg, and ShyamSundar Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of WWW*, pages 271–280, 2007.

13 Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of SOCG*, pages 253–262, 2004.

14 Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of VLDB*, pages 518–529, 1999.

15 Dan Greene, Michal Parnas, and Frances Yao. Multi-index hashing for information retrieval. In *Proceedings of FOCS*, pages 722–731. IEEE, 1994.

16 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

17 Theodore E Harris. *The theory of branching processes*. Courier Dover Publications, 2002.

18 Edwin H Jacox and Hanan Samet. Metric space similarity joins. *ACM Transactions on Database Systems (TODS)*, 33(2):7, 2008.

19 Yu Jiang, Dong Deng, Jiannan Wang, Guoliang Li, and Jianhua Feng. Efficient parallel partition-based algorithms for similarity search and join with edit distance constraints. In *Proceedings of Joint EDBT/ICDT Workshops*, pages 341–348. ACM, 2013.

20 Guoliang Li, Dong Deng, Jiannan Wang, and Jianhua Feng. Pass-join: A partition-based method for similarity joins. *Proceedings of the VLDB Endowment*, 5(3):253–264, 2011.

21 Yucheng Low and Alice X Zheng. Fast top-k similarity queries via matrix compression. In *Proceedings of CIKM*, pages 2070–2074. ACM, 2012.

22 Jiaheng Lu, Chunbin Lin, Wei Wang, Chen Li, and Haiyong Wang. String similarity measures and joins with synonyms. In *Proceedings of SIGMOD*, pages 373–384. ACM, 2013.

23 Marvin L Minsky and Seymour A Papert. *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press, 1987.

24 Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast search in hamming space with multi-index hashing. In *Proceedings of CVPR*, pages 3108–3115. IEEE, 2012.

25 Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. I/O-efficient similarity join in high dimensions. Manuscript, 2015.

26 Rasmus Pagh, Morten Stöckel, and David P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *Proceedings of PODS*, pages 109–120. ACM, 2014.

27 Ramamohan Paturi, Sanguthevar Rajasekaran, and John Reif. The Light Bulb Problem. *Information and Computation*, 117(2):187–192, March 1995.

28 Yasin N Silva, Walid G Aref, and Mohamed H Ali. The similarity join database operator. In *Proceedings of ICDE*, pages 892–903. IEEE, 2010.

29 Gregory Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas. In *Proceedings of FOCS*, pages 11–20. IEEE, October 2012.

30 Jeffrey Scott Vitter. *Algorithms and Data Structures for External Memory*. Now Publishers Inc., 2008.

31 Jiannan Wang, Guoliang Li, and Jianhua Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *Proceedings of ICDE*, pages 458–469. IEEE, 2011.

32 Jiannan Wang, Guoliang Li, and Jianhua Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *Proceedings of SIGMOD*, pages 85–96. ACM, 2012.

33 Ye Wang, Ahmed Metwally, and Srinivasan Parthasarathy. Scalable all-pairs similarity search in metric spaces. In *Proceedings of KDD*, pages 829–837, 2013.

34 Chenyi Xia, Hongjun Lu, Beng Chin Ooi, and Jing Hu. Gorder: an efficient method for knn join processing. In *Proceedings of VLDB*, pages 756–767. VLDB Endowment, 2004.

35 Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of WWW*, pages 131–140, 2008.

36 Reza Bosagh Zadeh and Ashish Goel. Dimension independent similarity computation. *The Journal of Machine Learning Research*, 14(1):1605–1626, 2013.

37 Xiang Zhang, Fei Zou, and Wei Wang. Fastanova: an efficient algorithm for genome-wide association study. In *Proceedings of KDD*, pages 821–829. ACM, 2008.

A Declarative Framework for Linking Entities

Doug Burdick[1], Ronald Fagin[1], Phokion G. Kolaitis[2,1], Lucian Popa[1], and Wang-Chiew Tan[2]

1 **IBM Research – Almaden**
2 **UC Santa Cruz**

Abstract

The aim of this paper is to introduce and develop a truly declarative framework for entity linking and, in particular, for entity resolution. As in some earlier approaches, our framework is based on the systematic use of constraints. However, the constraints we adopt are link-to-source constraints, unlike in earlier approaches where source-to-link constraints were used to dictate how to generate links. Our approach makes it possible to focus entirely on the intended properties of the outcome of entity linking, thus separating the constraints from any procedure of how to achieve that outcome. The core language consists of link-to-source constraints that specify the desired properties of a link relation in terms of source relations and built-in predicates such as similarity measures. A key feature of the link-to-source constraints is that they employ disjunction, which enables the declarative listing of all the reasons as to why two entities should be linked. We also consider extensions of the core language that capture collective entity resolution, by allowing inter-dependence between links.

We identify a class of "good" solutions for entity linking specifications, which we call *maximum-value solutions* and which capture the strength of a link by counting the reasons that justify it. We study natural algorithmic problems associated with these solutions, including the problem of enumerating the "good" solutions, and the problem of finding the certain links, which are the links that appear in every "good" solution. We show that these problems are tractable for the core language, but may become intractable once we allow inter-dependence between link relations. We also make some surprising connections between our declarative framework, which is deterministic, and probabilistic approaches such as ones based on Markov Logic Networks.

1998 ACM Subject Classification H.2.5 Heterogeneous Databases

Keywords and phrases entity linking, entity resolution, constraints, certain links

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.25

1 Introduction

Entity linking is a long-standing research problem that has received considerable attention over the years. The most extensively investigated case of entity linking is entity resolution, which is the problem of linking pieces of information occurring in one or more, possibly heterogeneous, datasets that refer to the same real-world object (entity). Entity resolution is known under various names: record linkage, data deduplication, reference reconciliation, merge-purge (see, e.g., [9, 11, 15, 22, 26]). Much of entity resolution research has focused on developing the algorithms, similarity measures, and the general methodologies for matching entities, while at the same time significant engineering effort has been devoted to experimenting and tuning the resulting systems.

In recent years, we have seen several new efforts aimed at raising the level of abstraction in entity resolution systems. These efforts, ranging from the earlier AJAX framework [17] to the more recent Dedupalog [2] and HIL [21] languages, represent attempts to specify, in

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 25–43

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a more declarative way, the basic ingredients of an entity resolution process. In particular, instead of using lower-level implementation algorithms, they employ SQL-like constructs or constraints expressed in logical formalisms as components of a high-level language. A common characteristic in these approaches is the use of *source-to-link* constraints, that is, constraints that specify the direct creation of the links from the source data. In turn, this feature has the consequence that operational semantics are used, hence the meaning of a specification in such a language is some link relation resulting from the operational semantics. For example, HIL uses SQL-like statements to express the creation of links from a set of sources and provides only an operational procedure to interpret such statements. As for Dedupalog, the specification has the form of a Datalog-style program with constraints of two types: hard constraints and soft constraints. The goal is to find a link relation that satisfies the hard constraints, and that minimizes the number of soft constraints violated. Since this turns out to be a computationally intractable problem, the semantics of Dedupalog is given by an algorithm that, in many cases, produces an approximately optimal result.

In this paper, we take a different approach to declarative entity linking (and, in particular, to declarative entity resolution), where we clearly separate the specification from the implementation, and also ensure that the implementation always satisfies the specification. Our goal is to provide a clean and expressive specification language, with rigorous semantics, which can serve as a foundation for the implementation or evaluation of entity linking systems. Two salient features of our framework are as follows.

First, we consider entity resolution as a general problem of defining links between source values. A (binary) link is modeled as a binary table that relates pairs of values from the given source relations. While, as described earlier, entity resolution is typically confined to the problem of matching entities representing the same real-world object, our framework allows linking entities that are not necessarily of the same type; in particular, a link relation need not be an equivalence relation. In other words, the same type of specification is used not only to match person records across multiple databases (which is a typical entity resolution application), but also to link a subsidiary company with its parent company or to link a CEO with his/her company.

Second, as in some of the earlier approaches, our specification language is based on constraints. However, the constraints we adopt are *link-to-source* constraints, unlike in earlier approaches where source-to-link constraints were used to dictate how to populate the link relations. Our approach makes it possible to focus entirely on the intended properties of the outcome of entity linking, thus separating the constraints from any procedure of how to achieve that outcome. The core language $\mathcal{L}_\mathbf{0}$ consists of link-to-source constraints that specify the desired properties of link relations in terms of the source relations and built-in predicates, such as similarity measures. We also consider extensions of $\mathcal{L}_\mathbf{0}$ in which other link relations may be used in the specification of link relations, thus allowing a link to also depend on other links. We distinguish two such extensions, namely, the language $\mathcal{L}_\mathbf{1}$ in which no recursion is allowed in the specification (i.e., no link relation depends on itself) and the language $\mathcal{L}_\mathbf{2}$ in which recursion is allowed; these extensions capture what is usually called *collective entity resolution* [6], where inter-dependence between links is allowed. A key feature of the link-to-source constraints is that, in their most general form, they are *disjunctive constraints* that enable the declarative listing of all the reasons as to why two entities are linked. In addition, our specification languages make use of inclusion dependencies that specify the provenance of the links w.r.t. the source data, and also allow for functional dependencies that specify when a link relation is many-to-one or one-to-one.

Since all our constraints are link-to-source, they always admit *solutions*, that is, link

relations that satisfy all the constraints at hand. (The empty link instance is always a solution, albeit not necessarily a desirable one.) Therefore, one of the main questions that has to be addressed is: what are the "good" solutions out of the space of all possible solutions? Moreover, since multiple good solutions may exist for a given specification and a given source instance, a related important problem is that of identifying the *certain links* and the *ambiguous links*, that is, those links that appear in every good solution and, respectively, in some, but not in every, good solution. From a practical point of view, the certain links are the links that should be kept, while the ambiguous links are the links that must be inspected by a human. In particular, examination of the ambiguous links may lead to a revised specification that will result into fewer ambiguous links. Thus, producing the ambiguous links is an important computational task.

As a first candidate of a class of "good" solutions, we consider *maximal solutions*, where goodness is maximality among solutions w.r.t. set containment. For each fixed entity linking specification in the core language $\mathcal{L}_\mathbf{0}$, we show that there is a polynomial-delay algorithm that, for each source instance I, enumerates all of the maximal solutions for I. (A *polynomial-delay algorithm* [24] for a problem is an algorithm that, given an input, generates all solutions to the problem, one after another, where the first solution is generated in polynomial time, and the next solution is generated in polynomial time after the previous solution.) We also show that there are polynomial-time algorithms that, given a source instance I, compute the certain links and the ambiguous links for I w.r.t. the class of maximal solutions. However, we point out that, in practice, there are too many maximal solutions, which implies that quite often there are too few certain links, if any. In other words, the semantics given by maximal solutions is too coarse-grained and does not have enough differentiating power among solutions. In view of the above, we refine the semantics by considering a subclass of maximal solutions that we call *maximum-value solutions*, which maximize the total strength of links. Under this semantics, the strength of a link is measured by counting the disjuncts and existential witnesses that "justify" the existence of a link. For each fixed entity linking specification in the core language $\mathcal{L}_\mathbf{0}$, we show that there is a polynomial-delay algorithm that, for each source instance I, enumerates all of the maximum-value solutions for I. We also show that there are polynomial-time algorithms that, for each source instance I, compute the certain links and the ambiguous links for I w.r.t. the class of maximum-value solutions.

We also establish that some existing probabilistic approaches for entity resolution can be captured, in a precise sense, by entity linking specifications in $\mathcal{L}_\mathbf{0}$ under a suitable extension of the maximum-value semantics that allows for weight functions. We start with a well-known class of probabilistic matching algorithms that originated with Fellegi and Sunter [15] and is at the core of many commercial systems, including IBM's QualityStage [23]. We show that the core logic behind these matching algorithms is captured by a fragment of $\mathcal{L}_\mathbf{0}$ where each disjunct in the matching constraint consists of a single atomic formula. We then consider the richer probabilistic framework of Markov Logic Networks (MLNs) [29], which in general allows for arbitrary first-order formulas to be interpreted in a probabilistic sense. We show that a class of *linear MLNs* that is useful for entity resolution [30] is captured by a fragment of $\mathcal{L}_\mathbf{0}$ (under the same extended semantics). Thus, rather surprisingly, a purely probabilistic approach (based on MLNs) can be captured in a deterministic way. As a byproduct, we show that for linear MLNs, there is a polynomial-delay algorithm for enumerating the maximum probability worlds, and polynomial-time algorithms for computing the certain and ambiguous links (w.r.t. the class of maximum probability worlds). To the best of our knowledge, these are the first polynomial-time results for MLN-based entity resolution.

The state of affairs turns out to be dramatically different for the extended language $\mathcal{L}_\mathbf{1}$

that allows dependence of a link on other links, but disallows recursive interdependence between links. To begin with, we show that there is a fixed entity linking specification in $\mathcal{L}_1$ for which the following problem is NP-complete: given a source instance I and a positive integer k, is there a solution for I whose value is at least k? Consequently, there is no polynomial-delay algorithm for enumerating the maximum-value solutions, unless P = NP. Moreover, we show that there is a fixed entity linking specification in $\mathcal{L}_1$ for which there are no polynomial-time algorithms for telling whether a link is certain or ambiguous w.r.t. the class of maximum-value solutions, unless NP = coNP. It should be noted that the intractability of recognizing the certain links and the ambiguous links is established by using results about the computational complexity of recognizing *frozen variables* in constraint satisfaction problems [25]. On the positive side, we identify a large syntactic fragment of $\mathcal{L}_1$ for which there is a polynomial-delay algorithm for enumerating maximum-value solutions, as well as polynomial-time algorithms for computing the certain links and the ambiguous links.

The complete proofs of all our results will appear in a full version of this paper.

2 A Declarative Framework for Linking Entities: Basics

A source relational schema is a finite sequence $\mathbf{S} = \langle R_1, \ldots, R_m \rangle$ of relation symbols, each of a fixed arity. When attribute names are not essential, we may identify attributes by their position. A *source instance* I over $\mathbf{S}$ is a sequence $(R_1^I, \ldots, R_m^I)$, where each R_i^I is a finite relation of the same arity as R_i. We often use R_i to denote both the relation symbol and the relation R_i^I that interprets it. Additionally, a *link schema* is a finite sequence $\mathbf{L} = \langle L_1, \ldots, L_n \rangle$ of link symbols, where each L_i is binary. A *link instance* J over $\mathbf{L}$ is a sequence $(L_1^J, \ldots, L_n^J)$ of finite binary relations. For a relation T (either source or link) and a tuple t in T, we denote by $T(t)$ the association between T and t and refer to it as a *fact*. When T is a link relation, we may refer to $T(t)$ as a *link*. An instance can be conveniently represented by its set of facts. Given instances K and K', we say that K is a subinstance of K' and write $K \subseteq K'$ if the set of facts in K is a subset of the set of facts in K'. We write $K \subset K'$ if this subset relationship is strict.

We specify a link relation, implicitly, by defining the properties that it must satisfy. For each link symbol L, there are three sets of constraints, as follows. The first set contains at most one *matching constraint* of the form

$$(m_L) \quad L(x, y) \rightarrow \forall \mathbf{u}(\psi(x, y, \mathbf{u}) \rightarrow \alpha_1 \vee \ldots \vee \alpha_k),$$

where $\psi(x, y, \mathbf{u})$ is a (possibly empty) conjunction of atomic formulas over $\mathbf{S}$, the universally quantified variables $\mathbf{u}$ must occur in ψ, and where $\alpha_i ::= \exists \mathbf{z}\ \phi_i(x, y, \mathbf{u}, \mathbf{z})$. Each ϕ_i is a conjunction of source atomic formulas, along with equalities and other built-in or user-defined boolean predicates (such as similarity or string containment). Also, note that x and y are universally quantified, but for simplicity of notation we omit their quantifiers.

The intuition behind the use of disjunction in the matching constraint is that it lists all the possible matching conditions (i.e., α_1, ..., α_k) for why a link $L(x, y)$ may exist. If a link $L(x, y)$ exists, then one or more of those reasons must be true. We do not require a matching constraint to be given for each link; for those links without a matching constraint, the link relation is implicitly defined by the rest of the constraints. We will give concrete examples of matching constraints shortly. We will also explain the role of the universal quantification $\forall \mathbf{u}$ and of the formula $\psi(x, y, \mathbf{u})$.

The second set of constraints, for a given link symbol L, consists of an inclusion dependency of the form $L[X] \subseteq R[A]$ and an inclusion dependency of the form $L[Y] \subseteq R'[A']$. Here,

X and Y denote the first and the second attribute of L, while A and A' denote attributes in source relations R and R'. As usual, $R[A]$ denotes the projection of R on A. The two dependencies specify an upper bound for the set of links that can appear in L: every link in L will be a pair relating a value in $R[A]$ with a value in $R'[A']$. Finally, the third set of constraints, for a given link symbol L, with attributes X and Y, consists of zero, one or both of the functional dependencies $L : X \to Y$ and $L : Y \to X$. Functional dependencies encode basic cardinality constraints on the result of entity linking. (See also [21] for the significance of such constraints in practice.) The presence of one functional dependency means that the links are required to be many-to-one, that is, an entity on one side must be linked with at most one entity on the other side. The presence of both functional dependencies means that the links must be one-to-one.

We use the term $\mathcal{L}_\mathbf{0}$ for the above language, consisting of matching constraints, and inclusion and functional dependencies. Later, we also consider extensions to $\mathcal{L}_\mathbf{0}$ (such as allowing for inter-dependencies among the links). We use the term *entity linking specification* (in $\mathcal{L}_\mathbf{0}$) for a triple $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ where $\mathbf{L}$ is a link schema, $\mathbf{S}$ is a source schema, and Σ is a set of constraints containing, for each link symbol L in $\mathbf{L}$, (1) at most one matching constraint, (2) two inclusion dependencies, and (3) zero, one or two functional dependencies, all as defined above.

▶ **Definition 1.** Let $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ be an entity linking specification, and I be a source instance. A *solution* for I w.r.t. $\mathcal{E}$ is a link instance J such that $(J, I) \models \Sigma$, where (J, I) is the instance over the schema $\mathbf{L} \cup \mathbf{S}$ obtained by taking the union of the facts in J and I.

▶ **Example 2.** In this scenario, we link subsidiaries in one database with parent companies in another database. Consider the following source schema $\mathbf{S}$:

$$\texttt{Subsid}(sid, sname, location) \qquad \texttt{Company}(cid, cname, hdqrt) \qquad \texttt{Exec}(eid, cid, name, title)$$

This source schema includes the relation symbols `Subsid` from the first database, and `Company` and `Exec` from the second database. A source instance I for $\mathbf{S}$ is given below as a set of facts:

$\texttt{Subsid}(s_1, \text{“Citibank N.A.”}, \text{“New York”})$
$\texttt{Subsid}(s_2, \text{“CIT Bank”}, \text{“Salt Lake City”})$
$\texttt{Company}(c_1, \text{“Citigroup Inc”}, \text{“New York”})$
$\texttt{Company}(c_2, \text{“CIT Group Inc”}, \text{“New York”})$
$\texttt{Exec}(e_1, c_1, \text{“E. McQuade”}, \text{“CEO, Citibank N.A.”})$

The intention is to generate links between subsidiary ids and corresponding company ids. Thus, the link schema $\mathbf{L}$ consists of a single link symbol $\texttt{L}(sid, cid)$. In the scenario,“Citibank N.A.” is the name of a true subsidiary of “Citigroup Inc”, while “CIT Bank” is the name of a true subsidiary of “CIT Group Inc”. We note that this is a real-life example, and “Citigroup Inc” and “CIT Group Inc” are two different financial institutions. The goal of entity linking is to identify links such as $L(s_1, c_1)$ and $L(s_2, c_2)$, given the available evidence.

A possible entity linking specification that exploits the available attributes and the relationship between the source tables is $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$, where Σ consists of a matching constraint:

$$\texttt{L}(sid, cid) \to \forall sn, loc, cn, hd\ (\texttt{Subsid}(sid, sn, loc) \wedge \texttt{Company}(cid, cn, hd) \to (sn \sim cn) \vee \exists e, n, t(\texttt{Exec}(e, cid, n, t) \wedge\ contains(t, sn))),$$

two inclusion dependencies $\texttt{L}[sid] \subseteq \texttt{Subsid}[sid]$, $\texttt{L}[cid] \subseteq \texttt{Company}[cid]$, and the functional dependency $\texttt{L} : sid \to cid$. While the inclusion dependencies specify where `L` is allowed to take values from, the functional dependency gives the additional requirement on `L` that the

links must be many-to-one from *sid* to *cid*. Thus, every subsidiary must link to at most one company, but the converse need not hold. The matching constraint gives the actual matching logic and includes a listing of all the possible matching conditions for why a link may exist. Concretely, if a subsidiary id and a company id are linked, then it must be that one of the two matching conditions holds: (1) there is an overlap in the names, as specified by $sn \sim cn$, or (2) there is some executive working for the company and this executive has a title that contains the subsidiary's name.

The universally quantified conjunction `Subsid` (sid, sn, loc) $\wedge$ `Company` (cid, cn, hd) gives the *context* surrounding the occurrences of *sid* and *cid* in the source relations. In general, the matching conditions can refer to any variable in the context (e.g., sn, cn), and each matching constraint's disjunction must be true for *every* instantiation of the universal variables. For example, if a subsidiary id (sid) is associated with two or more subsidiary names (sn) in the source relation `Subsid`, then the disjunction of the two matching conditions must hold for every such name. Thus, we consider every name variation of the subsidiary; if for some variation sn the matching conditions do not hold, then that may be an indication that we do not have a true subsidiary.

The following are solutions for I w.r.t. $\mathcal{E}$:

$$J_1 = \{\mathtt{L}(s_1, c_1), \mathtt{L}(s_2, c_1)\} \qquad J_2 = \{\mathtt{L}(s_1, c_1), \mathtt{L}(s_2, c_2)\}$$
$$J_3 = \{\mathtt{L}(s_1, c_2), \mathtt{L}(s_2, c_1)\} \qquad J_4 = \{\mathtt{L}(s_1, c_2), \mathtt{L}(s_2, c_2)\}$$

We assume here that the name overlap predicate $\sim$ evaluates to true for all pairs of subsidiary name and company name occurring in our instance I (thus, "Citibank N.A." $\sim$ "Citigroup Inc" but also "Citibank N.A." $\sim$ "CIT Group Inc", and so on). Note that the link $\mathtt{L}(s_1, c_1)$ satisfies both the $\sim$ predicate and the the `Exec`-based condition, while other links satisfy only the $\sim$ predicate. The link instance $J_5 = \{\mathtt{L}(s_1, c_1), \mathtt{L}(s_1, c_2)\}$ is not a solution, since it violates the functional dependency. Finally, we note that every subinstance of a solution is always a solution. ◀

The above example shows that, in general, we allow matching of entities that are not necessarily of the same type and where a link relation is not necessarily an equivalence relation.

A key feature of the language is that matching constraints do not "force" the existence of the links. They form only a necessary condition for the existence of the links. This is a departure from the more traditional approaches based on *source-to-link* rules of the form $\alpha \rightarrow L$, which eagerly populate (or require) links in L whenever the matching condition α is true. However, when other constraints are considered (e.g., functional dependencies), the links in L may become invalid. As a result, any specification that includes source-to-link rules will likely have no solutions. In contrast, our notion of entity linking specification always has solutions. A large part of this paper will then be focused on identifying a subset of "good" solutions among all the possible solutions.

Before we proceed to define concrete classes of "good" solutions, we first define the notions of certain, possible and ambiguous links. These notions can be defined, generally, w.r.t. an arbitrary *class* of solutions, that is, w.r.t. a subset C of solutions that satisfy some property. We may also refer to the solutions in a class C as C-solutions.

▶ **Definition 3.** Assume a class C of solutions and an entity linking specification $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$. Then, given a source instance I:

(i) The set of *certain links* for I w.r.t. C and $\mathcal{E}$ is the set of links that appear in every C-solution J for I w.r.t. $\mathcal{E}$.

(ii) The set of *possible links* for I w.r.t. C and $\mathcal{E}$ is the set of links that appear in some C-solution J for I w.r.t. $\mathcal{E}$.

(iii) The set of *ambiguous* links for I w.r.t. C and $\mathcal{E}$ is given by the set difference between the possible and the certain links for I w.r.t. $\mathcal{E}$.

3 A Naive Semantics Based on Maximal Solutions

The first class of "good" solutions that we investigate is the class of maximal solutions, where "goodness" of a solution is defined as maximality w.r.t. set containment.

▶ **Definition 4.** Assume an entity linking specification $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$. Given a source instance I, a *maximal solution* for I w.r.t. $\mathcal{E}$ is a link instance J such that: (1) (J, I) satisfies Σ, and (2) there is no J' such that $J \subset J'$ and (J', I) satisfies Σ.

▶ **Example 5.** We revisit Example 2. The solutions J_1, J_2, J_3, J_4 are maximal for the given source instance I (and w.r.t. the given $\mathcal{E}$), because in each of the four instances, we cannot add any further links over the *sid*- and *cid*-values in I without violating the functional dependency. It can also be verified that these four instances are all the maximal solutions for I. ◀

Maximal solutions vs. repairs. We next show a connection with source-to-link constraints and the framework of repairs [4], which we shall use later in this section. Given an entity linking specification $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ in $\mathcal{L}_\mathbf{0}$, we first extract a source-to-link specification $\mathcal{M} = (\mathbf{S}, \mathbf{L}, \Sigma')$ as follows. For each matching constraint m_L in Σ, and given the inclusion dependencies $L[X] \subseteq R[A]$ and $L[Y] \subseteq R'[A']$, we add the following source-to-link constraint in Σ':

$$(m'_L) \quad R(\ldots, x, \ldots) \wedge R'(\ldots, y, \ldots) \wedge \ (\forall \mathbf{u}(\psi(x, y, \mathbf{u}) \rightarrow \alpha_1 \vee \ldots \vee \alpha_k)) \ \rightarrow L(x, y)$$

In the above, the occurrence of x in the R-atom is in the position of attribute A and, similarly, the occurrence of y in the R'-atom is in the position of attribute A'. Intuitively, the formula m'_L inverts the direction of the implication in m_L. For every pair x, y of values, with x coming from $R[A]$ and y coming from $R'[A']$, we check that the left-hand side of m'_L is satisfied in the source. If that is the case then m'_L requires the addition of an appropriate link in L. We can formally define this process of adding links by using the chase as follows.

First, we note that $\mathcal{M}$ can be seen as a schema mapping or data exchange setting [12] where the link schema plays the role of a target schema. The constraints in Σ' are a particular case of first-order tgds [3], that is, source-to-target tgds where the left-hand side of the tgd can contain an arbitrary first-order formula (rather than just a conjunction of atomic formulas). As shown in [3], the chase with first-order tgds behaves in the same way as the chase with regular source-to-target tgds. In particular, it terminates in polynomial time in the size of the source instance. Furthermore, since there are no existentially quantified variables in L, each m'_L is a *full* tgd; hence, the chase produces no nulls and its result for a given source instance I is unique.

Let us denote the result of the chase by $U = chase_\mathcal{M}(I)$. Intuitively, U contains all the links that are possible based on just the matching constraints and inclusion dependencies. However, when we consider the additional functional dependencies in Σ, not all the links in U are possible due to conflicts. Instead we must consider subinstances of U that are consistent. The maximal subinstances of U that are consistent are also known as the *subset repairs* [4] of U.

As it turns out, the subset repairs of $U = chase_{\mathcal{M}}(I)$ w.r.t. the functional dependencies are precisely the maximal solutions w.r.t. the original entity linking specification.

▶ **Proposition 6.** *Assume an entity linking specification $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ in $\mathcal{L}_\mathbf{0}$, and let $\mathcal{M} = (\mathbf{S}, \mathbf{L}, \Sigma')$ be the source-to-link specification constructed from $\mathcal{E}$. Furthermore, let F be the set of functional dependencies in Σ. Then, for every source instance I, the set of maximal solutions for I w.r.t. $\mathcal{E}$ is the same as the set of subset repairs of $U = chase_{\mathcal{M}}(I)$ w.r.t. F.*

Based on the previous proposition and known results about the consistent answers of projection-free queries [8], we immediately obtain the following tractability results.

▶ **Theorem 7.** *Let $\mathcal{E}$ be an entity linking specification in $\mathcal{L}_\mathbf{0}$. Then:*

- *There is a polynomial-delay algorithm that, given a source instance I, enumerates all the maximal solutions for I w.r.t. $\mathcal{E}$.*
- *There is a polynomial-time algorithm that, given a source instance I, computes the set of certain links for I w.r.t. the class of maximal solutions and $\mathcal{E}$.*
- *There is a polynomial-time algorithm that, given a source instance I, computes the set of ambiguous links for I w.r.t. the class of maximal solutions and $\mathcal{E}$.*

Proposition 6 provides a useful connection between an entirely declarative specification, based on maximal solutions w.r.t. $\mathcal{E}$, and a more procedural approach, based on chasing with $\mathcal{M}$ and then applying repairs. It also gives us polynomial-time algorithms for the three problems of interest. While this connection with repairs is directly applicable for $\mathcal{L}_\mathbf{0}$ and the semantics of maximal solutions, the situation becomes more complex for the more refined semantics that we consider later, where we will need to employ graph-based techniques to handle link values.

Deficiency of maximal solutions. We now point out the main deficiency of the semantics based on maximal solutions: in general, there may be too many maximal solutions and, hence, too few certain links. Intuitively, the semantics given by maximal solutions is too coarse-grained and does not have enough discriminating power to identify the "good" links. Consider our scenario in Example 2. We showed that there are four maximal solutions, J_1, J_2, J_3, and J_4, for the given source instance I. It can be easily seen that the set of certain links in this example is empty: there is no link that appears in all four maximal solutions and, hence, no link qualifies as a certain link. On the flip side, every link that occurs in one of the four maximal solutions is possible (and ambiguous). However, some links are clearly stronger than others. In particular, the link $L(s_1, c_1)$ relating "Citibank N.A." to "Citigroup Inc." satisfies both the $\sim$ predicate and the `Exec`-based matching condition, while the other links satisfy only the $\sim$ predicate. Intuitively, there is evidence that suggests that $L(s_1, c_1)$ is a strong link that should be differentiated from the other links.

To address the above issue, we next refine the class of "good" solutions by assigning value to links, which in turn will increase the power of discriminating among the links. In particular, it will allow to increase the number of links that qualify as certain, thus reducing ambiguity.

4 Maximum-Value Solutions

We now consider a variation of the core language $\mathcal{L}_\mathbf{0}$ that allows us to differentiate among the links in a solution, based on the evidence supporting each link. More precisely, for each link fact $L(a, b)$ in a solution, we count the number of disjuncts that are satisfied among

all the possible disjuncts $\alpha_1, \ldots, \alpha_k$ in the matching constraint m_L. We also count, for each satisfied disjunct $\alpha_i = \exists \mathbf{z}\ \phi_i$, the number of different instantiations of the existentially quantified variables $\mathbf{z}$ that witness the satisfaction of ϕ_i. Intuitively, the larger these numbers are, the better the links are.

While syntactically similar to $\mathcal{L}_\mathbf{0}$, the new language will be semantically different due to the presence of counting. In particular, in the new language, one cannot drop disjuncts that are logically redundant, since such disjuncts may be important for measuring the strength of the links, so dropping them would change the semantics. To make this behavior explicit, in the notation for a matching constraint m_L, we replace $\vee$ with a new symbol $\oplus$ as follows:

$$(m_L) \quad L(x,y) \rightarrow \forall \mathbf{u}(\psi(x,y,\mathbf{u}) \rightarrow \alpha_1 \oplus \ldots \oplus \alpha_k).$$

Syntactically, everything else is the same as in $\mathcal{L}_\mathbf{0}$. We call the resulting language $\mathcal{L}_\mathbf{0}(\oplus)$.

While the notion of a solution is the same as for $\mathcal{L}_\mathbf{0}$ and continues to be based on logical satisfaction (where $\oplus$ is interpreted as $\vee$), the notion of a "good" solution in $\mathcal{L}_\mathbf{0}(\oplus)$ will now change to reflect the strength or the value of the links. Concretely, we will identify, among the maximal solutions, a subclass of solutions that additionally maximize the total value of the links.

Assume an entity linking specification $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ in $\mathcal{L}_\mathbf{0}(\oplus)$. Let I be a source instance and J be a solution for I w.r.t. $\mathcal{E}$. We define the value of a link $L(a,b)$ in the solution J as follows.

First, if there is no matching constraint m_L for L, we take the value of $L(a,b)$ to be 1. This is the case when there are no direct requirements on the link, other than the inclusion and functional dependencies (if any). Furthermore, the link is consistent with other links in the given solution (since it appears in the solution). Giving it a value of 1 (as opposed to 0, for example) ensures that the total value of a solution strictly increases with an increase in the number of links. Assume now that there is a matching constraint m_L for L. Since (J, I) satisfies m_L, it must be that I satisfies the right-hand side of m_L where x and y are instantiated with a and b. Assume first that there is no instantiation $\mathbf{u_0}$ of the vector of universally quantified variables $\mathbf{u}$ such that $I \models \psi(a,b,\mathbf{u}_0)$. This means that the matching constraint for $L(a,b)$ is satisfied for vacuous reasons. For the same reasons as above (in the case of no matching constraint), we take the value of the link to also be 1. In all other cases, we let the value of the link be:

$$\mathrm{Val}(L(a,b)) = \min_{\mathbf{u}_0}\Big(\sum_{\alpha_i, \mathbf{z}_0} 1\Big). \tag{1}$$

In the above, $\mathbf{u}_0$ ranges over all the distinct instantiations of the vector of universally quantified variables $\mathbf{u}$ such that $I \models \psi(a,b,\mathbf{u}_0)$. We take the minimum, over all such $\mathbf{u}_0$, of the *strength* with which the source instance I satisfies the disjunction $\alpha_1 \vee \ldots \vee \alpha_k$. This strength is defined as a sum that gives a value of 1 for *every* disjunct α_i such that I satisfies $\alpha_i(a,b,\mathbf{u}_0)$ and, moreover, for *every* distinct instantiation $\mathbf{z}_0$ of the vector $\mathbf{z}$ of existentially quantified variables of α_i that makes this satisfaction hold. (Recall that α_i is, in general, of the form $\exists \mathbf{z}\ \phi_i(x,y,\mathbf{u},\mathbf{z})$.) In the case when the existentially quantified variables are missing, then we count only 1 per disjunct.

Intuitively, the sum in formula (1) calculates the matching strength by counting the number of satisfied disjuncts together with the evidence (i.e., the number of existential witnesses), while the minimum guarantees that we take the weakest matching strength among all $\mathbf{u}_0$.

The value of a solution J, denoted by $\mathrm{Val}(J)$, is then the sum of the values of the links in J.

▶ **Definition 8.** Assume an entity linking specification $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ in $\mathcal{L}_\mathbf{0}(\oplus)$. Given a source instance I, a *maximum-value solution* for I w.r.t. $\mathcal{E}$ is a link instance J such that: (1) (J, I) satisfies Σ, and (2) for every J' such that (J', I) satisfies Σ, we have that $\mathrm{Val}(J') \leq \mathrm{Val}(J)$.

▶ **Example 9.** Recall Example 2. By applying formula (1), the values of the individual links that can be formed between subsidiary ids and company ids, based on the matching constraint, are:

$$\mathrm{Val}(\mathtt{L}(s_1, c_1)) = 2 \qquad \mathrm{Val}(\mathtt{L}(s_1, c_2)) = \mathrm{Val}(\mathtt{L}(s_2, c_1)) = \mathrm{Val}(\mathtt{L}(s_2, c_2)) = 1$$

The value of 2, for the link $\mathtt{L}(s_1, c_1)$, is obtained as follows. First, for the given s_1 and c_1, there is only one way to instantiate the universally quantified variables sn, loc, cn, and hd in the matching constraint. (This is because there is only one tuple for s_1 in `Subsid`, and one tuple for c_1 in `Company`.) Hence, the min in the formula (1) is applied over a single element. Then, it can be seen that both disjuncts in the matching constraint are satisfied for s_1 and c_1. The first disjunct contributes a value of 1, since the disjunct is simply the atomic formula $sn \sim cn$. The second disjunct also contributes a value of 1, since there is only one way to instantiate the existential variables in the `Exec`-based condition (with the values corresponding for "E. McQuade"). Thus, the total strength with which the disjuncts are satisfied is 2 and, hence, the value of the link is 2. A similar evaluation takes place for the other three links, with the difference that only the first disjunct is satisfied.

Consider the earlier solutions J_1, J_2, J_3, and J_4, which were shown to be the maximal solutions. By summing up the values of their links, we obtain that $\mathrm{Val}(J_1) = \mathrm{Val}(J_2) = 3$, while $\mathrm{Val}(J_3) = \mathrm{Val}(J_4) = 2$. So, J_1 and J_2 are maximum-value solutions, while J_3 and J_4 are not. It can also be seen that there is now one certain link, namely $\mathtt{L}(s_1, c_1)$, which appears in both J_1 and J_2 and correctly relates "Citibank N.A." with "Citigroup Inc". This in contrast with the case of the maximal solutions semantics where we had zero certain links. Also, the two links $\mathtt{L}(s_2, c_1)$ and $\mathtt{L}(s_2, c_2)$, relating "CIT Bank" with either "Citigroup Inc." or "CIT Group Inc." are now ambiguous, whereas in the case of the maximal solutions semantics all four links were ambiguous. Finally, the ambiguity of $\mathtt{L}(s_2, c_1)$ and $\mathtt{L}(s_2, c_2)$ is, intuitively, the best we can achieve here, since there is not enough information to differentiate between the two links, based on the given specification. A human user is needed at this point to further refine the entity linking specification, possibly by using additional information (e.g., additional attributes or relations). ◀

A simple but important observation for $\mathcal{L}_\mathbf{0}(\oplus)$ is that, even though $\mathrm{Val}(L(a, b))$ was defined relative to a solution J (in which $L(a, b)$ occurs), the actual value of $\mathrm{Val}(L(a, b))$ is independent of J. This is so because, in $\mathcal{L}_\mathbf{0}(\oplus)$, the formula ψ and the disjuncts α_1, …, α_k are over the source schema. In Section 6, we will consider richer languages, where the α's can also depend on link predicates. Even though the same definitions of value and maximum-value solutions continue to apply for the richer languages, there we will have that $\mathrm{Val}(L(a, b))$ depends, in general, on the choice of the solution J in which it occurs.

▶ **Proposition 10.** *If $\mathcal{E}$ is an entity linking specification in $\mathcal{L}_\mathbf{0}(\oplus)$ and I is a source instance, then every maximum-value solution for I w.r.t. $\mathcal{E}$ is also a maximal solution for I w.r.t. $\mathcal{E}$.*

The proposition is an immediate consequence of the fact that in $\mathcal{L}_\mathbf{0}(\oplus)$, the value of a link is independent of the solution in which it occurs, and is at least one. The reverse inclusion does not hold, as seen in Example 9. Thus, for $\mathcal{L}_\mathbf{0}(\oplus)$, maximum-value solutions form a strict subclass of maximal solutions. As a consequence, the set of certain links over maximum-value solutions is often a strict superset of the certain links over maximal solutions.

We give next the main complexity result of this section, stating the tractability of $\mathcal{L}_{\mathbf{0}}(\oplus)$. In contrast with Theorem 7, which follows from results on data repairs, the proof of the following theorem is of a different nature and makes use of maximum-weight matching type of algorithms. In particular, it is based on extensions of results from [10, 16, 27].

▶ **Theorem 11.** *Let $\mathcal{E}$ be an entity linking specification in $\mathcal{L}_{\mathbf{0}}(\oplus)$. Then:*

- *There is a polynomial-delay algorithm that, given a source instance I, enumerates all the maximum-value solutions for I w.r.t. $\mathcal{E}$.*
- *There is a polynomial-time algorithm that, given a source instance I, computes the certain links for I w.r.t. the class of maximum-value solutions and $\mathcal{E}$.*
- *There is a polynomial-time algorithm that, given a source instance I, computes the ambiguous links for I w.r.t. the class of maximum-value solutions and $\mathcal{E}$.*

5 Connection to Probabilistic Approaches

In this section, we investigate the relationship between our declarative framework based on disjunctive matching constraints and existing probabilistic methods for entity resolution.

We start by introducing a simple yet powerful extension of $\mathcal{L}_{\mathbf{0}}(\oplus)$ that incorporates weights and which we call $\mathcal{L}_{\mathbf{0}}(\oplus, \mathbf{w})$. For each matching constraint

$$(m_L) \quad L(x, y) \rightarrow \forall \mathbf{u}(\psi(x, y, \mathbf{u}) \rightarrow \alpha_1 \oplus \ldots \oplus \alpha_k),$$

and for each disjunct $\alpha_i ::= \exists \mathbf{z}\ \phi_i(x, y, \mathbf{u}, \mathbf{z})$ there is now a weight function $w_{\phi_i}(x, y, \mathbf{u}, \mathbf{z})$ that returns a number. Intuitively, with each disjunct that returns true or false we also have a function that computes a weight (or a score) for that disjunct. The semantics of $\mathcal{L}_{\mathbf{0}}(\oplus, \mathbf{w})$ is the same as that of $\mathcal{L}_{\mathbf{0}}(\oplus)$ except that when counting existential witnesses for each disjunct we also multiply by the number returned by the weight function for that disjunct. Theorem 11 goes through when we replace $\mathcal{L}_{\mathbf{0}}(\oplus)$ by $\mathcal{L}_{\mathbf{0}}(\oplus, \mathbf{w})$, by the same proof.

5.1 Comparison to Probabilistic Matching

The first connection we make is to a well-known class of probabilistic matching algorithms that has originated with Fellegi and Sunter [15] and is at the core of many commercial systems including IBM's QualityStage [23], which we use as a representative example.

The probabilistic matching algorithm in QualityStage approaches record matching in three steps. First, it applies pairwise comparison functions over the individual attributes (or fields) in the two records to be compared. For each pair of attributes, the function returns a score based on two probabilities (that must be learned or given to the system a priori): the "match" probability m, which is the probability that two fields match given that it is known that the two records match, and the "unmatch" (or accidental match) probability u, which is the probability that two fields match but the records do not match. Secondly, the algorithm aggregates the scores returned by individual comparison functions by taking a weighted sum, where each comparison function has its own weight (also to be learned or given to the system a priori). Finally, a link is returned if it has high-enough aggregated score (higher than a threshold, which also must be learned or tuned).

We show that the first two steps in the above algorithm can be captured by a single disjunctive matching constraint, while the third one can be captured as an implementation step. We use a canonical example for deduplication of mailing lists.

▶ **Example 12.** The source schema $\mathbf{S}$ consists of two relation symbols: `MasterList`, representing a master list of mailing addresses, and `NewList`, a list with new mailing addresses

that must be deduplicated against the first one. Both relations are assumed to have the same schema, including personal attributes (e.g., last name `ln`, first name `fn`, etc.) and also address attributes (e.g., street name `street`, etc.). Furthermore, we assume each record has been assigned a record id (`rid`) and the deduplication problem is one of linking an `rid` from the new list to a unique `rid` in the master list. The core functionality of a QualityStage algorithm can be logically expressed by an entity linking specification $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ in $\mathcal{L}_{\mathbf{0}}(\oplus, \mathbf{w})$, where Σ consists of a single matching constraint:

$$\begin{aligned}
&\mathtt{L}(rid_1, rid_2) \rightarrow \\
&\qquad \forall ln_1, fn_1, \ldots, street_1, ln_2, fn_2, \ldots, street_2 \\
&\qquad (\mathtt{NewList}(rid_1, ln_1, fn_1, street_1, \ldots) \wedge \mathtt{MasterList}(rid_2, ln_2, fn_2, street_2, \ldots) \\
&\qquad\quad \rightarrow \mathtt{SOUNDEX}(ln_1, ln_2) \oplus \mathtt{SOUNDEX}(fn_1, fn_2) \oplus \ldots \oplus \mathtt{EDIT}(street_1, street_2)),
\end{aligned}$$

along with the obvious inclusion dependencies. Although QualityStage does not have cardinality constraints, it is natural to add to our specification the functional dependency $\mathtt{L} : \mathtt{rid}_1 \rightarrow \mathtt{rid}_2$. In the above matching constraint, each disjunct calls a QualityStage built-in comparison function (e.g., `SOUNDEX`, which compares how similar two names sound, or edit distance `EDIT`), by passing the arguments to be compared. In turn, each call to a QualityStage comparison function returns a weight that depends on the given arguments and also on the aforementioned probabilities m and u for the particular attribute. The weight of each possible link is then the sum of the weights for all the disjuncts. Maximum-value solutions are obtained as solutions (containing non-conflicting links) that maximize the total value. (QualityStage has the additional requirement that only links whose weights are above a certain threshold are considered possible. This can be easily added in an implementation on top of our maximum-value semantics.)

We note that the above matching constraint uses only a fragment of $\mathcal{L}_{\mathbf{0}}(\oplus, \mathbf{w})$ where each disjunct is a simple atomic formula with no existential quantification and no conjunction. ◀

5.2 Comparison to Markov Logic Networks for Entity Resolution

We now connect to a richer probabilistic framework, that of Markov Logic Networks (MLNs) [29], which in general allows for arbitrary first-order formulas to be interpreted in a probabilistic sense. We show that a class of MLNs that is useful for entity resolution [30] is captured, in a precise sense, by the fragment of $\mathcal{L}_{\mathbf{0}}(\oplus, \mathbf{w})$ with no universal quantification. Thus, rather surprisingly, a purely probabilistic approach (based on MLNs) can be captured in a deterministic way (via $\mathcal{L}_{\mathbf{0}}(\oplus, \mathbf{w})$). We make use of this correspondence to obtain a polynomial-delay algorithm for enumerating the maximum-probability worlds in the MLN setting, and polynomial-time algorithms for finding the certain and ambiguous links over maximum-probability worlds. These are the first polynomial-time results, to the best of our knowledge, for MLN-based entity resolution.

5.2.1 Markov Logic Networks: Preliminaries

The fragment of MLNs that we consider is defined as follows. For simplicity of discussion, we assume that there is one single link symbol L; the same definitions extend immediately to the case of multiple link symbols in the schema. A *linear MLN* $\mathcal{M}$ is a set of formulas $\sigma_i \rightarrow L(x, y)$ (for $1 \leq i \leq n$), each with a weight w_i, where σ_i is a conjunction of atomic formulas over the source, and where the free variables of σ_i include x and y. Examples of linear MLN formulas for entity resolution appear in [30], with the provision that the role of the link relation is played there by the `Equals` predicate. Later we also consider

extensions of linear MLNs where a link symbol may also appear in the left-hand side, thus allowing for inter-dependencies among the links. We assume that the same requirements we have for the presence of inclusion dependencies involving the link relation L in our entity linking specifications are also required in the MLN setup; also, as with our entity linking specifications, there may be functional dependencies on L.

Note that the formulas $\sigma_i \to L(x, y)$ in linear MLNs are source-to-link constraints; thus, they fall in the category of rules that eagerly populate the link relations. As discussed earlier in Section 2, such specifications may not have solutions. However, in the MLN framework, these formulas are not required to be satisfied in a hard logical sense but rather in a probabilistic sense, which allows for violations and which we explain next.

Fix a source instance I. For each "possible world", that is, choice of link instance L_0 for L satisfying the inclusion dependencies w.r.t. I and the functional dependencies on L, we assign a probability to that possible world, based on the source-to-link formulas in $\mathcal{M}$ and their weights, as follows. Let K be an instance over the combined source and link schema, and let γ be a formula over the combined schema. A *(K, γ)-valuation* (or simply *valuation*, if K and γ are fixed or understood) is a function v from the free variables of γ to members of the domain of K. Denote by $|\gamma|$ the number of valuations that make γ true in K. Then the probability assigned to a link instance L_0 (for a given source instance I) is proportional (see also [30]) to $e^{w_1|\sigma_1 \to L| + \cdots + w_n|\sigma_n \to L|}$, where the role of L is played by L_0. (These probabilities are scaled so that they sum up to 1, over all choices for L_0.) Intuitively, the probability of a world increases with the number of valuations that make a formula in $\mathcal{M}$ true and also with the weight of the formula.

Define a link (a tuple over the L schema) to be *certain* (w.r.t. the class of maximum-probability worlds) if it is in every maximum-probability world w.r.t. $\mathcal{M}$ (for the given source instance I). Similarly, we define *ambiguous* links.

5.2.2 Translation to $\mathcal{L}_0(\oplus, \mathbf{w})$

Given the linear MLN with formulas $\sigma_i \to L(x, y)$ with weight w_i, for $1 \le i \le n$, we define the *corresponding entity linking specification in* $\mathcal{L}_0(\oplus, \mathbf{w})$ to consist of the matching constraint $L(x, y) \to \exists \mathbf{z_1}\sigma_1 \oplus \cdots \oplus \exists \mathbf{z_n}\sigma_n$, where $\mathbf{z_i}$ consists of the free variables of σ_i other than x and y, and where the disjunct $\exists \mathbf{z_i}\sigma_i$ has weight w_i, for $1 \le i \le n$. Also, this corresponding entity linking specification has the same inclusion dependencies and functional dependencies on L as the linear MLN. Note that the weight function for each disjunct σ_i is a constant, whereas for QualityStage we needed in general nonconstant weight functions for each disjunct in a matching constraint.

Let $\mathcal{M}$ be an MLN, and let $\mathcal{E}$ be the corresponding entity linking specification in $\mathcal{L}_0(\oplus, \mathbf{w})$. For a given source instance I, let us denote the set of maximum-probability worlds w.r.t. $\mathcal{M}$ by Max Probability Worlds$_{\mathcal{M}}(I)$, the set of solutions w.r.t. $\mathcal{E}$ by Solutions$_{\mathcal{E}}(I)$, and the set of maximum-value solutions w.r.t. $\mathcal{E}$ by Max Value Solutions$_{\mathcal{E}}(I)$.We have the following result, interrelating maximum-value solutions and maximum-probability worlds.

▶ **Theorem 13.** *Let $\mathcal{M}$ be a linear MLN, let $\mathcal{E}$ be the corresponding entity linking specification in $\mathcal{L}_0(\oplus, \mathbf{w})$, and let I be a source instance. Then:*

- *Max Value Solutions$_{\mathcal{E}}(I)$ = Max Probability Worlds$_{\mathcal{M}}(I) \cap$ Solutions$_{\mathcal{E}}(I)$.*
- *The certain links for I w.r.t. the class of maximum-probability worlds and $\mathcal{M}$ are precisely the certain links for I w.r.t. the class of maximum-value solutions and $\mathcal{E}$.*

Note that the second part of the theorem holds even though the sets of maximum-value solutions and of maximum-probability worlds do not coincide.

The proof of the second part is based on a characterization of maximum-probability worlds in terms of maximum-value solutions. (The details of this characterization will appear in the full version of the paper.) We also use that characterization to prove the analog of Theorem 11 for linear MLNs, that is, that for linear MLNs, there is a polynomial-delay algorithm for enumerating the maximum-probability worlds, and polynomial-time algorithms for finding the certain and ambiguous links.

5.3 Deterministic vs. Probabilistic: Discussion

We showed that our declarative language can capture important classes of probabilistic methods, under a suitable extension that allows for weights. While this translation is interesting in itself, we envision our language to be used as a deterministic framework (i.e., no probabilities) where the rules that govern the links are written out explicitly, by a domain expert, as semantic rules with a true/false interpretation and (primarily) with no weights (other than 1) in the disjuncts.

While probabilistic methods may provide more automation via learning algorithms to tune or learn weights, probabilities, and thresholds, these methods are also more opaque in that it is hard to explain the results other than in terms of scores or probabilities in the underlying model. These methods may also be hard to customize when the results are not satisfactory. A simple change in a parameter or a threshold may often have unintended consequences. In contrast, a high-level deterministic language (such as $\mathcal{L}_{\mathbf{0}}(\oplus)$) provides a more transparent way for linking entities where the results can be explained in terms of the rules (disjuncts) that are satisfied. When the results are not satisfactory, rather than changing some numbers, a domain expert can explicitly refine the entity linking logic by adding or removing disjuncts, or by adding, removing or changing a conjunct within a particular disjunct. We note that similar observations were made in the context of information extraction (IE) systems [7], where it is observed that rule-based IE systems are the dominant systems adopted by commercial companies for similar reasons (i.e., they are declarative, and easier to understand, to explain, and to incorporate domain knowledge).

6 More Expressive Languages

We now explore extensions of the core language $\mathcal{L}_{\mathbf{0}}$, to allow a matching constraint for a link to possibly refer to other links. These extensions allow us to express what is usually called *collective entity resolution* [6], that is, the process of creating multiple types of links *together*.

The matching constraint for a link symbol L has the same form m_L as in Section 2. However, in each disjunct $\alpha_i ::= \exists \mathbf{z}\ \phi_i(x, y, \mathbf{u}, \mathbf{z})$, the formula ϕ_i can now be a conjunction of source *and link* atomic formulas, along with equalities and other built-in or user-defined boolean predicates. If a specification is not allowed to have recursion among the link predicates, we call the resulting language $\mathcal{L}_{\mathbf{1}}$. Thus, in $\mathcal{L}_{\mathbf{1}}$, there is a hierarchy of links, where a matching constraint for a link L may call only links that are strictly lower in the hierarchy than L. When recursion is allowed, we call the language $\mathcal{L}_{\mathbf{2}}$. So $\mathcal{L}_{\mathbf{1}}$ is a sublanguage of $\mathcal{L}_{\mathbf{2}}$. The corresponding variations for maximum-value solutions, $\mathcal{L}_{\mathbf{1}}(\oplus)$ and $\mathcal{L}_{\mathbf{2}}(\oplus)$, are defined as in the case of $\mathcal{L}_{\mathbf{0}}$. We also consider the corresponding weighted versions $\mathcal{L}_{\mathbf{1}}(\oplus, \mathbf{w})$ and $\mathcal{L}_{\mathbf{2}}(\oplus, \mathbf{w})$.

▶ **Example 14.** Consider a bibliographic example where we link papers (from one database) with articles (from another database), while also linking the corresponding venues. The source schema $\mathbf{S}$ consists of $\texttt{Paper}(pid, title, venue, year)$ and $\texttt{Article}(ano, title, journal, year)$.

Here, `pid` is a unique id assigned to `Paper` records, while `venue` could be a conference, a journal, or some other place of publication. The `Article` relation represents publications that appeared in journals, and `ano` is a unique id assigned to such records. The link schema $\mathbf{L}$ consists of two relations: `PaperLink` (pid, ano) and `VenueLink` $(venue, journal)$. The first relation is intended to link paper ids from `Paper` with article numbers from `Article`, when they represent the same publication. The second relation is intended to relate `journal` values that occur in `Article` (e.g., "ACM TODS") to journal values that occur under the `venue` field in `Paper` (e.g., "TODS").

A possible entity linking specification in $\mathcal{L}_\mathbf{2}$ is $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$, where Σ contains:

$$\begin{aligned}\texttt{VenueLink}(ven, jou) \rightarrow\ & (ven \sim_1 jou) \\ \vee\ & \exists pid, t_1, y_1, ano, t_2, y_2\ (\ \texttt{Paper}(pid, t_1, ven, y_1) \\ & \qquad \wedge\ \texttt{Article}(ano, t_2, jou, y_2) \\ & \qquad \wedge\ \texttt{PaperLink}(pid, ano)\ )\end{aligned}$$

$$\begin{aligned}&\texttt{PaperLink}(pid, ano) \rightarrow \\ &\quad \forall t_1, ven, y_1, t_2, jou, y_2\ (\ \texttt{Paper}(pid, t_1, ven, y_1) \wedge \texttt{Article}(ano, t_2, jou, y_2) \\ &\qquad\qquad \rightarrow ((t_1 \sim_2 t_2) \wedge (y_1 = y_2)) \vee ((t_1 \sim_2 t_2) \wedge \texttt{VenueLink}(ven, jou))\ )\end{aligned}$$

The first constraint specifies that we may link a venue with a journal only if their string values are similar (via some similarity predicate $\sim_1$), or if there are papers and articles that have been published in the respective venue and journal and that are linked via `PaperLink`. The second constraint specifies that we may link a paper with an article only if their titles are similar (via a similarity predicate $\sim_2$) and their years of publication match exactly, or if their titles are similar and their venues of publications are linked via `VenueLink`.

Additionally, Σ includes two functional dependencies on `PaperLink`: $pid \rightarrow ano$, $ano \rightarrow pid$, to reflect that each paper id in `Paper` must match to at most one article number in `Article`, and vice-versa. We do not require any functional dependencies on `VenueLink`; thus, we could have multiple venue strings in `Paper` matching with a journal string in `Article`, and vice-versa. We also include in Σ the expected inclusion dependencies from the link attributes to the corresponding source attributes (e.g., $\texttt{PaperLink}[pid] \subseteq \texttt{Paper}[pid]$).

With a simple modification, where we remove the second disjunct in the matching constraint for `PaperLink`, we obtain a different entity linking specification that is in $\mathcal{L}_\mathbf{1}$. While the advantage of such specification is that it is non-recursive, the modified specification is more constrained: the matching conditions for `PaperLink` are stricter now (whereas before we had a disjunction of conditions). As a result, there will be less possible links for the modified specification. ◀

6.1 Results for $\mathcal{L}_1$ and $\mathcal{L}_2$

We now focus on the computational complexity of the relevant problems (computing/enumerating maximum-value solutions and computing certain and ambiguous links). We show that we hit intractability in general, even in the case of $\mathcal{L}_\mathbf{1}$, the non-recursive fragment of $\mathcal{L}_\mathbf{2}$. On the other hand, we show that there is a large syntactic fragment of $\mathcal{L}_\mathbf{1}$ that is tractable. Finally, we show that the correspondence between $\mathcal{L}_\mathbf{0}(\oplus, \mathbf{w})$ and MLNs breaks when we go to the richer $\mathcal{L}_\mathbf{1}(\oplus, \mathbf{w})$.

Our first result, for $\mathcal{L}_\mathbf{1}(\oplus)$, states the NP-completeness of determining whether there exists a solution of at least a given value. In turn, this implies that there is no polynomial-time algorithm to compute one maximum-value solution (unless P = NP). Hence, there is no

polynomial-delay algorithm for the problem of enumerating maximum-value solutions (again, unless P = NP).

▶ **Theorem 15.** *There is a fixed entity linking specification $\mathcal{E}$ in $\mathcal{L}_\mathbf{1}(\oplus)$ for which the following problem is* NP*-complete: Given source instance I and positive integer k, is there a solution for I w.r.t. $\mathcal{E}$ of value at least k?*

We now turn our attention to the problems of computing certain and ambiguous links. If C is a class of solutions and $\mathcal{E} = (\mathbf{L}, \mathbf{S}, \Sigma)$ is a fixed entity linking specification, then *recognizing certain links w.r.t. C and $\mathcal{E}$* is the following decision problem: given a source instance I and a link l, is l a certain link for I w.r.t. C and $\mathcal{E}$? The problem of *recognizing ambiguous links w.r.t. C and $\mathcal{E}$* is defined in a similar way. Here, we investigate the complexity of recognizing certain and ambiguous links for the class of all maximum-value solutions for entity linking specifications in $\mathcal{L}_\mathbf{1}(\oplus)$. The main result is that there is an entity linking specification in $\mathcal{L}_\mathbf{1}(\oplus)$ for which no polynomial-time algorithms for recognizing certain and ambiguous links exist, unless NP = coNP.

By Theorem 15, there is an entity linking specification $\mathcal{E}$ in $\mathcal{L}_\mathbf{1}(\oplus)$ such that the following problem is NP-complete: given a source instance I and a positive integer k, is there a solution for I of value at least k? For that particular specification, recognizing certain links and recognizing ambiguous links are trivial problems because no link is certain and every link is ambiguous; intuitively, this is so because $\mathcal{E}$ encodes 3-Colorability, a problem that has "symmetries".

To establish the intractability of recognizing certain and ambiguous links, we bring into the picture the concept of a *frozen variable* from constraint satisfaction. An instance of the constraint satisfaction problem consists of a set of variables, a domain of values for each variable, and a set of constraints that restrict the combinations of values that some tuples of variables may take. A *solution* to such an instance is an assignment of values to variables so that all constraints are satisfied. A variable is *frozen* if it takes the same value in all solutions of a given instance. Jonsson and Krokhin [25] showed that for every constraint satisfaction problem over a two-element domain the problem of recognizing frozen variables exhibits the following trichotomy: it is in PTIME or it is coNP-complete or it is DP-complete. Recall that DP is the class of all decision problems that can be written as the conjunction of a problem in NP and a problem in coNP; in particular, both NP and coNP are subclasses of DP (see also [28]). Constraint satisfaction problems over a two-element domain can be thought of as variants of boolean satisfiability. An important such NP-complete variant is Positive-1-in-3-SAT, which asks: given a positive 3CNF-formula φ (i.e., a 3CNF-formula in which each clause has the form $(x \vee y \vee z)$), is there a 1-in-3 satisfying truth assignment (i.e., a truth assignment that makes exactly one variable true in every clause of φ)? Theorem 6.1 in [25] implies that the following problem is DP-complete: given a positive 3CNF-formula φ and a variable x of φ, is it true that there is a 1-in-3 satisfying truth assignment for φ and the variable x is frozen? By exploiting the above result, we are able to establish the intractability of recognizing certain and ambiguous links for entity linking specifications in $\mathcal{L}_\mathbf{1}(\oplus)$.

▶ **Theorem 16.** *There is a fixed entity linking specification $\mathcal{E}$ in $\mathcal{L}_\mathbf{1}(\oplus)$ such that:*

- *Unless* NP = coNP*, there is no polynomial-time algorithm for recognizing certain links w.r.t. to the class of all maximum-value solutions and $\mathcal{E}$.*
- *Unless* NP = coNP*, there is no polynomial-time algorithm for recognizing ambiguous links w.r.t. to the class of all maximum-value solutions and $\mathcal{E}$.*

While the above complexity results for $\mathcal{L}_1$ show intractability in general, the next theorem gives special conditions under which the same problems become tractable for $\mathcal{L}_1(\oplus)$. However, if any of the conditions fails to be satisfied, then we fall back into intractability. We say that an entity linking specification is *2-level hierarchical* if the link relations each fall into one of two disjoint sets, the "top-level links" and the "bottom-level links". The right-hand side of the matching constraints for the bottom-level link relations can refer only to source relations and built-in predicates (like equality, similarity, and string containment). The right-hand side of the matching constraints for the top-level link relations can refer only to bottom-level link relations, source relations, and built-in predicates.

▶ **Theorem 17.** *Assume that the entity linking specification is 2-level hierarchical. Assume also the following three conditions.*

1. *The top-level links have no FDs.*
2. *There are no universal quantifiers in the matching constraints for the top-level links.*
3. *Each disjunct in each matching constraint for each top-level link refers to at most one bottom-level link relation.*

Then there is a polynomial-delay algorithm to enumerate the maximum-value solutions, and there are polynomial-time algorithms to compute the certain and the ambiguous links.

Furthermore, if any of the three assumptions (1), (2), or (3) is violated, then it may be NP*-complete even to decide the following: Given source instance I and positive integer k, is there a solution for I of value at least k?*

As an immediate application of the above theorem, recall the entity linking specification in $\mathcal{L}_2$ for `VenueLink` and `PaperLink` in Example 14, and the entity linking specification in $\mathcal{L}_1$ obtained from it by the modification described in the same Example 14. The entity linking specification in $\mathcal{L}_1$, where `VenueLink` is the top-level link, and `PaperLink` is the bottom-level link, satisfies the assumptions of Theorem 17, and so enjoys the desirable properties in the conclusions of (the positive part of) the theorem. Interestingly enough, it turns out that even the entity linking specification in $\mathcal{L}_2$ for this example enjoys the desirable properties.

We close this section by considering the weighted versions $\mathcal{L}_1(\oplus, \mathbf{w})$ and $\mathcal{L}_2(\oplus, \mathbf{w})$. Theorem 13 shows a precise correspondence between a linear MLN and its corresponding entity linking specification in $\mathcal{L}_0(\oplus, \mathbf{w})$. Does this correspondence carry over to $\mathcal{L}_1(\oplus, \mathbf{w})$ or $\mathcal{L}_2(\oplus, \mathbf{w})$? Let us define *extended linear MLNs* to be defined like linear MLNs, except that instead of taking σ_i to be a conjunction of atomic formulas over the source, we allow these atomic formulas to also involve another link relation. We then define the corresponding entity linking specification as before. It can be shown that the analog of Theorem 13 fails (the details will appear in the full version of the paper). Thus, the two frameworks, one based on deterministic entity linking specifications, the other based on probabilistic Markov Logic Networks, diverge when allowing for inter-dependencies among links.

7 Related Work

As mentioned in the introduction, there has been extensive earlier work on entity resolution; overviews can be found in the recent surveys [14] and [18] and the tutorial [19]. We have also made in-depth connections to existing probabilistic approaches for entity resolution. We now comment briefly on some other declarative approaches to entity resolution.

An early argument in favor of using link-centric constraints to specify links in a declarative manner appeared in [1], but no formal language, semantics or algorithms were given there. We already discussed Dedupalog [2], a high-level framework that enables collective entity

resolution through the use of Datalog-like constraints. The result of executing a Dedupalog program is an instance that satisfies the hard constraints of the program but may violate the soft constraints of the program. Thus, a Dedupalog program is only a guideline for the implementation, which is an algorithm that attempts to minimize the number of violations of soft constraints. In contrast, the constraints we use form a truly declarative specification, by stating only the necessary conditions that must be satisfied by links, thus decoupling the specification from any implementation.

The language LinQL [20] uses SQL-like syntax to define similarity predicates among string-valued attributes only. In contrast, we create links among structured entities, and the LinQL similarity functions could be used as one ingredient in our framework. Matching dependencies (MDs) were introduced in [13] to enforce equality on attribute values based on matching conditions. In effect, MDs are source-to-link constraints that may modify source relations. MDs have been given operational semantics in [5] via a variation of the chase procedure that fixes violations of a given set of MDs. Like Dedupalog, MDs look only at equivalence (same-as) type of linkage.

8 Concluding Remarks

We laid the foundation for a truly declarative entity-linking framework that is based on specifying only the desired properties of the links. We identified a class of maximum-value solutions for entity linking specifications, and studied the computational complexity of producing such solutions and identifying certain and ambiguous links. This work opens up several new directions in reasoning about entity linking specifications. These include studying the implication and equivalence of entity linking specifications (e.g., deciding when two such specifications have the same certain links), as well as delineating the expressive power of the languages we introduced. More broadly, this work may also provide a different perspective for linking heterogeneous entities in the Semantic Web.

Acknowledgements. Kolaitis is partially supported by NSF Grant IIS-1217869; Tan is partially supported by NSF grant IIS-1450560.

References

1 Bogdan Alexe, Douglas Burdick, Mauricio A. Hernández, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, Ioana R. Stanoi, and Ryan Wisnesky. High-Level Rules for Integration and Analysis of Data: New Challenges. In *LNCS 8000: In Search of Elegance in the Theory and Practice of Computation*, pages 36–55, 2013.

2 A. Arasu, C. Re, and D. Suciu. Large-Scale Deduplication with Constraints using Dedupalog. In *ICDE*, pages 952–963, 2009.

3 M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Solutions and Query Rewriting in Data Exchange. *Inf. Comp.*, pages 28–51, 2013.

4 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *PODS*, pages 68–79, 1999.

5 Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory of Computing Systems*, 52(3):441–482, 2013.

6 Indrajit Bhattacharya and Lise Getoor. Collective Entity Resolution in Relational Data. *TKDD*, 1(1), 2007.

7 Laura Chiticariu, Yunyao Li, and Frederick R. Reiss. Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems! In *EMNLP*, pages 827–832, 2013.

8 Jan Chomicki and Jerzy Marcinkowski. Minimal-Change Integrity Maintenance using Tuple Deletions. *Inf. Comp.*, 197:90–121, 2005.

9 Xin Dong, Alon Y. Halevy, and Jayant Madhavan. Reference Reconciliation in Complex Information Spaces. In *SIGMOD*, pages 85–96, 2005.

10 J. Edmonds. Maximum Matching and a Polyhedron with 0,1-vertices. *Journal of Research National Bureau of Standards Section B*, 69:125–130, 1965.

11 Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate Record Detection: A Survey. *IEEE TKDE*, 19(1):1–16, 2007.

12 R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science (TCS)*, 336(1):89–124, 2005.

13 Wenfei Fan. Dependencies Revisited for Improving Data Quality. In *PODS*, pages 159–170, 2008.

14 Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Morgan & Claypool Publishers, 2012.

15 I. P. Fellegi and A. B. Sunter. A Theory for Record Linkage. *J. Am. Statistical Assoc.*, 64(328):1183–1210, 1969.

16 K. Fukuda and T. Matsui. Finding All the Perfect Matchings in Bipartite Graphs. *Appl. Math. Lett.*, 7(1):15–18, 1994.

17 H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *VLDB*, pages 371–380, 2001.

18 Venkatesh Ganti and Anish Das Sarma. *Data Cleaning: A Practical Perspective*. Morgan & Claypool Publishers, 2013.

19 Lise Getoor and Ashwin Machanavajjhala. Entity Resolution: Theory, Practice & Open Challenges. *PVLDB*, 5(12):2018–2019, 2012.

20 Oktie Hassanzadeh, Anastasios Kementsietsidis, Lipyeow Lim, Renée J. Miller, and Min Wang. A Framework for Semantic Link Discovery over Relational Data. In *CIKM*, pages 1027–1036, 2009.

21 Mauricio A. Hernández, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, and Ryan Wisnesky. HIL: A High-Level Scripting Language for Entity Integration. In *EDBT*, pages 549–560, 2013.

22 Mauricio A. Hernández and Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In *SIGMOD*, pages 127–138, 1995.

23 IBM InfoSphere QualityStage. `http://www.ibm.com/software/products/en/ibminfoqual`.

24 D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. On Generating All Maximal Independent Sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

25 Peter Jonsson and Andrei A. Krokhin. Recognizing Frozen Variables in Constraint Satisfaction Problems. *Theoretical Computer Science (TCS)*, 329(1-3):93–113, 2004.

26 Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record Linkage: Similarity Measures and Algorithms. In *SIGMOD*, pages 802–803, 2006.

27 K.G. Murty. An Algorithm for Ranking All the Assignments in Order of Increasing Cost. *Operations Research*, 16(3):682–687, 1968.

28 C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

29 Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, 2006.

30 Parag Singla and Pedro Domingos. Entity Resolution with Markov Logic. In *ICDM*, pages 572–582, 2006.

Asymptotic Determinacy of Path Queries using Union-of-Paths Views

Nadime Francis

ENS-Cachan, Inria
francis@lsv.ens-cachan.fr

Abstract

We consider the view determinacy problem over graph databases for queries defined as (possibly infinite) unions of path queries. These queries select pairs of nodes in a graph that are connected through a path whose length falls in a given set. A view specification is a set of such queries. We say that a view specification $\mathbf{V}$ determines a query Q if, for all databases D, the answers to $\mathbf{V}$ on D contain enough information to answer Q.

Our main result states that, given a view $\mathbf{V}$, there exists an explicit bound that depends on $\mathbf{V}$ such that we can decide the determinacy problem for all queries that ask for a path longer than this bound, and provide first-order rewritings for the queries that are determined. We call this notion asymptotic determinacy. As a corollary, we can also compute the set of almost all path queries that are determined by $\mathbf{V}$.

1998 ACM Subject Classification H.2.4 [Database Management]: Systems – *Query processing*, H.2.3 [Database Management]: Languages – Query languages

Keywords and phrases Graph databases, Views, Determinacy, Rewriting, Path queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.44

1 Introduction

View determinacy is a static analysis problem on databases that consists in deciding whether a given set of initial queries, called a view, contains enough information to answer a new query, and this on all databases. Solving this problem has many applications, namely in query optimization and caching. Assume that querying the database is costly, but that answers to all previous queries are kept in cache. Then it is useful to know whether a new query can be answered using only cached information and without accessing the database. Query determinacy can also be stated as a security problem. Assume that views represent information that can be publicly accessed, but that the considered query contains private data that should not be disclosed. Then it should be ensured that the view does not determine the query.

We consider this question over graph databases. Graph databases are relational databases in which all relations are binary. Equivalently, they can be seen as directed graphs with edges labeled from a finite alphabet. Such databases arise naturally in several scenarios, which include social networks, crime detection, biological data and the semantic Web. For instance, in social networks, individual data such as name or phone number are represented as nodes, whereas relationships between members of the network are edges linking the corresponding nodes and labeled by the nature of the relationship. Thus, a person X is a friend of a person Y if there is an edge going from X to Y with label *friend*.

Information contained in a graph database does not only lie in the content of the graph but also in its topology, that is in *how* the different data nodes are connected to each other. Typical queries then naturally ask about topological properties of the graph, namely the

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 44–59

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

existence of links, paths, and so on. In a social network, a user X could be interested in computing the transitive closure of the *friend* relation: she would like to retrieve all nodes Y such that there is a path going from X to Y using only the *friend* label. Relevant query and view languages to consider in this context should have at least this expressive power.

The determinacy problem has been considered in various contexts (see [1], [5] among others). It was shown in [2] that determinacy is decidable when queries and views are defined as path queries, that is, queries Q_k that select pairs of nodes (x, y) such that there is a path from x to y whose length is a given integer k. For instance, it proves that the two views Q_3 and Q_4 determine the query Q_5, which is not immediate to see. The main contribution of our work is to extend this result by considering a broader class of views that allows disjunction. For instance, a query $Q_{k,\ell}$ selects pairs of node that are linked either by a path of length k or a path of length ℓ. A typical case covered by our work is the following: views are Q_2, $\mathrm{Q}_{1,2}$ and $\mathrm{Q}_{2,3}$, and we will see that these views determine the query Q_5.

More precisely, we consider here arbitrary unions of path queries. A union of path queries Q is a query that selects pairs of nodes in a graph that are connected through a path whose length falls in a given set. Note that we do not have any restriction on how these sets are defined, in particular we do not require them to be finite, or even have a finite representation. Of course, our algorithmic constructions only work when these representations are effective, but our theoretical criteria do not require it. We actually require very little of these representations, and a lot of formalisms would fit our needs. For instance, regular path queries on a one letter alphabet could define the query Q_{odd} that selects all pairs of nodes linked by a path of odd length, and a query similarly defined by a context-sensitive language could be Q_{exp}, that selects pairs of nodes linked by a path of length 2^n for some n. Note that a path query can be seen as a special case of union of paths queries of size 1.

In this paper, we show that, given a view $\mathbf{V}$ defined by unions of path queries, we can decide whether $\mathbf{V}$ determines a path query Q assuming Q is "big enough," that is, Q asks for the existence of a path longer than some n_0 that we can effectively compute from $\mathbf{V}$. We call this notion *asymptotic determinacy*. Although n_0 is of exponential size, our decision procedure actually works in Π_2^{P} in the size of the finite sets that are associated with the view, disregarding the infinite ones. When it concludes that $\mathbf{V}$ determines Q, we also provide a first-order rewriting of Q using $\mathbf{V}$. Otherwise, it produces a generic counter-example that shows that $\mathbf{V}$ does not determine Q. Our technique starts by reducing $\mathbf{V}$ to a much simpler view $\mathbf{V}'$ that has many useful properties, namely all queries in $\mathbf{V}'$ are finite unions, and some $Q \in \mathbf{V}'$ is actually a path query Q_c, for some integer c. This particular query is a key to our reasoning, as it allows us to reduce infinite structures to finite ones by computing modulo c. The finite number of small queries that we are not able to process are cases where both our criterion of determinacy and our generic counter-examples fail.

Related Work

The determinacy problem has been considered in [3] for regular path queries, i.e. queries that select pairs of nodes that are connected through a path whose sequence of labels satisfies some regular expression. In [3], determinacy is known as *losslessness under the exact views assumption*. However, it is still unknown whether this property is decidable and what a good rewriting language could be. Note that, on a one-letter alphabet, regular path queries are actually weaker than infinite unions of path queries.

The determinacy problem has been solved in two specific cases. First, [2] showed how to decide whether a path view determines a path query and provides first-order rewritings of the query using the view when it is the case. The work presented here can be seen as

an extension of [2], where we consider more expressive views, by allowing (possibly infinite) disjunction. In [2], a simple criterion is given: the view determines the query if and only if the view image of a simple path satisfying the query is connected. We will see in Example 14 and Example 15 that this decision criterion no longer applies here, as the view images in these examples are connected, but the view does not determine the query.

Secondly, [4] proved that regular path queries can always be rewritten as Datalog queries using regular path views, assuming *monotone determinacy*. Monotone determinacy is a stronger form of determinacy that is known to be decidable in this setting. It basically requires both determinacy and the fact that rewritings of the query using the views are monotone. Our work can be seen as an attempt to lift this monotonicity assumption, while still retaining some of the expressive power of regular path queries and views, such as disjunction and transitive closure. Of course, without assuming monotonicity there can be no hope of finding a rewriting in Datalog, since it can only express monotone queries.

2 Preliminaries

Graph Databases

A *binary schema* σ is a finite set of binary relational predicates. A *graph database* D over σ is a relational structure over a binary schema σ. Alternatively, it can also be seen as a directed edge-labeled graph whose labeling alphabet are symbols in σ. An element in the domain of D is called a *node*.

A *path* π in a database D from x_0 to x_m is a finite sequence $\pi = x_0 a_0 x_1 \dots x_{m-1} a_{m-1} x_m$, where each x_i is a node of D, each a_i is a symbol of σ, and for all i, $a_i(x_i, x_{i+1})$ holds in D. Such a path is said to be *simple* if each node occurs at most once in π. We simply write $\pi = x_0 \dots x_m$ when the specific symbol of σ that holds for each pair $(i, i+1)$ is irrelevant. The length of π, denoted by $|\pi|$, is the length of the word $a_0 \dots a_{m-1}$, in this case m. To denote the fact that π is a path from x_0 to x_m, we will often write $x_0 \xrightarrow{\pi} x_m$. By abuse of notation, we also consider π as a graph database that contains exactly the nodes $x_0, \dots, x_m$, and in which only $a_i(x_i, x_{i+1})$ holds, for all i.

Queries

A binary *query* Q over a schema σ is a mapping associating to each graph database D over σ a binary relation Q(D) over the domain of D. In this work, we only consider the following two query languages.

A *path query* Q is defined by a single integer k. On a given database D, Q returns all the pairs of nodes (x, y) in D such that there exists a path in D from x to y of length k. In other words, $\mathrm{Q(D)} = \{(x, y) \in \mathrm{D} \mid \exists \pi, x \xrightarrow{\pi} y \text{ and } |\pi| = k\}$. For ease of notation and consistency with what follows, we will write $\mathrm{Q} = \{k\}$.

A *union of path queries* Q is defined by a (possibly infinite) set of integers $\{k_1, k_2, \dots\}$, and returns all the pairs of nodes that are connected through a path whose length belongs to the set, i.e. $\mathrm{Q(D)} = \{(x, y) \in \mathrm{D} \mid \exists \pi, x \xrightarrow{\pi} y \text{ and } |\pi| \in \mathrm{Q}\}$. By abuse of notation, Q represents both the query and the associated set. Unions of path queries are a generalization of path queries, as any path query can be seen as a union of path queries whose associated set is of size 1.

These two query languages can be compared with other core languages commonly used in the field. Path queries are conjunctive queries on a single predicate whose underlying graph is a directed path. Alternatively, they are also regular path queries whose associated

language is a single word on a single-letter alphabet. Unions of path queries are arbitrary unions of such queries. Note that these are more expressive than regular path queries on a single letter alphabet. Indeed, $Q = \{p \mid p \text{ is prime}\}$ is a union of path queries that is not regular.

We do not impose any way of representing the infinite sets associated to unions of path queries. For our constructions to be effective, we only require:

- the ability to decide, given a query, whether its associated set is infinite.
- the ability to effectively list all the elements in the associated set, when it is finite.

Thus, many formalisms would fit our needs, such as regular sets, but we could possibly think of stronger languages for finitely describing infinite sets of integers. While considering infinite unions may seem rather strange, this should be understood on a conceptual level, as a way to ease comparisons and extensions to existing work. Most of the work presented is actually only relevant to finite unions. Indeed, we will see in Lemma 4 that infinite unions cannot be used for the determinacy and rewriting of path queries, which explains the very low requirements we have on queries with an infinite associated set.

Views

Let σ and τ be two binary schemas. A *view* $\mathbf{V}$ from σ to τ is a set of binary queries over σ, one for each symbol in τ. Note that, since both σ and τ are finite, then $\mathbf{V}$ is also a finite set. By abuse of notation, we will use the same notation for both the relational predicate in τ and the corresponding query in $\mathbf{V}$. For a given graph database D over σ, $\mathbf{V}(\mathrm{D})$ is then defined as another graph database E over τ, such that for each $V \in \tau$, the interpretation of V in E is exactly $V(\mathrm{D})$. Finally, the nodes of E are exactly those that appear in one of those relations, also known as the *active domain* of E.

Determinacy

A formal definition of determinacy is given in [6] as:

▶ **Definition 1** (Determinacy). We say that a view $\mathbf{V}$ determines a query Q if:

$$\forall \mathrm{D}, \mathrm{D}', \mathbf{V}(\mathrm{D}) = \mathbf{V}(\mathrm{D}') \;\Rightarrow\; \mathrm{Q}(\mathrm{D}) = \mathrm{Q}(\mathrm{D}')$$

Intuitively, this means that a view $\mathbf{V}$ determines a query Q, which we write $\mathbf{V} \twoheadrightarrow \mathrm{Q}$, if, for all databases D, $\mathbf{V}(\mathrm{D})$ always contains enough information to answer Q on D. Moreover, we say that a query R is a *rewriting* of Q using $\mathbf{V}$ if $\mathrm{R}(\mathbf{V}(\mathrm{D})) = \mathrm{Q}(\mathrm{D})$ for all D.

The *determinacy problem* is the problem of deciding, given a view $\mathbf{V}$ and a query Q, whether $\mathbf{V} \twoheadrightarrow \mathrm{Q}$. To the best of our knowledge, its decidability status is still open when Q is a conjunctive query and $\mathbf{V}$ a conjunctive view, and also when Q is a regular path query, and $\mathbf{V}$ a regular path view. Nonetheless, several cases have been considered and solved. The results in [2] solve the problem when Q is a path query and $\mathbf{V}$ a path view. In [6], this problem is considered for Q and $\mathbf{V}$ defined by conjunctive queries, and in [4] for Q and $\mathbf{V}$ defined with regular path queries, both with the added restriction that $\mathbf{V}$ must determine Q in a monotone way, which means that a monotone rewriting of Q using $\mathbf{V}$ exists.

Here, we consider the determinacy problem for Q defined as a path query, and $\mathbf{V}$ defined as a set of unions of path queries. However, for each $\mathbf{V}$, there exist a finite number of Q on which our technique does not work. Hence, we are actually solving a slightly weaker problem, that we call the *α-asymptotic determinacy problem* which allows, for each $\mathbf{V}$, to exclude a finite number of queries Q. The excluded queries are those that ask for a path longer than

$\alpha(\mathbf{V})$, where α is a fixed function that maps each view to a natural number. Note that providing an answer or a rewriting in "almost all" cases is something that has already been considered in the same context, for instance in [2]. We can now formally state the problem and our main result:

PROBLEM :	α-ASYMPTOTIC DETERMINACY
INPUT :	A union-of-paths view $\mathbf{V}$ and a path query $\mathrm{Q} = \{n\}$ with $n > \alpha(\mathbf{V})$
QUESTION :	Does $\mathbf{V} \twoheadrightarrow \mathrm{Q}$?

▶ **Theorem 2.** *There exists an explicit and computable function α for which the α-asymptotic determinacy problem is decidable. Moreover, when the view determines the query, the decision procedure effectively computes a first-order rewriting of the query using the view.*

It will actually come from the proof that the specific α for which we can solve the problem grows exponentially in the size of the finite unions in $\mathbf{V}$, while disregarding the infinite ones. However, the decision procedure itself works with a much lower Π_2^{P} complexity.

Arithmetic Notations

Some of the proofs in this work involve a lot of arithmetic reasonings. We present here the notations that we use. Given two integers n and d, $n[d]$ represents the remainder in the division of n by d. We say that two integers n_1 and n_2 are equivalent modulo d, and we write $n_1 \equiv n_2[d]$ if they have the same remainder modulo d. We denote by $\gcd(A)$ the greatest common divisor of a set of integers A, and we use $n_1 \wedge n_2$ for $\gcd(\{n_1, n_2\})$. Additionally, for two binary relations R and S, we write $R \cdot S$ for $\{(x, z) \mid \exists y, R(x, y) \text{ and } S(y, z)\}$. Let n be a positive integer, we also use $R^1 = R$, and $R^n = R^{n-1} \cdot R$ if $n \geq 2$.

Organization

The rest of the paper investigates the determinacy problem for a path query Q using view $\mathbf{V}$ defined by unions of path queries. In Section 3, we start by providing some conditions on Q and $\mathbf{V}$ that are necessary for $\mathbf{V}$ to determine Q. Section 4 is dedicated to proving Theorem 2, which gives a procedure for deciding the determinacy problem for almost all path queries Q, as well as a first-order rewriting of Q using $\mathbf{V}$ for the queries that are determined. Finally, in Section 5, we discuss the issue that remains to be solved in order to decide determinacy for all queries.

Note that, due to space constraints, we only provide sketches of the most long and technical proofs. The missing proofs can be found in a more complete version of this paper.

3 Necessary Conditions and First Results

In this section and the next, we only consider path queries and union of path views. When we simply say "query" or "view", it is implied that they belong to those specific classes. The goal here is to provide a set of necessary conditions for a view $\mathbf{V}$ to determine a query Q.

Our first lemma states that a view $\mathbf{V}$ cannot possibly determine a query Q if $\mathbf{V}$ does not at least contain a path query. In other words, even though $\mathbf{V}$ is defined using union of path queries, at least one of them cannot make use of the union.

▶ **Lemma 3.** *Assume that a view $\boldsymbol{V}$ and query Q are such that $\boldsymbol{V} \twoheadrightarrow Q$. Then there exists $C \in \boldsymbol{V}$ such that $|C| = 1$.*

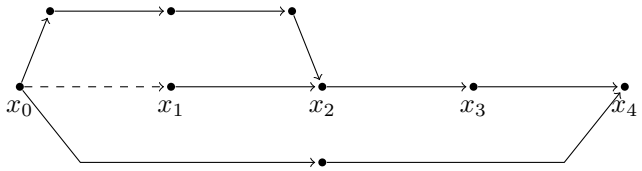


Figure 1 Illustration for the proof of Lemma 3, showing here that $V = \{2, 4\} \not\twoheadrightarrow \mathrm{Q} = \{4\}$. Following the notations in the proof, the top path is $\pi_{2,V}$ and the bottom path is $\pi_{4,V}$. Remark then that adding or removing the dashed edge does not change the view, but changes the query result.

Proof. Assume by contraposition that, for all $V \in \mathbf{V}$, $|V| > 1$. Let $\mathrm{Q} = \{n\}$. We build a database D as follows:

- D contains $n + 1$ distinct nodes $x_0, \ldots, x_n$.
- For all $i < n$, $a(x_i, x_{i+1})$ holds in D.
- For all $i \leq n$, for all $V \in \mathbf{V}$ such that $i \in V$, we add to D a simple path $\pi_{i,V}$ from x_0 to x_i, such that $|\pi_{i,V}| \in V - \{i\}$. Such a path exists because $|V| > 1$.

We then construct another database D' which is a copy of D except that $a(x_0, x_1)$ does not hold in D'. It is then easy to check that $\mathbf{V}(\mathrm{D}) = \mathbf{V}(\mathrm{D}')$ and that $\mathrm{Q}(\mathrm{D}) \neq \mathrm{Q}(\mathrm{D}')$. In particular, $(x_0, x_n) \in \mathrm{Q}(\mathrm{D})$ and $(x_0, x_n) \notin \mathrm{Q}(\mathrm{D}')$. Hence $\mathbf{V} \not\twoheadrightarrow \mathrm{Q}$, which concludes the proof. This construction is illustrated on Figure 1. ◀

Our second lemma states that we can safely ignore the queries in $\mathbf{V}$ that are defined by infinite unions. This means that if a view $\mathbf{V}$ determines a query Q, then $\mathbf{V}$ also determines Q without making use of its infinite components.

▶ **Lemma 4.** *Let Q be a query and $\boldsymbol{V}$ be a view. Let $\boldsymbol{V} = \boldsymbol{V}_f \uplus \boldsymbol{V}_\infty$, such that $\boldsymbol{V}_f$ only contains queries defined by finite sets, and $\boldsymbol{V}_\infty$ only contains queries defined by infinite sets. Then $\boldsymbol{V} \twoheadrightarrow Q$ if and only if $\boldsymbol{V}_f \twoheadrightarrow Q$.*

Proof. It is easy to see that if $\mathbf{V}_f \twoheadrightarrow \mathrm{Q}$, then $\mathbf{V} \twoheadrightarrow \mathrm{Q}$. Conversely, assume that $\mathbf{V}_f$ does not determine Q. Then there exists two databases D_1 and D_2 such that D_1 and D_2 agree on $\mathbf{V}_f$ but not on Q. Let k be the biggest number that appears in $\mathrm{Q} \cup \mathbf{V}_f$. We transform D_1 into a new database D'_1 as follows:

- We add to D'_1 $k + 1$ new nodes $x_0, \ldots, x_k$, as well as the following edges:
  - For all i, $a(x_i, x_{i+1})$ holds in D'_1.
  - $a(x_0, x_0)$ and $a(x_k, x_k)$ hold in D'_1.
- For each original node x of D_1, we add $a(x, x_0)$ and $a(x_k, x)$ to D'_1.

We then apply the same steps to D_2 and get a new database D'_2. This construction has no effect on Q or $\mathbf{V}_f$ for the original nodes of D_1 and D_2. However, it makes it so that for each $(x, y) \in \mathrm{D}_1$ (respectively D_2), for each $V \in \mathbf{V}_\infty$, $V(x, y)$ holds in $\mathbf{V}_\infty(\mathrm{D}'_1)$ (respectively $\mathbf{V}_\infty(\mathrm{D}'_2)$). Thus, we can check that D'_1 and D'_2 agree on $\mathbf{V}$ but not on Q. Hence $\mathbf{V} \not\twoheadrightarrow \mathrm{Q}$, which concludes the proof. ◀

Altogether, these two lemmas show than we can restrict our attention to views $\mathbf{V}$ that contain only queries defined by finite sets, and contain at least one query C such that $|C| = 1$. This reduction is effective if we can decide which views correspond to infinite sets, and which views correspond to singletons. We can now state the following definition for views that contain such a path query C:

▶ **Definition 5** (Complete)**.** Let $\mathbf{V}$ be a view and $C \in \mathbf{V}$ such that $C = \{c\}$. We say that $\mathbf{V}$ is *C-complete* if, for all $i \in \{0, \ldots, c - 1\}$, there exists $V \in \mathbf{V}$ and $k \in V$ such that $k \equiv i[c]$.

Our next necessary condition is an adaptation of the condition in [2]:

▶ **Claim 6.** *Let $\boldsymbol{V}$ be a view and $Q = \{n\}$. Let $\pi = x_0 \dots x_n$. If $\boldsymbol{V} \twoheadrightarrow Q$ then there is an undirected path from x_0 to x_n in $\boldsymbol{V}(\pi)$.*

This condition allows us to reduce any determinacy problem to an equivalent problem with the added hypothesis that the view is C-complete:

▶ **Lemma 7.** *Let $\boldsymbol{V}$ be a view and $C \in \boldsymbol{V}$ such that $C = \{c\}$. Let $Q = \{n\}$, and $\pi = x_0 \dots x_n$. Assume that there is a path from x_0 to x_n in $\boldsymbol{V}(\pi)$. Then we can effectively compute a view $\boldsymbol{V}'$ with $C' \in \boldsymbol{V}'$ such that $C' = \{c'\}$ and a query Q' such that $\boldsymbol{V}'$ is C'-complete and $\boldsymbol{V} \twoheadrightarrow Q$ if and only if $\boldsymbol{V}' \twoheadrightarrow Q'$.*

Sketch of proof. There are two cases to this proof. Consider the set U of all numbers that appear in $\mathbf{V}$. If $\gcd(U) = 1$, then we can construct a new query V as a combination of the other queries in $\mathbf{V}$, such that V contains some $v_i \equiv i[c]$ for all i.

Assume now that $\gcd(U) = d$, with $d \neq 1$. This means that all numbers appearing in $\mathbf{V}$ can be divided by d. Since x_0 is connected to x_n in $\mathbf{V}(\pi)$, this also means that d divides n. In this case, we build a new view $\mathbf{V}'$ and a new query Q' by dividing by d all numbers that appear respectively in $\mathbf{V}$ and Q. We then show that $\mathbf{V}$ determines Q if and only if $\mathbf{V}'$ determines Q', and we apply the first case to $\mathbf{V}'$ and Q'. ◀

Finally, the last lemma of this section shows that proving that $\mathbf{V} \twoheadrightarrow \mathrm{Q}$ for some Q also yields a lot of other determinacy results easily.

▶ **Lemma 8.** *Let $\boldsymbol{V}$ be a view and $C \in \boldsymbol{V}$ such that $C = \{c\}$. Let $Q = \{n\}$, and assume that $\boldsymbol{V} \twoheadrightarrow Q$. Then, for all positive integer k, $\boldsymbol{V} \twoheadrightarrow \{n + kc\}$.*

Proof. Let R be a rewriting of Q using $\mathbf{V}$. Let k be a positive integer. Then it is easy to check that $R \cdot C^k$ is a rewriting of $\mathrm{Q}' = \{n + kc\}$ using $\mathbf{V}$. ◀

4 Asymptotic Determinacy

The goal of this section is to prove Theorem 2. By using the results of Section 3, we can restrict our attention to C-complete views $\mathbf{V}$, with $C \in \mathbf{V}$ and such that all $V \in \mathbf{V}$ are finite. Theorem 2 is a consequence of the following proposition:

▶ **Proposition 9.** *Given a C-complete view $\boldsymbol{V}$ defined by finite unions of path queries, such that $C \in \boldsymbol{V}$ with $C = \{c\}$ for some $c \in \mathbb{N}$ and a natural number $o \in \{0, \dots, c-1\}$, it is decidable whether there exists a query $Q = \{n\}$ such that $n \equiv o[c]$ and $\boldsymbol{V} \twoheadrightarrow Q$. If this is the case, such a query Q and a first-order rewriting of Q with regards to $\boldsymbol{V}$ can be effectively computed.*

Indeed, given a C-complete view $\mathbf{V}$ and a query $\mathrm{Q} = \{m\}$, we can first decide if there exists another query n, with $n \equiv m[c]$ such that $\mathbf{V} \twoheadrightarrow \{n\}$. If this is not the case, then we can safely conclude that $\mathbf{V} \not\twoheadrightarrow \mathrm{Q}$. Otherwise, Proposition 9 gives us an explicit n that is determined by $\mathbf{V}$. If $m > n$, then Lemma 8 concludes that $\mathbf{V} \twoheadrightarrow \mathrm{Q}$. Else, Q is one of those small queries that we cannot handle, but there are only finitely many of them. Hence α in Theorem 2 can be defined as the function that maps $\mathbf{V}$ to the maximal n given by Proposition 9. In order to decide general determinacy, we would need the *smallest* $n \equiv m[c]$ that is determined by $\mathbf{V}$. This particular issue is discussed in Section 5.

The proof of Proposition 9 is divided in three parts. In Section 4.1, we introduce a tool that describes the possible behaviors that can be observed through the view $\mathbf{V}$. We use it to

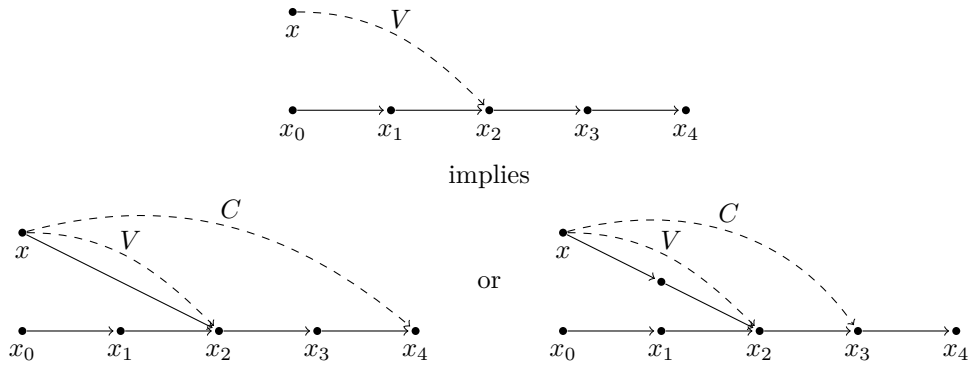


Figure 2 Example of possible behaviors for a database (full) and its view (dashed), with $C = \{3\}$ and $V = \{1, 2\}$. Assume we know the information represented in the top figure. Then, one of the two bottom pictures must hold. More generally, if $C = \{c\}$ and $V(x, x_i)$ holds in E, then $C^k(x, x_j)$ must also hold, with $j = i + (c - v[c])$ and $kc = v + (c - v[c])$ for some $v \in V$.

prove the propositions in Section 4.2 and Section 4.3. More precisely, Section 4.2 settles the case where no query $\mathrm{Q} = \{n\}$ with $n \equiv o[c]$ is determined by $\mathbf{V}$, and Section 4.3 builds an approriate query Q when one does exist.

4.1 Behavior graph

The goal of this section is to define a tool that will help us deal with the high combinatorial complexity when trying to find a target path in a database based on its view image. We now give a rough sketch of the idea behind this tool. Assume we want to prove that some database D contains a path π of length n by only looking at $\mathrm{E} = \mathbf{V}(\mathrm{D})$, where $\mathbf{V}$ is a C-complete view that contains only finite unions of path queries. If D does indeed contain such a path, then the following properties must necessarily hold in E:

C1. E contains the $n + 1$ (not necessarily distinct) nodes of π, $x_0, \ldots, x_n$.

C2. For each $V \in \mathbf{V}$ and $u \in V$, $V(x_i, x_{i+u})$ holds in E for all i.

C3. For each x in E such that $V(x, x_i)$ holds in E, there exists an appropriate value of k and j such that $C^k(x, x_j)$ holds in E. The values of k and j depend on the witness path that proves $V(x, x_i)$, as shown in Figure 2.

Of course, there are many ways for a view instance E to satisfy all these properties without D actually having a path of length n from x_0 to x_n, let alone one that goes through all the x_i's in the right order. Let μ be a path in D from some x_i to some x_j. We define the *delay* of this path as $\delta(\mu) = |\mu| - (j - i)$, that is the length of μ minus the expected length of μ if μ had been the section from x_i to x_j of a path of length n whose nodes are the x_i's. Note that $\delta(\mu)$ can be positive (μ is longer than expected), negative (μ is shorter than expected), or zero, in which case there is a path of length $(j - i)$ from x_i to x_j as intended.

Let D be a database and $\mathrm{E} = \mathbf{V}(\mathrm{D})$ such that E satisfies the necessary conditions above. For this D and E, we build a graph H_{D} that represents the delays of the paths of D that are induced by the conditions (C1), (C2) and (C3) as follows:

- H_{D} has $n + 1$ nodes that represent $x_0, \ldots, x_n$, as in (C1). We simply note them $0, \ldots, n$.
- For all $V \in \mathbf{V}$ and $u \in V$, (C2) implies that $V(x_i, x_{i+u})$ holds in E. Hence, there exists a path π in D going from x_i to x_{i+u} of length v, for some $v \in V$.
  - We represent this as an edge in H_{D} going from i to $i + u$ of label $\delta(\pi) = (v - u)$.

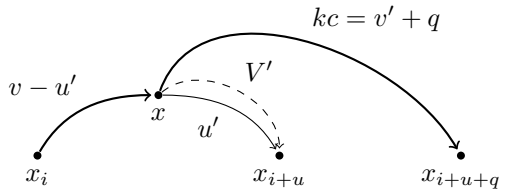


Figure 3 Illustration for the existence of the path π' of delay $(v-u)+(v'-u')$ in the construction of H_{D}. Full arrows represent paths in D and are labeled by their length. Dashed arrows represent edges in E. π' is the thick path, and $q = c - v'[c]$.

- For all $u' < v$ such that $u' \in V'$ for some $V' \in \mathbf{V}$, we know that $V'(x, x_{i+u})$ holds in E, where x is the u'th predecessor of x_{i+u} along π. We apply (C3) as shown in Figure 3. This leads to a path π' from x_i to $x_{i+u+(c-v'[c])}$ such that $\delta(\pi') = (v-u)+(v'-u')$, for some $v' \in V'$, which we similarly represent in H_{D}.

Assume that there is a path from node 0 to node n in H_{D} whose sum of labels is 0. By composing all the paths in D that led to this path in H_{D}, we can prove that there exists in D a path π from x_0 to x_n such that $\delta(\pi) = 0$. Hence, π is of length n, and we have actually found a path of length n from x_0 to x_n in D.

Consider the case where this is true for all databases D, that is, for all databases D such that $\mathbf{V}(\mathrm{D})$ satisfies the necessary conditions, H_{D} contains such a path. Then all these databases contain a path of length n from x_0 to x_n. This means that the necessary conditions for the existence of a path of length n in D are also sufficient. Since these conditions can be checked by looking only at the view instance, it implies that $\mathbf{V} \twoheadrightarrow \{n\}$.

Unfortunately, the size of H_{D} depends on the size of the target query. In order to have a representation that does not depend on n, we identify in H_{D} all nodes i and j such that $i \equiv j[c]$. Note that this is consistent with the fact that such nodes were already linked by paths of delay 0 thanks to $C \in \mathbf{V}$. This is exactly the idea behind *choice graphs*, that are formally defined below. While we do lose some information by doing this merging, these graphs are still rich enough to allow us to decide asymptotic determinacy, as we will see in the rest of the proof.

▶ **Definition 10** (Choice graph)**.** Given a C-complete view $\mathbf{V}$ such that $C \in \mathbf{V}$ with $C = \{c\}$, we define $\mathcal{H}_{\mathbf{V}}$ as the set of all directed, edge-labeled graphs H such that:

1. H has c nodes, which we will simply note $0, 1, \ldots, c-1$.
2. The edges of H carry labels in $\{-2(m-1), \ldots, 2(m-1)\}$, where m is the biggest element that appears in the views, that is $m = \max_{V \in \mathbf{V}} \max_{u \in V} u$.
3. For each $i, j \in \{0, \ldots, c-1\}$, for each $V \in \mathbf{V}$, for each $u \in V$ such that $u \equiv (j-i)[c]$, there exists $v \in V$ such that:
   - there is an edge in H from i to j labeled by $v - u$.
   - for each $V' \in \mathbf{V}$, for each $u' \in V'$, there exist $v' \in V'$ and an edge in H from i to $(j - v')[c]$ labeled by $(v-u)+(v'-u')$.

▶ Remark. Since each $H \in \mathcal{H}_{\mathbf{V}}$ has a bounded number of nodes and edges, then $\mathcal{H}_{\mathbf{V}}$ is finite. Moreover all $H \in \mathcal{H}_{\mathbf{V}}$ are complete graphs, because $\mathbf{V}$ is C-complete.

▶ **Definition 11** (Weight)**.** The weight of a path in a graph H is the sum of all labels along edges of the path. A path with no edge is of weight 0.

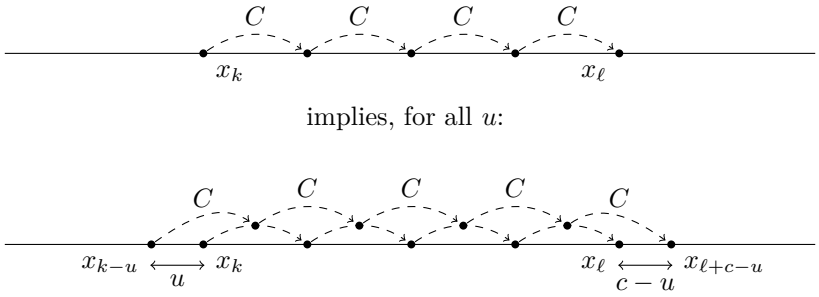


Figure 4 Illustration of the intuition for the construction of a behavior graph. Assume that the x_i's form a path of length n from x_0 to x_n. If x_k and x_ℓ are connected via a sequence of C's, represented by the dashed edges, then for all $u < c$, there exists some intermediate nodes such that x_{k-u} and $x_{\ell+c-u}$ are connected as shown in the picture.

▶ **Definition 12** (Behavior graph)**.** Given a C-complete view $\mathbf{V}$ such that $C \in \mathbf{V}$ with $C = \{c\}$, we define $\mathcal{G}_\mathbf{V}$ as the set of all directed, edge-labeled graphs G constructed as follows:

1. Pick $H \in \mathcal{H}_\mathbf{V}$, and start with $G = H$.
2. Pick $i, j \in \{0, \dots, c-1\}$ such that:
   - There exists in G a path from i to j of weight $(i-j)[c]$. Let a be the weight of a path of minimal length satisfying this property.
   - For all $a' \equiv a[c]$, there exists i', j' such that $(j'-i') \equiv (j-i)[c]$, and there is no edge from i' to j' of label a'.

   Then, for all i', j' such that $(j'-i') \equiv (j-i)[c]$, add an edge a from i' to j'.
3. Repeat step 2 until no more edges can be added.

▶ Remark.

- Step 2 of the construction of $\mathcal{G}_\mathbf{V}$ can only be applied a finite number of times for each G, since it can be done at most once for each (i, j). Moreover, there is a finite amount of choice at each step. Hence, $\mathcal{G}_\mathbf{V}$ is finite.
- As soon as there is a path from some i to some j of weight $(i-j)[c]$, then there is a weight $a \equiv (i-j)[c]$ such that all i', j' that are at the same distance than i is from j are linked by an edge of this particular weight.

Behavior graphs contain another necessary property of the existence of a path of length n that goes through the x_i's. Remark that a path π of delay $i-j$ going from x_i to x_j is actually of length 0 modulo c. Hence π appears in E as a sequence of C edges. Then the reasoning shown in Figure 4 implies the existence of paths of identical delay from nodes x_{i-u} to nodes x_{j+c-u} for all u.

All the intuitions presented in this section are made precise in Section 4.2 and Section 4.3, when this tool is actually used. In Section 4.2, we show that, if there exists some behavior graph G such that there is no path of weight 0 from 0 to o in G, then we can build two databases that agree on the view but not on paths of length $n \equiv o[c]$, for all n. In other words, we can build a database whose view satisfies all the necessary conditions for the existence of a path of length n, while still maintaining a non-zero delay between the relevant nodes. On the contrary, in Section 4.3, we show that, if for all behavior graphs G, there is a path of weight 0 from 0 to o in G, then it is enough to satisfy the necessary conditions in order to have a path of length n.

Our decision algorithm uses these properties of behavior graphs as follows. For a given C-complete view $\mathbf{V}$ and a given natural number $o \in \{0, \dots, c-1\}$, we are simply looking

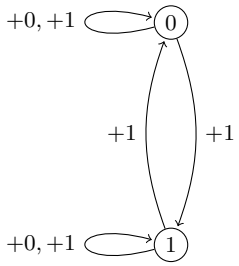


Figure 5 A behavior graph for the view defined in Example 14.

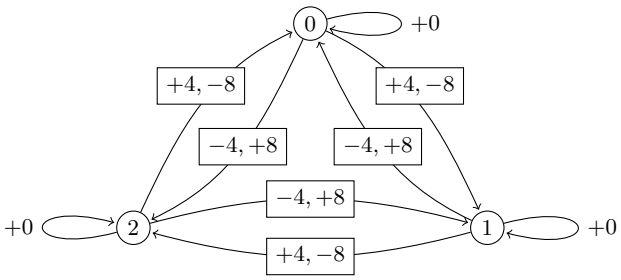


Figure 6 A behavior graph for the view defined in Example 15.

for the occurrence of a specific graph $G \in \mathcal{G}_{\mathbf{V}}$, namely one that does not contain a path of weight 0 from node 0 to node o. If we do find one such G, then, for all $n \equiv o[c]$, $\mathbf{V} \not\twoheadrightarrow \{n\}$. Otherwise, $\mathbf{V} \twoheadrightarrow \{n\}$ for some $n \equiv o[c]$. We do not actually need to compute $\mathcal{G}_{\mathbf{V}}$: we simply guess the appropriate graph G and check that it does contain the critical path. Since G is of size polynomial in $\mathbf{V}$ and the considered path, if it exists, can be assumed to be polynomial in the size of G (thanks to Bezout's Identity), our decision procedure is in PSPACE, more precisely in Π_2^{P}.

4.2 Negative direction: building counter-examples

In this section, we solve the negative case of Proposition 9 by proving the following proposition:

▶ **Proposition 13.** *Assume that there exists $G \in \mathcal{G}_{\boldsymbol{V}}$ such that there is no path of weight 0 from 0 to o. Then, for all $n \equiv o[c]$, $\boldsymbol{V} \not\twoheadrightarrow \{n\}$.*

The proof of Proposition 13 is split across Lemma 16 and Lemma 17. The canonical counter-examples that we build in order to prove that $\mathbf{V} \not\twoheadrightarrow \{n\}$ depend on whether G only contains cycles of positive or negative weights, or if it actually has both. These two cases are respectively dealt with in Lemma 16 and Lemma 17, and Example 14 and Example 15 give examples of both situations.

▶ **Example 14.** Let $\mathbf{V} = \{C, V\}$, $C = \{2\}$ and $V = \{1, 2\}$. Figure 5 represents one of the graphs in $\mathcal{G}_{\mathbf{V}}$, that additionally satisfies the condition of Lemma 16. Namely, there is no path of weight 0 from 0 to 1, and 0 only has non-negative cycles.

▶ **Example 15.** Let $\mathbf{V} = \{C, V\}$, $C = \{3\}$ and $V = \{1, 5\}$. Figure 6 represents one of the graphs in $\mathcal{G}_{\mathbf{V}}$, that additionally satisfies the condition of Lemma 17. Namely, there is no path of weight 0 from 0 to 1, and 0 has positive and negative cycles.

▶ **Lemma 16.** *Assume that there exists $G \in \mathcal{G}_{\boldsymbol{V}}$ such that 0 does not have both a cycle of positive weight and a cycle of negative weight and that there is no path of weight 0 from 0 to o. Then, for all $n \equiv o[c]$, $\boldsymbol{V} \not\twoheadrightarrow \{n\}$.*

Sketch of proof. Assume that G does not have cycles of negative weights. An example of such a case is given in Example 14. We explain how the proof works on this example.

We build a counter-example showing that $\mathbf{V}$ does not determine any odd n as follows. Start from a database D which consists of two simple paths of length n denoted $x_0 \dots x_n$ and $x'_0 \dots x'_n$. For each i, add a path of length 2 from x_i to x_{i+1}, and a path of length 2 from x'_i to x'_{i+1}, sharing their middle nodes. Notice now that you can switch the position of

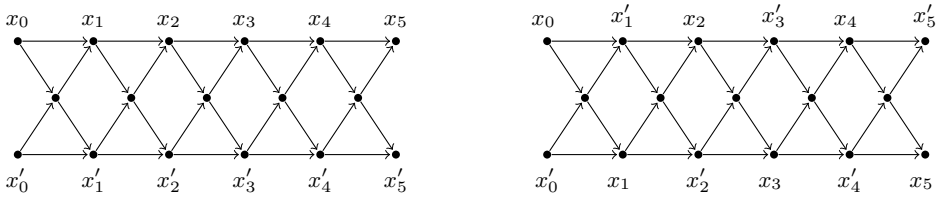


Figure 7 Example of the construction in Lemma 16 for the view defined in Example 14.

x_i with x'_i for all odd i without altering the view. This defines a new database D′, as shown in Figure 7. D and D′ agree on the view, but any path that goes from x_0 to x_n in D′ has to go at least once through one of the new paths, and is thus longer than n. Hence D and D′ disagree on Q, which concludes the proof.

This construction highlights the ideas in the general proof, for any $\mathbf{V}$ and Q. The fact that the new paths introduced from x_0 to x'_n in D are strictly longer than n actually stems from the assumption that G has no cycles of negative weights. The complete proof consists in giving an exact characterization of which new paths to add, and which nodes to switch between the two original paths. ◀

▶ **Lemma 17.** *Assume that there exists $G \in \mathcal{G}_{\mathbf{V}}$ such that 0 has both cycles of positive and negative weight, and that there is no path of weight 0 from 0 to o. Then, for all $n \equiv o[c]$, $\mathbf{V} \not\twoheadrightarrow \{n\}$.*

Sketch of proof. Remark that the conditions of this lemma are satisfied in Example 15, for $o = 1$. Let $\mathrm{Q} = \{n\}$, with $n \equiv 1[c]$. We illustrate the proof on this particular example.

Let W be the set of all cycles of 0 in G. Let $d = \gcd(W)$. Here, $d = 12$. Notice that, for each $i, j \in G$, there is a unique $w(i,j) \in \{0, \ldots, 11\}$ such that, for all path π from i to j, π is of weight $w(i,j)$ modulo d. For instance, all paths from 0 to 1 are of weight 4 modulo 12. A careful analysis of the arithmetic properties of G can actually show that this is true in general.

We build a counter-example showing that $\mathbf{V} \not\twoheadrightarrow \mathrm{Q}$ by using the information contained in G as follows. Start with a database D which is a cycle of length d whose nodes are $x_0, \ldots, x_{d-1}$. From D, we define a database D′ by renaming each node x_i of D to $x_{i+w(0,i[c])}$. For instance, x_0 remains as x_0, x_1 is renamed to $x_{1+w(0,1)}$, which is x_5, x_2 is renamed to x_{10}, and so on. See Figure 8.

Remark now that D′ is also a cycle of length d, whose nodes have simply been reordered, compared to D. Moreover, we can check that $\mathbf{V}(\mathrm{D}) = \mathbf{V}(\mathrm{D}')$. However, all paths of D of length 1 modulo c starting from x_0 end in one of x_1, x_4, x_7 or x_{10}, whereas for D′, they end in one of x_2, x_5, x_8 or x_{11}. This is due to the fact that $w(0,1) \neq 0$, as assumed by the lemma. Hence, D and D′ cannot agree on Q, which concludes this counter-example.

The fact that D′ is always a correctly-defined cycle of length d whose view image is equal to $\mathbf{V}(\mathrm{D})$ relies once again on the good arithmetic properties of G. ◀

4.3 Positive direction: computing a rewriting

In this section, we solve the positive case of Proposition 9. We start by giving a simple example that shows some of the features of the rewritings that will be used to prove Proposition 19.

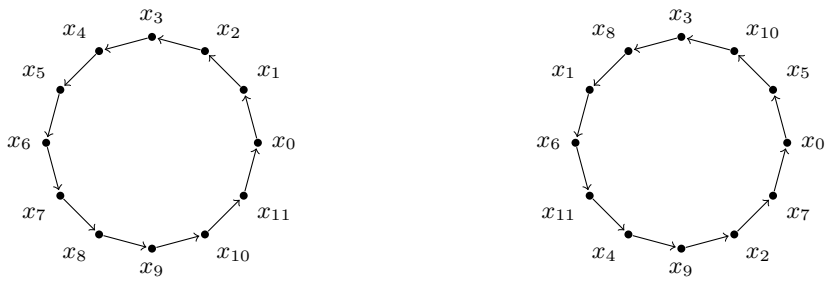


Figure 8 Example of the construction in Lemma 17 for the view defined in Example 15.

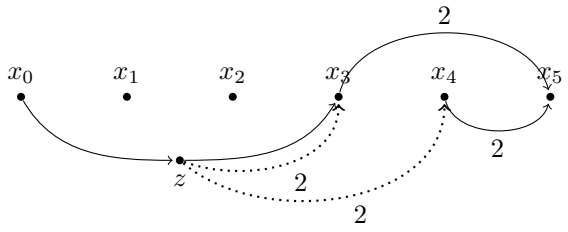


Figure 9 Illustration for the last case of Example 18. The full edges represent paths in the database, along with their length when it is more than 1. The dotted edges represent the two possible implications of $V_1(z, x_3)$.

▶ **Example 18.** In this example, we work with:

- $\mathbf{V} = \{C, V_1, V_2\}$
- $C = \{2\}$
- $V_1 = \{1, 2\}$
- $V_2 = \{2, 3\}$
- $\mathrm{Q} = \{5\}$

We show that $\mathbf{V} \twoheadrightarrow \mathrm{Q}$. Indeed, the following formula R is a rewriting of Q using $\mathbf{V}$.

$$\mathrm{R}(x, y) = \exists x_0, \ldots, x_5,\ x_0 = x \wedge x_5 = y \wedge CQ_{\pi_5} \wedge \Big(\forall z, V_1(z, x_3) \Rightarrow (C(z, x_3) \vee C(z, x_4))\Big)$$

where π_5 is a simple path whose nodes are $x_0, \ldots, x_5$ and CQ_{π_5} is the conjunctive query that states all the atoms that hold in $\mathbf{V}(\pi_5)$. First, remark that R only states necessary conditions for the existence of a path of length 5 from x to y, as explained in Section 4.1, hence, for all D, $\mathrm{Q}(\mathrm{D}) \subseteq \mathrm{R}(\mathbf{V}(\mathrm{D}))$.

Assume now that $(x, y) \in \mathrm{R}(\mathbf{V}(\mathrm{D}))$. Let $x_0, \ldots, x_5$ be a quantification for which $\mathrm{R}(x, y)$ is satisfied. We can prove the following:

- $C(x_0, x_2)$, $C(x_1, x_3)$ and $C(x_2, x_4)$ hold in $\mathbf{V}(\mathrm{D})$. Hence, these pairs of nodes are at distance 2 in D.
- $V_1(x_4, x_5)$ holds in $\mathbf{V}(\mathrm{D})$. Hence, x_4 and x_5 are either at distance 1 or 2. If this distance is 1, then we immediately get a path of length 5 from x_0 to x_5 by using the previous point, as $x_0 \xrightarrow{2} x_2 \xrightarrow{2} x_4 \xrightarrow{1} x_5$.
- Similarly, $V_2(x_0, x_3)$ holds in $\mathbf{V}(\mathrm{D})$. If the distance from x_0 to x_3 is 3, we immediately get $x_0 \xrightarrow{3} x_3 \xrightarrow{2} x_5$. Otherwise, there exists z such that $x_0 \to z \to x_3$. This implies $V_1(z, x_3)$.
- The remaining case is represented in Figure 9, with the two possible implications of $V_1(z, x_3)$ given by R. Both possibilities also imply a path of length 5 from x_0 to x_5.

▶ **Proposition 19.** *Assume that for all $G \in \mathcal{G}_{\mathbf{V}}$ there is a path of weight 0 from 0 to o. Then there exists $n \equiv o[c]$ such that $\mathbf{V} \twoheadrightarrow \{n\}$ and we can effectively compute a first-order rewriting that witnesses it.*

Sketch of proof. Let $\mathrm{Q} = \{n\}$, with $n \equiv o[c]$. We define a rewriting R as:

$$\mathrm{R}(x,y) = \exists x_0, \ldots, x_n,\ x_0 = x \wedge x_n = y \wedge \bigwedge_{i=1}^{3} R_i(x_0, \ldots, x_n)$$

where R_1, R_2 and R_3 are first-order formulas such that:

- R_1 states that $x_0, \ldots, x_n$ satisfy $\mathbf{V}(\pi)$ with $\pi = x_0 \ldots x_n$.
- R_2 states that, if $V(z, x_i)$ holds for some z, then $C^{\lfloor u/c \rfloor + 1}(z, x_{i+c-u[c]})$ must also hold for some $u \in V$, as explained in Figure 2.
- R_3 states that, if x_i and x_j are at distance $d \equiv 0[c]$, then x_{i-l} and x_{i+c-l} must be at distance $d + c$, for all $l \in \{0, \ldots, c-1\}$. As stated, R_3 is not in first-order because it is an infinite disjunction on the values of d. We can actually make it finite, but we omit the argument here for simplicity.

First, remark that $\mathrm{Q}(\mathrm{D}) \subseteq \mathrm{R}(\mathbf{V}(\mathrm{D}))$. Indeed, each R_i only states necessary conditions. The rest of this sketch is devoted to showing the converse.

Let D be a database such that $(x, y) \in \mathrm{R}(\mathbf{V}(\mathrm{D}))$. Hence, there exist nodes $x_0, \ldots, x_n$ in D such that $\mathrm{R}_i(x_0, \ldots, x_n)$ holds in $\mathbf{V}(\mathrm{D})$. Let π be a path in D from x_i to x_j, we denote the delay of this path by $\delta(\pi) = |\pi| - (j - i)$, that is the length of π minus the expected distance between x_i and x_j. For instance, assume that there is a path of length 2 from x_0 to x_1, then the delay of this path would be 1. By following this path, x_1 is too far from x_0 by 1, because x_1 was intended to be the successor of x_0.

From D, we build a graph H with $n + 1$ nodes such that there is an edge of label w between nodes i and j if and only if there is a path π from x_i to x_j with $\delta(\pi) = w$. Then we deduce from H a graph G by merging the nodes of H that have the same index modulo c. Remark that R_1, R_2 and R_3 imply that G actually contains a behavior graph G_0, as defined in Section 4.1. Thus, by hypothesis, this graph contains a path π_0 from 0 to o of weigth 0.

If n is big enough, then we can actually choose G_0 in such a way that each edge of G_0 has many witness paths in D. In other words, if G_0 contains an edge from i to j of weight w, then there exists many pairs of nodes $x_{i'}$ and $x_{j'}$ such that $i' \equiv i[c]$, $j' \equiv j[c]$ and there exists a path π from $x_{i'}$ to $x_{j'}$ with $\delta(\pi) = w$. With some additional combinatorial work, we show that we can mimic π_0 in D through the use of these witness paths, and thus produce a path of delay 0 from x_0 to x_n. Hence, this path is of length n, which implies that $(x_0, x_n) \in \mathrm{Q}(\mathrm{D})$ and concludes the proof. ◀

5 The case of small queries

In this section, we investigate the following example, in order to illustrate the issue that remains to be solved in the case of small queries.

- $\mathbf{V} = \{C, V_1, V_2\}$
- $C = \{2\}$
- $V_1 = \{1, 2\}$
- $V_2 = \{2, 5\}$

▶ **Claim 20.** *For all even n, $\boldsymbol{V} \twoheadrightarrow Q = \{n\}$. This easily comes from $C = \{2\}$.*

By applying Theorem 9 we can show that there exists some odd n such that $\mathbf{V} \twoheadrightarrow \mathrm{Q} = \{n\}$, hence $\mathbf{V}$ also determines all bigger queries. In order to get the full picture, we need to find the smallest odd n that is determined by $\mathbf{V}$. Our work so far actually gives us:

▶ **Claim 21.** *For all odd $n \leq 7$, $\boldsymbol{V} \not\twoheadrightarrow Q = \{n\}$.*

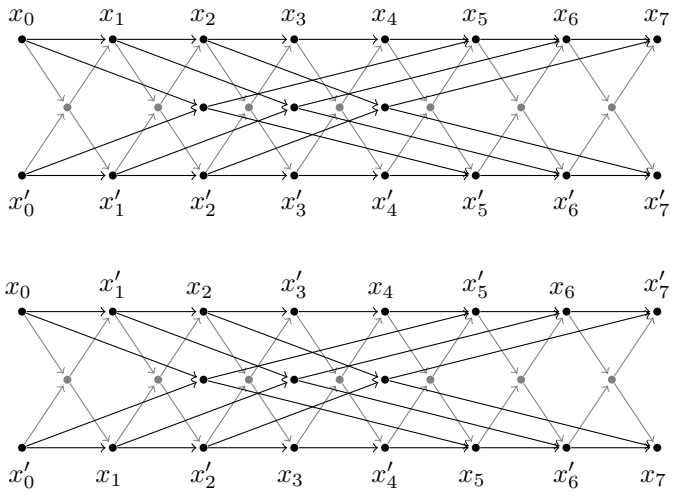


Figure 10 The two databases above are a proof that $\mathbf{V} \not\twoheadrightarrow \mathrm{Q} = \{n\}$ for any odd n that is not greater than 7. Indeed, we can check that both databases agree on $\mathbf{V}$. However, there is no path of length 1 (respectively 3, 5 and 7) from x_0 to x_1 (respectively x_3, x_5 and x_7) in the bottom database.

To prove this claim, we use a technique that is very similar to Lemma 16. More precisely, the two databases in Figure 10 agree on $\mathbf{V}$, but disagree on all $\mathrm{Q} = \{n\}$ when n is odd and not greater than 7.

Note that this technique does not work for n greater than 7. Indeed, in the case shown above, any path that goes from x_0 to x_7 in the bottom database has to cross from the top section to the bottom section. By doing so, it suffers a delay of either $+1$ or -3 compared to the expected value. It works here because 7 is "too small" and does not provide enough space to catch-up on this delay. Assume now that $n = 9$, then a delay of -3 can be mitigated by following a $+1$ path three times, and thus does not provide a counter-example.

▶ **Claim 22.** *For all $n \geq 11$, $\boldsymbol{V} \twoheadrightarrow Q = \{n\}$.*

We show this by arguing that $\mathbf{V} \twoheadrightarrow \mathrm{Q} = \{11\}$. This is done by actually proving that the canonical rewriting R given in Section 4.3 works in this case. Although the proof given in Section 4.3 does not apply (because 11 is not "big enough" for all the combinatorial arguments to go through), a careful enumeration of all the possibilities for a database satisfying R actually shows that $\mathrm{R}(x, y)$ implies a path of length 11 from x to y, as was done in Example 18.

It is then straightforward to prove that $\mathbf{V}$ determines every odd query bigger than 11. Let $n = 11 + 2k$ be such a query. Then a rewriting for n is simply $\mathrm{R}_{11} \cdot C^k$, as in Lemma 8. As we already know that $\mathbf{V}$ determines every even query, this end the proof of the claim.

The case of $n = 9$. There remains only a single unsolved case, which is $n = 9$. This qualifies as a "small query" for the view $\mathbf{V}$: a query for which we are unable to either build a generic counter-example, as in Section 4.2, or provide a generic rewriting, as in Section 4.3. We actually proved that $\mathbf{V} \not\twoheadrightarrow \mathrm{Q} = \{9\}$. However, the smallest counter-example that we know is a pair of databases of 154 nodes each, that were built by hand through a very tedious trial and error process and checked by a computer program. At this time, we are unfortunately unable to provide any technique to generate such a counter-example for other views and queries. We conjecture that the combinatorial complexity of these "small queries" might be way higher than what we have dealt with so far.

6 Conclusions

We have shown that, given a view $\mathbf{V}$ defined by unions of path queries, we can decide determinacy of almost all path queries Q. Although the smallest query that we can handle is of exponential size in the size of $\mathbf{V}$, our decision procedure still works with Π_2^{P} complexity. Moreover, for the queries that are big enough to be handled by our algorithm, we also provide a first-order rewriting when they are determined, and a canonical counter-example otherwise.

A natural continuation of this work would be to try and solve the determinacy problem even for small queries. Another possible continuation stems from the following remark: on all examples where $\mathbf{V}$ determines Q that we are aware of, it also turns out that our rewriting is actually correct, even when the query is too small to be handled by our technique. Perhaps it so happens that this rewriting is always correct, as soon as we assume that $\mathbf{V}$ determines Q. Failing that, it might still be the case that a first-order rewriting can always be found.

Acknowledgements. I gratefully thank Luc Segoufin and Cristina Sirangelo for carefully proofreading this paper and providing many invaluable comments and advices.

References

1 Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 254–263, 1998.

2 Foto N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, 412(11):1005–1021, 2011.

3 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Lossless regular views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 247–258. ACM, 2002.

4 Nadime Francis, Luc Segoufin, and Cristina Sirangelo. Datalog rewritings of regular path queries using views. In *Proceedings of the 17th International Conference on Database Theory (ICDT'14)*, pages 107–118, 2014.

5 Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 95–104, 1995.

6 Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Transactions on Database Systems*, 35(3), 2010.

Games for Active XML Revisited

Martin Schuster and Thomas Schwentick

TU Dortmund, Germany

Abstract

The paper studies the rewriting mechanisms for intensional documents in the Active XML framework, abstracted in the form of *active context-free games*. The *safe rewriting* problem studied in this paper is to decide whether the first player, JULIET, has a winning strategy for a given game and (nested) word; this corresponds to a successful rewriting strategy for a given intensional document. The paper examines several extensions to active context-free games.

The primary extension allows more expressive schemas (namely XML schemas and regular nested word languages) for both target and replacement languages and has the effect that games are played on nested words instead of (flat) words as in previous studies. Other extensions consider validation of input parameters of web services, and an alternative semantics based on insertion of service call results.

In general, the complexity of the safe rewriting problem is highly intractable (doubly exponential time), but the paper identifies interesting tractable cases.

1998 ACM Subject Classification F.2.m Miscellaneous, F.4.2 Grammars and Other Rewriting Systems, H.3.5 Online Information Services

Keywords and phrases Active XML, Computational Complexity, Nested Words, Rewriting Games, Semistructured Data

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.60

1 Introduction

Scientific context

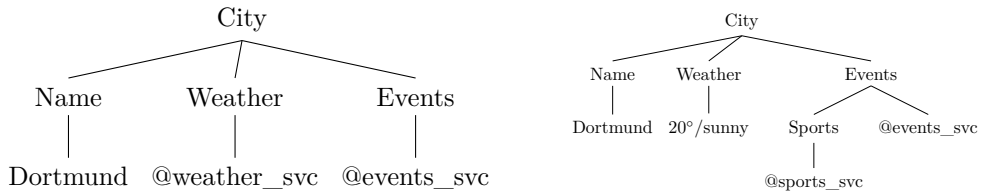
This paper contributes to the theoretical foundations of intensional documents, in the framework of *Active XML* [1]. It studies game-based abstractions of the mechanism transforming intensional documents into documents of a desired form by calling web services. One form of such games has been introduced under the name *active context-free games* in [11] as an abstraction of a problem studied in [9].[1] The setting in [9] is as follows: an *Active XML* document is given, where some elements consist of functions representing web services that can be called. The goal is to rewrite the document by a series of web service calls into a document matching a given *target schema*.

Towards an intuition of Active XML document rewriting, consider the example in Figure 1 of an online local news site dynamically loading information about weather and local events (adapted from [9] and [11]). Figure 1a shows the initial Active XML document for such a site, containing *function nodes* which refer to a weather and an event service, respectively, instead of concrete weather and event data. After a single function call to each of these services has been materialised, the resulting document may look like the one depicted in Figure 1b. Note that the rewritten document now contains new function nodes; further rewriting might be

[1] Actually, the two notions were introduced in the respective conference papers.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 60–75

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) Example document before rewriting.

(b) Same document after function calls.

Figure 1 Example of Active XML rewriting.

necessary to reach a document in a given target schema (which could, for instance, require that the document contains at least one indoor event if the weather is rainy).

Modelling this rewriting problem as a game follows the approach of dealing with uncertainty by playing a "game against nature": We model the process intended to rewrite a given document into a target schema by performing function calls as a player (Juliet). As her moves, she chooses which function nodes to call, and her goal is to reach a document in the target schema. Returns of function calls, on the other hand, are chosen (in accordance with some schema for each called service) by an antagonistic second player (Romeo), whose goal is to foil Juliet. The question whether a given document can always be rewritten into the target schema may then be solved by deciding whether Juliet has a winning strategy. More specifically, given an input document, target schema and return schemas for function calls, there should exist a *safe rewriting* algorithm that always rewrites the input document into the target schema, no matter the concrete returns of function calls, if and only if Juliet has a winning strategy in the corresponding game.[2]

In [9], the target schema is represented by an XML document type definition (DTD). It was argued that, due to the restricted nature of DTDs, the problem can be reduced to a rewriting game on strings where, in each move a single symbol is replaced by a string, the set of allowed replacement strings for each symbol is a regular language and the target language is regular[3], as well.

In [11] the complexity of the problem to determine the winner in such games (mainly with finite replacement languages) was studied. Whereas this problem is undecidable in general, there are important cases in which it can be solved, particularly if Juliet chooses the symbols to be replaced in a left-to-right fashion. In and after [11, 9], research very much concentrated on games on strings (and thus on the setting with DTDs). Furthermore, to achieve tractability, a special emphasis was given to the restriction to *bounded* strategies, in which the recursion depth with respect to web service calls is bounded by some constant.

Our approach

The aim of this paper is to broaden the scope and extend the investigation of games for Active XML in several aspects. First of all, we consider stronger schema languages (compared to DTDs) such as XML Schema and Relax NG, due to their practical importance. To allow for this extension, our games are played on nested words [3].[4]

[2] It is hard to give a precise statement of *safe rewriting* that does not already involve games, but we hope that the general idea of this statement becomes sufficiently clear.
[3] More precisely, it should be given by a *deterministic* regular expression.
[4] More precisely: word encodings of nested words in the sense of [3].

Table 1 Summary of complexity results. All results are completeness results.

	No replay	Bounded	Unbounded
Regular target language			
Regular replacement	PSPACE	2-EXPTIME	2-EXPTIME
Finite replacement	PSPACE	PSPACE	EXPTIME
DTD or XML Schema target language			
Regular replacement	PTIME	PSPACE	EXPTIME
Finite replacement	PTIME	PTIME	EXPTIME

Furthermore, we study the impact of the validation of input parameters for web service calls (partly considered already in [9]), and investigate an alternative semantics, where results of web service calls are inserted next to the node representing the web service, as opposed to replacing that node.

As we are particularly interested in the identification of tractable cases, we follow the previous line of research by concentrating on strategies in document order (left-to-right strategies) and by considering bounded strategies (*bounded replay*) and strategies in which no calls in results from previous web service calls are allowed (*no replay*). However, we also pinpoint the complexity of the general setting.

As a basic intuition for the concept of replay, consider again the online news site example from Figure 1, and assume that the schema for the event service's returns is (partially) given by @event_svc $\rightarrow$ (Sports|Movie)@event_svc, i.e. the event service allows for dynamic loading of additional results. A strategy with no replay would not be allowed to fetch any additional results in the situation of Figure 1b, while a strategy with bounded replay k (for some constant k) could load up to k more events after the first. A strategy with unbounded replay would be able to fetch an arbitrary number of results, but might lead to a rewriting process that does not terminate if unsuccessful.

Our contributions

Our complexity results with respect to stronger schema languages are summarised in Table 1. In the general setting, the complexity is very bad: doubly exponential time. However, there are tractable cases for XML Schema: replay-free strategies in general and strategies with bounded replay in the case of finite replacement languages (that is, when there are only finitely many possible answers, for each web service). It should be noted that the PSPACE-hardness result for the case with DTDs, bounded replay and infinite replacement languages indicates that the respective PTIME claim in [9] is wrong.

In the setting where web services come with an input schema that restricts the parameters of web service calls, we only study replay-free strategies. It turns out that this case is tractable if all schemas are specified by DTDs and the number of web services is bounded. On the other hand, if the desired document structure is specified by an XML Schema or the number of function symbols is unbounded, the task becomes PSPACE-hard.

For insertion-based semantics, we identify an undecidable setting and establish a correspondence with the standard "replacement" semantics, otherwise.

As a side result of independent interest, we show that the word problem for alternating nested word automata is PSPACE-complete.

Related work

We note that the results on flat strings in this paper do not directly follow from the results in [11], as [11] assumed target languages given by DFAs as opposed to *deterministic* regular expressions, which are integral to both DTDs and more expressive XML schema languages. However, the techniques from [11] can be adapted.

More related work for active context-free games than the papers mentioned so far is discussed in [11]. Further results on active context-free games in the "flat strings" setting can be found in [2, 4]. A different form of 2-player rewrite games are studied in [13]. More general *structure rewriting games* are defined in [7].

Organisation

We give basic definitions in Section 2. Games with regular schema languages (given by nested word automata) are studied in Section 3, games in which the schemas are given as DTDs or XML Schemas are investigated in Section 4. Validation of parameters and insertion of web service results are considered in Section 5. Due to space restrictions, most proofs are omitted here. An appendix containing the omitted proofs can be found in [12].

Acknowledgements

We would like to thank the anonymous reviewers for their insightful and constructive comments. We are grateful to Nils Vortmeier and Thomas Zeume for careful proof reading, and to Krystian Kensy for checking our proof of Proposition 18 (b) and for pinpointing the problems in the algorithm of [9] as part of his Master's thesis.

2 Preliminaries

For any natural number $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\}$. Where M is a (finite) set, $\mathcal{P}(M)$ denotes the powerset of M, i.e. the set of all subsets of M. For an alphabet Σ, we denote the set of finite strings over Σ by Σ^* and ϵ denotes the empty string.

Nested words

We use nested words[5] as an abstraction of XML documents [3]. For a finite alphabet Σ, $\langle\Sigma\rangle \stackrel{\text{def}}{=} \{\langle a\rangle \mid a \in \Sigma\}$ denotes the set of all *opening* Σ*-tags* and $\langle/\Sigma\rangle \stackrel{\text{def}}{=} \{\langle/a\rangle \mid a \in \Sigma\}$ the set of all *closing* Σ*-tags*. The set $\mathrm{WF}(\Sigma) \subseteq (\langle\Sigma\rangle \cup \langle/\Sigma\rangle)^*$ of *(well-)nested words over* Σ is the smallest set such that $\epsilon \in \mathrm{WF}(\Sigma)$, and if $u, v \in \mathrm{WF}(\Sigma)$ and $a \in \Sigma$, then also $u\langle a\rangle v\langle/a\rangle \in \mathrm{WF}(\Sigma)$. We (informally) associate with every nested word w its *canonical forest representation*, such that words $\langle a\rangle\langle/a\rangle$, $\langle a\rangle v\langle/a\rangle$ and uv correspond to an a-labelled leaf, a tree with root a (and subforest corresponding to v), and the forest of u followed by the forest of v, respectively. A nested string w is *rooted*, if its corresponding forest is a tree. In a nested string $w = w_1 \ldots w_n \in \mathrm{WF}(\Sigma)$, two tags $w_i \in \langle\Sigma\rangle$ and $w_j \in \langle/\Sigma\rangle$ with $i < j$ are *associated* if the substring $w_i \ldots w_j$ of w is rooted. To stress the distinction from nested strings in $\mathrm{WF}(\Sigma)$, we refer to strings in Σ^* as *flat strings* (over Σ).

What we describe as opening and closing tags is often referred to as *call symbols* and *return symbols* in the literature on nested words; we avoid these terms to avoid confusion with Read and Call moves used in context-free games (see below).

[5] Our definition of nested words corresponds to word encodings of well-matched nested words in [3].

Context-free games

A *context-free game on nested words (cfG)* $G = (\Sigma, \Gamma, R, T)$ consists[6] of a finite alphabet Σ, a set $\Gamma \subseteq \Sigma$ of *function symbols*, a *rule set* $R \subseteq \Gamma \times \mathrm{WF}(\Sigma)$ and a *target language* $T \subseteq \mathrm{WF}(\Sigma)$. We will only consider the case where T and, for each symbol $a \in \Gamma$, the set $R_a \stackrel{\text{def}}{=} \{u \mid (a, u) \in R\}$ is a non-empty regular nested word language, to be defined in the next subsection.

A play of G is played by two players, Juliet and Romeo, on a word $w \in \mathrm{WF}(\Sigma)$. In a nutshell, Juliet moves the focus along w in a left-to-right manner and decides, for every closing tag[7] $\langle/a\rangle$ whether she plays a *Read* or, in case $a \in \Gamma$, a *Call* move. In the latter case, Romeo then replaces the rooted word ending at the position of $\langle/a\rangle$ with some word $v \in R_a$ and the focus is set on the first symbol of v. In case of a Read move (or an opening tag) the focus just moves further on. Juliet wins a play if the word obtained at its end is in T.

Towards a formal definition, a *configuration* is a tuple $\kappa = (p, u, v) \in \{\mathrm{J}, \mathrm{R}\} \times (\langle\Sigma\rangle \cup \langle/\Sigma\rangle)^* \times (\langle\Sigma\rangle \cup \langle/\Sigma\rangle)^*$ where p is the player to move, $uv \in \mathrm{WF}(\Sigma)$ is the *current word*, and the first symbol of v is *the current position*. A *winning configuration* for Juliet is a configuration $\kappa = (\mathrm{J}, u, \epsilon)$ with $u \in T$. The configuration $\kappa' = (p', u', v')$ is a *successor configuration* of $\kappa = (p, u, v)$ (Notation: $\kappa \to \kappa'$) if one of the following holds:

(1) $p' = p = \mathrm{J}$, $u' = us$, and $sv' = v$ for some $s \in \langle\Sigma\rangle \cup \langle/\Sigma\rangle$ (Juliet plays Read);
(2) $p = \mathrm{J}$, $p' = \mathrm{R}$, $u = u'$, $v = v' = \langle/a\rangle z$ for $z \in (\langle\Sigma\rangle \cup \langle/\Sigma\rangle)^*$, $a \in \Gamma$, (Juliet plays Call);
(3) $p = \mathrm{R}$, $p' = \mathrm{J}$, $u = x\langle a\rangle y$, $v = \langle/a\rangle z$ for $x, z \in (\langle\Sigma\rangle \cup \langle/\Sigma\rangle)^*$, $y \in \mathrm{WF}(\Sigma)$, $u' = x$ and $v' = y'z$ for some $y' \in R_a$ (Romeo plays y');[8]

The *initial configuration* of game G for string w is $\kappa_0(w) \stackrel{\text{def}}{=} (\mathrm{J}, \epsilon, w)$. A *play* of G is either an infinite sequence $\Pi = \kappa_0, \kappa_1, \ldots$ or a finite sequence $\Pi = \kappa_0, \kappa_1, \ldots, \kappa_k$ of configurations, where, for each $i > 0$, $\kappa_{i-1} \to \kappa_i$ and, in the finite case, κ_k has no successor configuration. In the latter case, Juliet *wins* the play if κ_k is of the form $(\mathrm{J}, u, \epsilon)$ with $u \in T$, in all other cases, Romeo wins.

Strategies

A *strategy* for player $p \in \{\mathrm{J}, \mathrm{R}\}$ maps prefixes $\kappa_0, \kappa_1, \ldots, \kappa_k$ of plays, where κ_k is a p-configuration, to allowed moves. We denote strategies for Juliet by $\sigma, \sigma', \sigma_1, \ldots$ and strategies for Romeo by $\tau, \tau', \tau_1, \ldots$

A strategy σ is *memoryless* if, for every prefix $\kappa_0, \kappa_1, \ldots, \kappa_k$ of a play, the selected move $\sigma(\kappa_0, \kappa_1, \ldots, \kappa_k)$ only depends on κ_k. As context-free games are reachability games we only need to consider memoryless games; see, e.g., [6].

▶ **Proposition 1.** *Let G be a context-free game, and w a string. Then either* Juliet *or* Romeo *has a winning strategy on w, which is actually memoryless.*

---

[6] Some of the following definitions are taken from [4].

[7] It is easy to see that the winning chances of the game do not change if we allow Juliet to play Call moves at opening tags: if Juliet wants to play Call at an opening tag she can simply play Read until the focus reaches the corresponding closing tag and play Call then. On the other hand, if she can win a game by calling a closing tag, she can also win it by calling the corresponding opening tag, thanks to the fact that she has full information.

[8] We note that a Call move on $\langle/a\rangle$ in a substring of the form $\langle a\rangle y\langle/a\rangle$ actually deletes the substring y along with the opening and closing a-tags. This is consistent with the AXML intuition of the subtree rooted at a function node getting replaced when the function node is called.

Therefore, in the following, strategies σ for Juliet map configurations κ to moves $\sigma(\kappa) \in \{\text{Call}, \text{Read}\}$ and strategies τ for Romeo map configurations κ to moves $\tau(\kappa) \in \text{WF}(\Sigma)$.

For configurations κ, κ' and strategies σ, τ we write $\kappa \xrightarrow{\sigma,\tau} \kappa'$ if κ' is the unique successor configuration of κ determined by strategies σ and τ. Given an initial word w and strategies σ, τ the play[9] $\Pi(\sigma, \tau, w) \stackrel{\text{def}}{=} \kappa_0(w) \xrightarrow{\sigma,\tau} \kappa_1 \xrightarrow{\sigma,\tau} \cdots$ is uniquely determined. If $\Pi(\sigma, \tau, w)$ is finite, we denote the word represented by its final configuration by $\text{word}_G(w, \sigma, \tau)$.

A strategy σ for Juliet is *finite* on string w if the play $\Pi(\sigma, \tau, w)$ is finite for every strategy τ of Romeo. It is a *winning strategy* on w if Juliet wins the play $\Pi(\sigma, \tau, w)$, for every τ of Romeo. A strategy τ for Romeo is a *winning strategy* for w if Romeo wins $\Pi(\sigma, \tau, w)$, for every strategy σ of Juliet. We only consider finite strategies for Juliet, due to Juliet's winning condition. We denote the set of all finite strategies for Juliet in the game G by $\text{STRAT}_\text{J}(G)$, and the set of all strategies for Romeo by $\text{STRAT}_\text{R}(G)$.

The *Call depth* of a play Π is the maximum nesting depth of Call moves in Π, if this maximum exists. That is, the Call depth of a play is zero, if no Call is played at all, and one, if no Call is played inside a string yielded by a replacement move. For a strategy σ of Juliet and a string $w \in \text{WF}(\Sigma)$, the *Call depth* $\text{Depth}^G(\sigma, w)$ of σ on w is the maximum Call depth in any play $\Pi(\sigma, \tau, w)$. A strategy σ has *k-bounded Call depth* if $\text{Depth}^G(\sigma, w) \leq k$ for all $w \in \text{WF}(\Sigma)$. We denote by $\text{STRAT}^k_\text{J}(G)$ the set of all strategies with k-bounded Call depth for Juliet on G. As a more intuitive formulation, we use the concept of *replay*, which is defined as Call depth (if it exists) minus one: Strategies for Juliet of Call depth one are called *replay-free*, and strategies of k-bounded Call depth, for any k, have *bounded replay*. For technical reasons, we need to use Call depth for some formal proofs and definitions, but we will stick with the more intuitive concept of replay wherever possible.

By $\text{JWin}(G)$ we denote the set of all words for which Juliet has a winning strategy in $\text{STRAT}_\text{J}(G)$ (likewise for $\text{JWin}^k(G)$ and $\text{STRAT}^k_\text{J}(G)$).

Nested word automata

A *nested word automaton (NWA)* $A = (Q, \Sigma, \delta, q_0, F)$ [3] is basically a pushdown automaton which performs a push operation on every opening tag and a pop operation on every closing tag, and in which the pushdown symbols are just states. More formally, A consists of a set Q of *states*, an alphabet Σ, a *transition function* δ, an *initial state* $q_0 \in Q$ and a set $F \subseteq Q$ of *accepting states*. The function δ is the union of a function $(Q \times \langle\Sigma\rangle) \to \mathcal{P}(Q \times Q)$ and a function $(Q \times Q \times \langle/\Sigma\rangle) \to \mathcal{P}(Q)$.

A *configuration* κ of A is a tuple $(q, \alpha) \in Q \times Q^*$, with a *linear state* q and a sequence α of *hierarchical states*, reflecting the pushdown store. A *run of A on $w = w_1 \dots w_n \in WF(\Sigma)$* is a sequence $\kappa_0, \dots, \kappa_n$ of configurations $\kappa_i = (q_i, \alpha_i)$ of A such that for each $i \in [n]$ and $a \in \Sigma$ it holds that

- if $w_i = \langle a\rangle$, $(q_i, p) \in \delta(q_{i-1}, \langle a\rangle)$ (for some $p \in Q$), and $\alpha_i = p\alpha_{i-1}$, or
- if $w_i = \langle/a\rangle$, $q_i \in \delta(q_{i-1}, p, \langle/a\rangle)$ (for some $p \in Q$), and $p\alpha_i = \alpha_{i-1}$.

In this case, we also write $\kappa_0 \overset{w}{\rightsquigarrow}_A \kappa_n$. We say that A *accepts* w if $(q_0, \epsilon) \overset{w}{\rightsquigarrow}_A (q', \epsilon)$ for some $q' \in F$. The language $L(A) \subseteq \text{WF}(\Sigma)$ is defined as the set of all strings accepted by A and is called a *regular language* (of nested words).

An NWA is *deterministic* (or DNWA) if $|\delta(q, \langle a\rangle)| = 1 = |\delta(q, p, \langle/a\rangle)|$ for all $p, q \in Q$ and $a \in \Sigma$. In this case, we simply write $\delta(q, \langle a\rangle) = (q', p')$ instead of $\delta(q, \langle a\rangle) = \{(q', p')\}$

[9] As the underlying game G will always be clear from the context, our notation does not mention G explicitly.

(and accordingly for $\delta(q, p, \langle/a\rangle)$), and $\delta^*(p, w) = q$ if q is the unique state, for which $(p, \epsilon) \overset{w}{\rightsquigarrow}_A (q, \epsilon)$.

An NWA is in *normal form* if every transition function $\delta(p, \langle a\rangle)$ only uses pairs of the form (q, p). Informally, when A reads an opening tag it always pushes its current state (before the opening tag) and therefore can see this state when it reads the corresponding closing tag. As in this case the hierarchical state is just the origin state p of the transition, we write $\delta(p, \langle a\rangle) = q$ as an abbreviation of $\delta(p, \langle a\rangle) = (q, p)$, for DNWAs in normal form.

▶ **Lemma 2.** *There is a polynomial-time algorithm that computes for every deterministic NWA an equivalent deterministic NWA in normal form.*

Algorithmic Problems

In this paper, we study the following algorithmic problem JWin($\mathcal{G}$) for various classes $\mathcal{G}$ of context-free games.

JWin($\mathcal{G}$)	
Given:	A context-free game $G \in \mathcal{G}$ and a string w.
Question:	Is $w \in \text{JWin}(G)$?

A class $\mathcal{G}$ of context-free games in JWin($\mathcal{G}$) comes with three parameters:

- the representation of the target language T,
- the representation of the replacement languages R_a, and
- to which extent replay is restricted.

It is a fair assumption that the representations of the target language and the replacement languages are of the same kind, but we will always discuss the impact of the replacement language representations separately. In our most general setting, investigated in Section 3, target languages are represented by deterministic nested word automata, and replacement languages by (not necessarily deterministic) nested word automata. We do not consider the representation of target languages by non-deterministic NWAs, as (1) already for DNWAs the complexity is very high in general, and (2) we can show that even in the replay-free case the complexity would become EXPTIME-complete. We usually denote the automata representing the target and replacement languages by $A(T)$ and $A(R_a)$, respectively.

In Section 4 we study the cases where T is given as an XML Schema or a DTD. In each setting, we consider the cases of unrestricted replay, bounded replay (Call depth k, for some k), and no replay (Call depth 1). We note that replay depth is formally not an actual game parameter, but the algorithmic problem can be restricted to strategies of Juliet of the stated kind.

If the class $\mathcal{G}$ of games is clear from the context, we often simply write JWin instead of JWin($\mathcal{G}$).

We denote by $|R|$ the combined size of all $A(R_a)$, $a \in \Gamma$, and by $|G|$ the size of (a sensible representation of) G, i.e. $|G| = |\Sigma| + |R| + |A(T)|$.

3 Games with regular target languages

We first consider our most general case, where target languages are given by DNWAs, replacement languages by NWAs and replay is unrestricted, because the algorithm that we develop for this case can be adapted (and sped up) for many of the more restricted cases. It is important to note that our results do not *rely* on the presentation of schemas as nested word automata. In fact, in Section 4, we will assume that the target schema is given as an XML

Schema or a DTD. However, for our algorithms nested word automata are handy to represent (linearisations of) regular tree languages and therefore in *this* section target languages are represented by NWAs. We emphasize that deterministic bottom-up tree automata can be translated into deterministic NWAs in polynomial time [3].

This generic algorithm works in two main stages for a given cfG G and word w. It first analyses the game G and aggregates all necessary information in a so-called *call effect* C. Then it uses C to decide whether Juliet has a winning strategy in the game G on w.

The call effect C only depends on G and contains, for every function symbol f and every state q of the $A(T)$, all possible effects of the subgame starting with a Call move of Juliet on some symbol $\langle/f\rangle$ on the target language T, under the assumption that the sub-computation of $A(T)$ on the word yielded by the game from $\langle/f\rangle$ starts in state q. More precisely, it summarises which sets S of states Juliet can enforce by some strategy σ, where each S is a set of states of $A(T)$ that Romeo might enforce with a counter strategy against σ.

The first stage of the algorithm consists of an inductive computation in which successive approximations $C^1, C^2, \ldots$ of C are computed, where C^i is the restriction of C to strategies of Juliet of Call depth i. The size of call effects and the number of iterations are at most exponential in $|G|$. However, the first stage can not be performed in exponential time as a single iteration might take doubly exponential time in $|G|$. It turns out through our corresponding lower bound that single iterations can not be done faster.

At the end of the first stage, the algorithm computes an alternating NWA A_G (of exponential size) from C that decides the set $\mathrm{JWin}(G)$. In the second stage, A_G is evaluated on w, taking at most polynomial space in $|A_G|$ and $|w|$.

A restriction of games to bounded replay does not improve the general complexity of the problem, as this is dominated by the doubly exponential effort of a single iteration. However, for replay-free games, no iterations are needed, the initial call effect C^1 is of polynomial size and can easily be computed and therefore, in this case, the overall complexity is dominated by the second stage, yielding a polynomial-space algorithm.

Altogether we prove the following theorem in this section.

▶ **Theorem 3.** *For the class of unrestricted games* $\textsc{JWin}(\mathcal{G})$ *is*
(a) 2-EXPTIME-*complete with unbounded replay,*
(b) 2-EXPTIME-*complete with bounded replay, and*
(c) PSPACE-*complete without replay.*

The rest of this section gives a proof sketch for Theorem 3.

Before we describe the generic algorithm in more detail, we discuss the very natural and more direct approach by alternating algorithms, in which a strategy for Juliet is nondeterministically guessed and the possible moves of Romeo are taken care of by universal branching. In our setting of context-free games, there are the following obstacles to this approach: (1) Romeo can, in general, choose from an infinite number of (and thus arbitrarily long) strings in R_a, for the current a, and (2) it is not a priori clear that such algorithms terminate on all branches. Whereas the latter obstacle is not too serious (if Juliet has a winning strategy, termination on all branches is guaranteed), the former requires a more refined approach. We basically deal with it in two ways: in some cases it is possible to show that it does not help Romeo to choose strings of length beyond some bound; in the remaining cases (in particular in those cases considered in this section), the algorithms use abstracted moves instead of the actual replacement moves of the game. The two stages that were sketched above, then come very naturally: first, the abstraction has to be computed, then it can be used for the actual alternating computation.

Our abstraction from actual cfGs is based on the simple observation that instead of knowing the final word $\mathrm{word}_G(w,\sigma,\tau)$ that is reached in a play $\Pi(\sigma,\tau,w)$, it suffices to know whether $\delta^*(q_0,\mathrm{word}_G(w,\sigma,\tau)) \in F$ to tell the winner. If we fix a strategy σ of JULIET in a game on w, the possible outcomes of the game (for the different strategies of ROMEO) can thus be summarised by $\mathrm{states}_G(q_0,w,\sigma) \stackrel{\text{def}}{=} \{\delta^*(q_0,\mathrm{word}_G(w,\sigma,\tau)) \mid \tau \in \mathrm{STRAT}_{\mathrm{R}}(G)\}$.

To this end, it will be particularly useful to study the (abstractions of) possible outcomes of *subgames* that start from a Call move on some tag $\langle/a\rangle$ until the focus moves to the symbol after $\langle/a\rangle$.

▶ **Definition 4.** For a cfG $G = (\Sigma,\Gamma,R,T)$ with a deterministic target NWA $A(T) = (Q,\Sigma,\delta,q_0,F)$, the *call effect* $\mathcal{C}[G] : \Gamma \times Q \to \mathcal{P}(\mathcal{P}(Q))$ is defined, for every $a \in \Gamma$, $q \in Q$, by

$$\mathcal{C}[G](a,q) \stackrel{\text{def}}{=} \left[\{\mathrm{states}_G(q,\langle a\rangle\langle/a\rangle,\sigma) \mid \sigma \in \mathrm{STRAT}_{\mathrm{J,Call}}(G)\}\right]_{\min},$$

where $\mathrm{STRAT}_{\mathrm{J,Call}}(G)$ contains all strategies of JULIET that start by playing Read on $\langle a\rangle$ and Call on $\langle/a\rangle$, and the operator $[\cdot]_{\min}$ removes all non-minimal sets from a set of sets.

We next describe how to compute $\mathcal{C}[G]$ from a given cfG G. As already mentioned, our algorithm follows a fixpoint-based approach. It computes inductively, for $k = 1,2,\ldots$ the call effect of the restricted game of maximum Call depth k. We show that the fixpoint reached by this process is the actual call effect $\mathcal{C}[G]$.

To this end, let, for every cfG G, $a \in \Sigma$, $q \in Q$, and $k \geq 1$,

$$\mathcal{C}^k[G](a,q) \stackrel{\text{def}}{=} \left[\{\mathrm{states}_G(q,\langle a\rangle\langle/a\rangle,\sigma) \mid \sigma \in \mathrm{STRAT}^k_{\mathrm{J,Call}}(G)\}\right]_{\min}.$$

As an important special case, the call effect of replay-free games – the basis for the inductive computation – consists of only one set.

▶ **Lemma 5.** *For every $q \in Q$ and $a \in \Sigma$, it holds that*

$$\mathcal{C}^1[G](a,q) = \{\{\delta^*(q,v) \mid v \in R_a\}\}.$$

In particular, $\mathcal{C}^1[G]$ can be computed from G in polynomial time.

This just follows from the definitions, as ROMEO can choose any string from R_a.

We next describe how each $\mathcal{C}^{k+1}[G]$ can be computed from $\mathcal{C}^k[G]$. The algorithm uses alternating nested word automata (ANWAs) which we will now define.

An *alternating nested word automaton (ANWA)* $A = (Q,\Sigma,\delta,q_0,F)$ is defined like an NWA, except that the two parts of δ map $(Q \times \langle\Sigma\rangle)$ into $\mathcal{B}^+(Q \times Q)$ and $(Q \times Q \times \langle/\Sigma\rangle)$ into $\mathcal{B}^+(Q)$, respectively, where $\mathcal{B}^+(Q)$ denotes the set of all positive boolean combinations over elements of Q using the binary operators $\wedge$ and $\vee$ (and likewise for $\mathcal{B}^+(Q \times Q)$).

The semantics of ANWA is defined via *runs*, which require the notion of *tree domains*. A tree domain is a prefix-closed language $D \subseteq \mathbb{N}^*$ of words over $\mathbb{N}$ such that, if $wk \in D$ for some $w \in D, k \in \mathbb{N}$, then also $wj \in D$ for all $j < k$. Strings in a tree domain are interpreted as node addresses for ordered trees in the standard way: ϵ addresses the root, and if $w \in D$ addresses some node v with k children, then $w1,\ldots,wk \in D$ address those children.

For any function $\lambda : D \to (Q \cup (Q \times Q))$ and node address $x \in D$, we denote by $\overline{\lambda(x)}$ the linear state component of $\lambda(x)$, i.e. if $\lambda(x) = q$ or $\lambda(x) = (q,p)$ for some $p,q \in Q$, then $\overline{\lambda(x)} = q$.

A *run* $r = (D,\lambda)$ of an ANWA A over a nested word $w = w_1 \ldots w_n$ is a finite tree of depth n, represented by a tree domain D and a labelling function $\lambda : D \to (Q \cup (Q \times Q))$ such that $\lambda(\epsilon) = q_0$ and, for every $x \in D$ of length i with ℓ children, it holds that

- if $w_{i+1} \in \langle \Sigma \rangle$, then $\{\lambda(x \cdot 1), \ldots, \lambda(x \cdot \ell)\} \models \delta(\overline{\lambda(x)}, w_{i+1})$, and
- if $w_{i+1} \in \langle / \Sigma \rangle$ with associated opening tag w_j, and $\lambda(y) = (q, p)$ for some $p, q \in Q$ (where y is the prefix of x of length j), then $\{\lambda(x \cdot 1), \ldots, \lambda(x \cdot \ell)\} \models \delta(\overline{\lambda(x)}, p, w_{i+1})$.

An ANWA A *accepts* a nested word w if there is a run (D, λ) over w such that $\lambda(x) \in F$, for every $x \in D$ of length $|w|$.

ANWAs are used twice in the generic algorithm, first, to inductively compute $\mathcal{C}^{k+1}[G]$ from $\mathcal{C}^k[G]$, second to actually decide $\mathrm{JWin}(G)$, given $\mathcal{C}[G]$. The following proposition will be crucial, in both cases.

▶ **Proposition 6.** *There is an algorithm that computes from the call effect $\mathcal{C}[G]$ of a game G in polynomial time in $|\mathcal{C}[G]|$ and $|G|$ an ANWA $A_{\mathcal{C}[G]}$ such that $L(A_{\mathcal{C}[G]}) = JWin(G)$.*

The computation of $\mathcal{C}^{k+1}[G]$ from $\mathcal{C}^k[G]$ involves a non-emptiness test for ANWAs, the second stage a test whether $w \in L(A_{\mathcal{C}[G]})$. Therefore, both of the following complexity results for ANWAs influence the complexity of our algorithms.

▶ **Proposition 7.**
(a) *Non-emptiness for ANWAs is* 2-EXPTIME*-complete.*
(b) *The membership problem for ANWAs is* PSPACE*-complete.*

Statement (a) follows immediately from the corresponding result for visibly pushdown automata in [5], statement (b) is new, to the best of our knowledge, and seems to be interesting in its own right. It is shown in [12].

Now we continue describing the ingredients of the first stage of the generic algorithm.

▶ **Lemma 8.** *Given a state $q \in Q$, an alphabet symbol $a \in \Gamma$, and $\mathcal{C}^k[G]$, for some $k \geq 1$, the call effect $\mathcal{C}^{k+1}[G](a, q)$ can be computed in doubly exponential time in $|G|$.*

By Lemmas 5 and 8, one can compute $\mathcal{C}^k[G]$ inductively, for every $k \geq 1$. By definition it holds, for every q and a, that $\mathcal{C}^k[G](a, q)$ is contained in the closure of $\mathcal{C}^{k+1}[G](q, a)$ under supersets. As there are $\leq 2^{|Q|}$ sets in each $\mathcal{C}^k[G](a, q)$ (for $a \in \Gamma, q \in Q$), the computation reaches a fixed point after at most exponentially many iterations. We denote this fixed point by $\mathcal{C}^*[G]$, that is, we define, for every $a \in \Sigma$, $q \in Q$:

$$\mathcal{C}^*[G](a, q) \stackrel{\text{def}}{=} \left[\bigcup_{k=1}^{\infty} \mathcal{C}^k[G](a, q). \right]_{\min}$$

In particular, for each game G, there is a number $\ell \leq |\Gamma| \times |Q| \times 2^{|Q|}$ such that $\mathcal{C}^*[G] = \mathcal{C}^\ell[G]$ and $\mathcal{C}^m[G] = \mathcal{C}^\ell[G]$, for every $m \geq \ell$. However, it is not self evident that this process actually constructs $\mathcal{C}[G]$, i.e., that $\mathcal{C}^*[G] = \mathcal{C}[G]$. The following result shows that this is actually the case.

▶ **Proposition 9.** *For every cfG G it holds: $\mathcal{C}^*[G] = \mathcal{C}[G]$.*

Now we can give a (high-level) proof for Theorem 3.

Proof of Theorem 3. We first justify the upper bounds. Let G be a cfG and w a word. By Lemma 5, $\mathcal{C}^1[G]$ can be computed in polynomial time from G. For the replay-free case, we can immediately construct an ANWA for $\mathrm{JWin}(G)$ and evaluate it on w, yielding a PSPACE upper bound by Proposition 7.

For (a) and (b), $\mathcal{C}[G]$ ($\mathcal{C}^k[G]$, respectively) can be computed in doubly exponential time, A_C can be computed in exponential time (in the size of G), and whether $w \in L(A_C)$ can

then be tested in polynomial space in $|A_C|$ and $|w|$, that is, in at most exponential space in $|G|$ and $|w|$.

That these upper bounds can not be considerably improved, is stated in the following proposition, thereby completing the proof of Theorem 3. ◀

▶ **Proposition 10.** *For the class of unrestricted games* JWin *is*
(a) 2-EXPTIME-*hard with bounded replay, and*
(b) PSPACE-*hard with no replay.*

Claims (a) and (b) of Proposition 10 follow from the corresponding parts of Proposition 7; in the proof, we construct from an ANWA A a replay-free cfG simulating A on any input word w (yielding claim (b)) and explain how replay can be added to that game to find and verify a witness for the non-emptiness of A, if one exists (yielding claim (a)).

For finite (and explicitly given) replacement languages the complexity changes considerably in the cases with replay, but not in the replay-free case.

▶ **Proposition 11.** *For the class of unrestricted games with finite replacement languages,* JWin($\mathcal{G}$) *is*
(a) EXPTIME-*complete with unbounded replay, and*
(b) PSPACE-*complete with bounded or without replay.*

The upper bound in (a) follows as for finite replacement languages $\mathcal{C}^{k+1}[G](a,q)$ can be computed from $\mathcal{C}^k[G](a,q)$ in polynomial space[10]. The PSPACE upper bound in (b) can then be achieved by the usual "recomputation technique" of space-bounded computations.

The lower bound in (a) already holds for flat words (see Theorem 4.3 in [11]). The lower bound in (b) follows as the proof of Proposition 10 only uses finite replacement languages.

As our algorithms generally construct ANWAs deciding JWin(G), the data complexity for JWin is in PSPACE for all cases considered in this section due to Proposition 7.

4 Games with XML Schema target languages

The results of Section 3 provide a solid foundation for our further studies, but the setting studied there suffers from two problems: (1) the complexities are far too high (at least for games with replay) and (2) the assumption that target and replacement languages are specified by (D)NWAs is not very realistic. In this section, we address both issues at the same time: when we require that target languages are specified by typical XML schema languages (DTD or XML Schema), we get considerably better complexities.

The better complexities basically all have the same reason: XML Schema target languages can be described by a restriction of nested word automata, which we call *simple* below. This restriction translates to the alternating NWAs corresponding to call effects. For simple ANWAs, however, the two basic algorithmic problems, Non-emptiness and Membership have dramatically better complexities: PSPACE and PTIME as opposed to 2-EXPTIME and PSPACE, respectively. We emphasise that, in accordance with the official standards, our definitions for DTDs and XML Schema require *deterministic* regular expressions.

Altogether, we prove the following complexity results.

▶ **Theorem 12.** *For classes of games with XML Schemas or DTDs, respectively,* JWin *is*

[10] It is worth noting that this upper bound even holds if the finite replacement language is not explicitly given, but represented by NWAs.

(a) EXPTIME-*complete for unbounded replay,*
(b) PSPACE-*complete for bounded replay, and*
(c) PTIME-*complete (under logspace-reductions) without replay.*

Here, the lower bounds are proven for DTDs, and the upper bounds for XML Schemas.

The lower bound in Theorem 12 (b) for the case of games with DTDs contradicts the statement of a PTIME algorithm in Section 4.3 of [9] (unless PTIME = PSPACE).[11]

Before we describe the proof of Theorem 12, we first define *single-type tree grammars* and *local tree grammars* as well-established abstractions of XML Schema and DTDs, respectively (see, e.g., [10]). However, we will refer to grammars of these types as XML Schemas and DTDs, respectively.

▶ **Definition 13.** A *(regular) tree grammar* is a tuple $T = (\Sigma, \Delta, S, P, \lambda)$, where
- Σ is a finite alphabet of *labels*,
- Δ is a finite alphabet of *types*,
- $S \in \Delta$ is the *root* or *starting type*,
- P is a set of *productions* of the form $X \to r_X$ mapping each type $X \in \Delta$ to a deterministic regular expression r_X over Δ, called the *content model* of X, and
- $\lambda : \Delta \to \Sigma$ is a *labelling function* assigning a label from Σ to each type in Δ.

T is *single-type* if for each $X \in \Delta$, the content model r_X contains no competing types, i.e. if r_X contains no two types $Y \neq Z$ with $\lambda(Y) = \lambda(Z)$. T is *local*, if it has exactly one type for every label.

We omit the definition of the formal semantics of regular tree grammars. The nested word language $L(T)$ described by T is just the set of linearisations of trees of the tree language that is defined in the standard way.

We next define *simple* DNWAs, a restriction of DNWAs that captures all languages specified by single-type tree grammars. In simple DNWAs, states are typed, i.e. each state has a component in some type alphabet Δ. Informally, when a simple DNWA A reads a subword $w = \langle a \rangle v \langle / a \rangle$ in state q, it determines already on reading $\langle a \rangle$ which state q' it will take after processing w, and this state will be of the same type as q. After reading $\langle a \rangle$, the linear state of A only depends on the *type* of q, not the exact state; this models the single-type restriction. After reading $\langle a \rangle$, A goes on to validate v, and if this validation fails, A enters a failure state $\perp$ instead of q'. Thus, the state of A at a position basically only depends on its ancestor positions (in the tree view of the document) and their left siblings. The only way in which other nodes in subtrees of these nodes can influence the state is by assuming the sink state $\perp$. Thus, in the spirit of [8], we could call such DNWAs *ancestor-sibling-based* but we prefer the term *simple* for simplicity.

▶ **Definition 14.** A deterministic NWA $A(T) = (Q, \Sigma, \delta, q_0, F)$ in normal form is *simple* (SNWA) if there exist a *type alphabet* Δ and *state set* P with $Q \subseteq P \times \Delta$, a *local acceptance function* $F_{\text{loc}} : \Sigma \to \mathcal{P}(Q)$, a *target state function* $t : Q \times \Sigma \to Q$ and a *failure state* $\perp \in Q \setminus F$, such that the following conditions are satisfied for every $a \in \Sigma$:
- for every $p, p' \in P, X \in \Delta$: $\delta((p, X), \langle a \rangle) = \delta((p', X), \langle a \rangle)$;
- for every $q \in F_{\text{loc}}(a)$: $\delta(q, p, \langle / a \rangle) = t(p, a)$;

[11] A close inspection of the construction in the proof in [9] reveals that the automaton constructed there does not deal correctly with the alternation between the choices of Romeo and Juliet. More precisely, the automaton allows Romeo to let the suffix of a replacement string depend on the choices of Juliet on its prefix.

- for every $q \in Q \setminus F_{\text{loc}}(a)$: $\delta(q, p, \langle/a\rangle) = \bot$ and
- for every $q \in Q$: $\delta(\bot, \langle a\rangle) = \delta(\bot, q, \langle/a\rangle) = \bot$.
- for every $(p, X) \in Q$: $t((p, X), a) = (p', X)$ for some $p' \in P$.

A cfG is called *simple* if its target DNWA is simple.

▶ **Proposition 15.** *From every single-type tree grammar T, a simple DNWA A can be computed in polynomial time, such that $L(A) = L(T)$.*

The following adaptation of the notion of *simplicity* to ANWAs is a bit technical. It will guarantee however that the ANWAs obtained from simple games are simple and have reasonable complexity properties.

▶ **Definition 16.** An ANWA $A = (Q, \Sigma, \delta, q_0, F)$ with $Q \subseteq P \times \Delta$ (for some state set P and type alphabet Δ) is *simple* (SANWA), if it has the following two properties.

- *(Horizontal simplicity)* There are a *local acceptance function* $F_{\text{loc}} : \Sigma \to \mathcal{P}(Q)$, a *test state* $q_? \in Q$, and a *target state function* $t : Q \times \Sigma \to Q$, such that the transition function δ of A satisfies the following conditions:
  - $\delta(q, q', \langle/a\rangle) = t(q', a)$ for all $q \in Q$ and $q' \neq q_?$;
  - $\delta(q, q_?, \langle/a\rangle) = \begin{cases} \text{true}, & \text{if } q \in F_{\text{loc}}(a) \\ \text{false}, & \text{if } q \notin F_{\text{loc}}(a) \end{cases}$

  Furthermore, for each $(p, X) \in Q$ and $a \in \Sigma$, it holds that $t((p, X), a) = (p', X)$ for some $p' \in P$.
- *(Vertical Simplicity)* For each $X \in \Delta$ and $a \in \Sigma$, there is a $q \in Q$ such that for all $p \in P$ it holds that $\delta((p, X), \langle a\rangle) \in \mathcal{B}^+(\{q\} \times ((P \times \{X\}) \cup \{q_?\}))$.

Essentially, horizontal simplicity states that A has two kinds of computations on a well-nested subword: (1) computations starting from a pair $(q, q_?)$ test a property of the subword and can either succeed or fail at the end of the subword (and thus influence the overall computation); (2) computations starting from a pair (q, q') for $q' \neq q_?$ basically ignore the subword. Even though they may branch in an alternating fashion, the state after the closing tag $\langle/a\rangle$ is the same in all subruns, is determined by $t(q', a)$ and has the same type as q'.

Vertical simplicity, on the other hand, states that all alternation in A happens in the choice of hierarchical states – while, on an opening tag, A may branch into sub-runs pushing different hierarchical states onto the stack, the choice of linear follow-up state is "locally deterministic", depending only the type of the previous state of A and the label of the tag being read, and the current type is preserved in all hierarchical states except for $q_?$. Together, these two conditions also guarantee that SNWAs may also be interpreted as SANWAs.

▶ **Proposition 17.**

(a) *Non-emptiness for SANWA is* PSPACE*-complete.*

(b) *The membership problem for SANWA is decidable in polynomial time.*

Proof of Theorem 12. The generic algorithm from the previous section can be adapted for simple cfGs, but with better complexity thanks to Proposition 17, to yield the upper bounds stated in Theorem 12.

More precisely, Proposition 17 (b) and Lemma 5 yield a polynomial time bound for replay-free games. Proposition 17 (a) guarantees that the inductive step in the computation of $\mathcal{C}[G]$ can be carried out in polynomial space (as opposed to doubly exponential time).[12]

[12] We actually use a slightly stronger result than Proposition 17 (a): deciding whether, for an NWA A_1 and a SANWA A_2, it holds $L(A_1) \cap L(A_2) \neq \emptyset$, is complete for PSPACE.

The upper bounds for games with unrestricted replay follows immediately and the upper bound for bounded replay can be shown similarly as in Proposition 11 (b).

The lower bounds are given by the following proposition. They mostly follow from careful adaptation of lower bound proofs of [11] for games on flat strings. ◀

▶ **Proposition 18.** *For the class of games with target languages specified by DTDs,* JWin *is*
(a) EXPTIME*-hard with unrestricted replay,*
(b) PSPACE*-hard with bounded replay, and*
(c) PTIME*-hard (under logspace-reductions) without replay*

For finite (and explicitly given) replacement languages we get feasibility even for bounded replay, but no improvement for unbounded replay.

▶ **Proposition 19.** *For the class of games with target languages specified by XML Schemas and explicitly enumerated finite replacement languages,* JWin *is*
(a) EXPTIME*-complete with unrestricted replay, and*
(b) PTIME*-complete (under logspace-reductions) with bounded replay or without replay.*
The same results hold for DTDs in place of XML Schemas.

Once again, as our algorithm generally computes a SANWA deciding JWin(G), the data complexity for JWin is in PTIME for all cases considered here, due to Proposition 17.

5 Validation of parameters and Insertion

In this section, we focus on two features that have not been addressed in the previous two sections: validation of the parameters of a function call with respect to a given schema, and a semantics which allows that returned trees do not *replace* their call nodes but are inserted next to them.

5.1 Validation of parameters

As pointed out in [9], in Active XML, parameters of function calls should be valid with respect to some schema. Transferred to the setting of cfGs this means that Juliet should only be able to play a Call move in a configuration $(\mathrm{J}, u\langle a\rangle v, \langle/a\rangle w)$ if $\langle a\rangle v\langle/a\rangle$ is in V_a for some set V_a of words that are valid for calls of $\langle/a\rangle$. Our definition of cfGs and the previous ones studied in the literature mostly ignore this aspect.[13] We do not investigate all possible game types in combination with parameter validation but rather concentrate on the most promising setting with respect to tractable algorithms. It turns out, that games without replay and with DTDs to specify target, replacement and validation languages have a tractable winning problem as long as the number of different validation DTDs is bounded by some constant.[14] It becomes intractable if the number of validation schemas can be unbounded and (already) with target and validation languages specified by XML Schemas, even with only one validation schema.

More precisely, we prove the following results.

[13] Actually, [9] takes validation into account but the precise way in which parameters are specified and tested is not explained in full detail.
[14] Note that this implies a polynomial-time data complexity for arbitrary replay-free games with DTD target, replacement and validation languages.

▶ **Theorem 20.** *For the class of games with validation with a bounded number of validation DTDs and target languages specified by DTDs,* JWin *is in* PTIME *without replay.*

The algorithm uses a bottom-up approach. The basic idea is that, starting from the leaves, at each level of the tree (that is for some node v and its leaf children) all relevant information about the game in the subtree t_v is computed with the help of flat replay-free games and aggregated in v. Then the children of v are discarded and the algorithm continues until only the root remains.

The following result shows that for slightly stronger games, parameter validation worsens the complexity.[15]

▶ **Theorem 21.** *For the class of games with validation,* JWin *(without replay) is*

(a) EXPTIME*-hard, if target and validation languages are specified by DNWAs (even with only one function symbol);*

(b) PSPACE*-hard, for games with only one function symbol, if the validation language is given by an XML schema, the target language by a DTD and a finite replacement language; and*

(c) PSPACE*-hard, for games with an unbounded number of validation DTDs and replacement and target languages specified by DTDs.*

Part (a) is proven by reduction from the intersection emptiness problem for DNWAs, while parts (b) and (c) use similar reductions from the problem of determining whether a quantified Boolean formula in disjunctive normal form is true.

Due to time constraints and as we are mainly interested in finding tractable cases, we have not looked for matching upper bounds.

5.2 Insertion rules

In our definition of Call moves, we define the successor configuration of a configuration $(\mathrm{R}, u\langle a\rangle v, \langle/a\rangle w)$ to be $(\mathrm{J}, u, v'w)$, that is, $\langle a\rangle v\langle/a\rangle$ is *replaced* by a string $v' \in R_a$. However, Active XML also offers an "append" option, where results of function calls are inserted as siblings after the calling function node (cf. [1]). There are (at least) three possible semantics of a Call move for insertion (as opposed to replacement) based games: the next configuration could be (1) $(\mathrm{J}, u, \langle a\rangle v\langle/a\rangle v'w)$, (2) $(\mathrm{J}, u\langle a\rangle v\langle/a\rangle, v'w)$, or (3) $(\mathrm{J}, u\langle a\rangle v\langle/a\rangle v', w)$, depending on "how much replay" we allow for Juliet. We consider (1) as the general setting, (2) as the setting with *weak replay* and (3) as the setting *without replay.* It turns out that the weak replay setting basically corresponds to the (unrestricted) setting with replacement rules and that (3) corresponds to the replay-free setting with replacement rules. Setting (1), however, gives Juliet a lot of power and makes JWin$(\mathcal{G})$ undecidable.

▶ **Theorem 22.** *For the class of games with insertion semantics, target DNWAs and replacement NWAs,* JWin *is*

(a) *undecidable in general;*

(b) 2-EXPTIME*-complete for games with weak replay; and*

(c) *PSPACE-complete for games without replay.*

The proof idea for Theorem 22 is to simulate insertion-based games by replacement-based games and vice versa; part (a) additionally uses the undecidability of JWin for arbitrary (i.e. not necessarily left-to-right) strategies on games with flat strings, which was proven to be undecidable in [11].

[15] This is, of course not surprising. If any, the surprising result is Theorem 20.

6 Conclusion

The complexity of context-free games on nested words differs considerably from that on flat words (2-EXPTIME vs. EXPTIME), but there are still interesting tractable cases. One of the main insights of this paper is that the main tractable cases remain tractable if one allows XML Schema instead of DTDs for the specification of schemas.

Another result is that adding validation of input parameters can worsen the complexity, but tractability can be maintained by a careful choice of the setting. However, here the step from DTDs to XML Schema may considerably worsen the complexity.

Insertion semantics with unlimited replay yields undecidability.

We leave open some corresponding upper bounds in the setting with validation of input parameters. In future work, we plan to study the impact of parameters of function calls more thoroughly.

References

1 Serge Abiteboul, Omar Benjelloun, and Tova Milo. The Active XML project: an overview. *VLDB J.*, 17(5):1019–1040, 2008.

2 Serge Abiteboul, Tova Milo, and Omar Benjelloun. Regular rewriting of active XML and unambiguity. In *PODS*, pages 295–303, 2005.

3 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.

4 Henrik Björklund, Martin Schuster, Thomas Schwentick, and Joscha Kulbatzki. On optimum left-to-right strategies for active context-free games. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 105–116, 2013.

5 Laura Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In *CONCUR- Concurrency Theory, 18th International Conference*, pages 476–491, 2007.

6 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*. Springer, 2002.

7 Lukasz Kaiser. Synthesis for structure rewriting systems. In Rastislav Královic and Damian Niwinski, editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2009.

8 Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006.

9 Tova Milo, Serge Abiteboul, Bernd Amann, Omar Benjelloun, and Frederic Dang Ngoc. Exchanging intensional XML data. *ACM Trans. Database Syst.*, 30(1):1–40, 2005.

10 Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.

11 Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Active context-free games. *Theory Comput. Syst.*, 39(1):237–276, 2006.

12 Martin Schuster and Thomas Schwentick. Games for Active XML revisited. *CoRR*, abs/1412.5910, 2014. Available online at `http://arxiv.org/abs/1412.5910`.

13 Johannes Waldmann. Rewrite games. In Sophie Tison, editor, *RTA*, volume 2378 of *Lecture Notes in Computer Science*, pages 144–158. Springer, 2002.

Answering Conjunctive Queries with Inequalities*

Paraschos Koutris[1], Tova Milo[2], Sudeepa Roy[3], and Dan Suciu[4]

1 **University of Washington**
pkoutris@cs.washington.edu
2 **Tel Aviv University**
milo@cs.tau.ac.il
3 **University of Washington**
sudeepa@cs.washington.edu
4 **University of Washington**
suciu@cs.washington.edu

Abstract

In this parer, we study the complexity of answering conjunctive queries (CQ) with inequalities ($\neq$). In particular, we compare the complexity of the query with and without inequalities. The main contribution of our work is a novel combinatorial technique that enables the use of any Select-Project-Join query plan for a given CQ without inequalities in answering the CQ with inequalities, with an additional factor in running time that only depends on the query. To achieve this, we define a new projection operator that keeps a small representation (independent of the size of the database) of the set of input tuples that map to each tuple in the output of the projection; this representation is used to evaluate all the inequalities in the query. Second, we generalize a result by Papadimitriou-Yannakakis [18] and give an alternative algorithm based on the color-coding technique [4] to evaluate a CQ with inequalities by using an algorithm for the CQ without inequalities. Third, we investigate the structure of the query graph, inequality graph, and the augmented query graph with inequalities, and show that even if the query and the inequality graphs have bounded treewidth, the augmented graph not only can have an unbounded treewidth but can also be NP-hard to evaluate. Further, we illustrate classes of queries and inequalities where the augmented graphs have unbounded treewidth, but the CQ with inequalities can be evaluated in poly-time. Finally, we give necessary properties and sufficient properties that allow a class of CQs to have poly-time combined complexity with respect to any inequality pattern.

1998 ACM Subject Classification H.2.4 [Systems]: Query Processing

Keywords and phrases query evaluation, conjunctive query, inequality, treewidth

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.76

1 Introduction

In this paper, we study the complexity of answering conjunctive queries (CQ) with a set of inequalities of the form $x_i \neq x_j$ between variables in the query. The complexity of answering CQs without inequalities has been extensively studied in the literature during the past three decades. Query evaluation of CQs is NP-hard in terms of *combined complexity* (both query and database are inputs), while the *data complexity* of CQs (query is fixed) is in AC_0 [1]. Yannakakis [23] showed that evaluation of acyclic CQs has polynomial-time combined

* This work has been partially funded by the NSF awards IIS-1247469 and IIS-0911036, European Research Council under the FP7, ERC grant MoDaS, agreement 291071 and the Israel Ministry of Science.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 76–93

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

complexity. This result has been generalized later to CQs with bounded treewidth, bounded querywidth, or bounded hypertreewidth: the combined complexity remains polynomial if the width of a tree or query decomposition of the query (hyper-)graph is bounded [6, 10, 14, 9].

However, the complexity of query evaluation changes drastically once we add inequalities in the body of the query. Consider the following Boolean acyclic CQ P^k which can be solved in $O(k|D|)$ time on a database instance D:

$$P^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$$

If we add the inequalities $x_i \neq x_j$ for every $i < j$ and evaluate it on an instance where each $R_\ell, 1 \leq \ell \leq k$, corresponds to the edges in a graph with $k+1$ vertices, query evaluation becomes equivalent to asking whether the graph contains a Hamiltonian path, and therefore is NP-hard in k. Papadimitriou and Yannakakis [18] observed this fact and showed that still the problem is *fixed-parameter tractable* for acyclic CQs:

▶ **Theorem 1** ([18]). *Let q be an acyclic conjunctive query with inequalities and D be a database instance. Then, q can be evaluated in time $2^{O(k \log k)} \cdot |D| \log^2 |D|$ where k is the number of variables in q that appear in some inequality.*

The proof is based on the *color-coding* technique introduced by Alon-Yuster-Zwick in [4] that finds subgraphs in a graph. In general, answering CQs with inequalities is closely related to finding patterns in a graph, which has been extensively studied in the context of graph theory and algorithms. For example, using the idea of *representative sets*, Monien [16] showed the following: given a graph $G(V, E)$ and a vertex $s \in V$, there exists a deterministic $O(k! \cdot |E|)$ algorithm that finds all vertices v with a length-k path from s and also reports these paths (a trivial algorithm will run in time $O(|V|^k)$). Later, Alon *et al.* proposed the much simpler color-coding technique that can solve the same problem in expected time $2^{O(k)}|V|$ for undirected graphs and $2^{O(k)}|E|$ for directed graphs. These two ideas have been widely used to find other patterns in a graph, *e.g.*, for finding cycles of even length [3, 25, 4].

In the context of databases, Papadimitriou and Yannakakis [18] showed that answering acyclic CQs with comparison operators between variables ($<, \leq$ etc.) is harder than answering acyclic CQs with inequalities ($\neq$) since this problem is no longer fixed-parameter tractable. The query containment problem for CQs with comparisons and inequalities ($\neq, <, \leq$), *i.e.*, whether $Q_1 \subseteq Q_2$, has been shown to be Π_2^p-complete by van der Meyden [21]; the effect of several syntactic properties of Q_1, Q_2 on the complexity of this problem has been studied by Kolaitis *et al.* [14]. Durand and Grandjean [8] improved Theorem 1 from [18] by reducing the time complexity by a $log^2|D|$ factor. Answering queries with views in the presence of comparison operators has been studied by Afrati *et al.* [2]. Rosati [20] showed that answering CQs with inequalities is undecidable in description logic.

Our Contributions. In this paper we focus on the combined complexity of answering CQs with inequalities ($\neq$) where we explore both the structure of the query and the inequalities. Let q be a CQ with a set of variables, $\mathcal{I}$ be a set of inequalities of the form $x_i \neq x_j$, and k be the number of variables that appear in one of the inequalities in $\mathcal{I}$ ($k < |q|$). We will use $(q, \mathcal{I})$ to denote q with inequalities $\mathcal{I}$, and D to denote the database instance. We will refer to the combined complexity in $|D|, |q|, k$ by default (and not the data complexity on $|D|$) unless mentioned otherwise.

The main result in this paper says that any query plan for evaluating a CQ can be converted to a query plan for evaluating the same CQ with arbitrary inequalities, and the increase in running time is a factor that only depends on the query:

▶ **Theorem 2 (Main Theorem).** *Let q be a CQ that can be evaluated in time $T(|q|, |D|)$ using a Select-Project-Join (SPJ) query plan $\mathcal{P}_q$. Then, a query plan $\mathcal{P}_{q,\mathcal{I}}$ for $(q, \mathcal{I})$ can be obtained to evaluate $(q, \mathcal{I})$ in time $g(q, \mathcal{I}) \cdot \max(T(|q|, |D|),\ |D|)$ for a function g that is independent of the input database.* [1]

The key techniques used to prove the above theorem (Sections 3 and 4), and our other contributions in this paper (Sections 5, 6, and 7) are summarized below.

1. **(Section 3, 4)** Our main technical contribution is a new *projection* operator, called $\mathcal{H}$-projection. While the standard projection in relational algebra removes all other attributes for each tuple in the output, the new operator computes and retains a certain representation of the group of input tuples that contribute to each tuple in the output. This representation is of size independent of the database and allows the updated query plan to still correctly filter out certain tuples that do not satisfy the inequalities. In Section 3 we present the basic algorithmic components of this operator. In Section 4, we show how to apply this operator to transform the given query plan to another query plan that incorporates the added inequalities.
2. **(Section 5)** We generalize Theorem 1 to arbitrary CQs with inequalities (*i.e.*, not necessarily acyclic) by a simple application of the color-coding technique. In particular, we show (Theorem 21) that any algorithm that computes a CQ q on a database D in time $T(|q|, |D|)$ can be extended to an algorithm that can evaluate $(q, \mathcal{I})$ in time $f(k) \cdot \log(|D|) \cdot T(|q|, |D|)$. While Theorem 2 and Theorem 21 appear similar, there are several advantages of using our algorithm over the color-coding-based technique which we also discuss in Section 5.
3. **(Section 6)** The multiplicative factors dependent on the query in Theorem 1, Theorem 21, and (in the worst case) Theorem 2 are exponential in k. In Section 6 we investigate the combined structure of the queries and inequalities that allow or forbid poly-time combined complexity. We show that, even if q and $\mathcal{I}$ have a simple structure, answering $(q, \mathcal{I})$ can be NP-hard in k (Proposition 25). We also present a connection with the list coloring problem that allows us to answer certain pairings of queries with inequalities in poly-time combined complexity (Proposition 27).
4. **(Section 7)** We provide a sufficient condition for CQs, *bounded fractional vertex cover*, that ensures poly-time combined complexity when evaluated with *any set of inequalities* $\mathcal{I}$. Moreover, we show that families of CQs with unbounded integer vertex cover are NP-hard to evaluate in k (Theorem 29).

2 Preliminaries

We are given a CQ q, a set of inequalities $\mathcal{I}$, and a database instance D. The goal is to evaluate the query with inequality, denoted by $(q, \mathcal{I})$, on D. We will use $vars(q)$ to denote the variables in the body of query q and `Dom` to denote the active domain of D. The set of variables in the head of q (*i.e.*, the variables that appear in the output of q) is denoted by $head(q)$. If $head(q) = \emptyset$, q is called a *Boolean query*, while if $head(q) = vars(q)$, it is called a *full query*.

The set $\mathcal{I}$ contains inequalities of the form $x_i \neq x_j$, where $x_i, x_j \in vars(q)$ such that they belong to two distinct relational atoms in the query. We do not consider inequalities of the

[1] Some queries like $q() = R(x)S(y)$ can be evaluated in constant time whereas to evaluate the inequality constraints we need to scan the relations in D.

form $x_i \neq c$ for some constant c, or of the form $x_i \neq x_j$ where x_i, x_j only belong to the same relational atoms because these can be preprocessed by scanning the database instance and filtering out the tuples that violate these inequalities in time $O(|\mathcal{I}||D|)$. We will use k to denote the number of variables appearing in $\mathcal{I}$ ($k \leq |vars(q)| < |q|$).

Query Graph, Inequality Graph, and Augmented Graph. Given a CQ q and a set of inequalities $\mathcal{I}$, we define three undirected graphs on $vars(q)$ as the set of vertices:

The *query incidence graph* or simply the *query graph*, denoted by G^q, of a query q contains all the variables and the relational atoms in the query as vertices; an edge exists between a variable x and an atom S if and only if x appears in S.

The *inequality graph* $G^{\mathcal{I}}$ adds an edge between $x_i, x_j \in vars(q)$ if the inequality $x_i \neq x_j$ belongs to $\mathcal{I}$.

The query $(q, \mathcal{I})$ can be viewed as an augmentation of q with additional predicates, where for each inequality $x_i \neq x_j$ we add a relational atom $I_{ij}(x_i, x_j)$ to the query q, and add new relations I_{ij} to D instantiated to tuples $(a, b) \in \texttt{Dom} \times \texttt{Dom}$ such that $a \neq b$. The *augmented graph* $G^{q,\mathcal{I}}$ is the query incidence graph of this augmented query. Note that $G^{q,\mathcal{I}}$ includes the edges from G^q, and for every edge $(x_i, x_j) \in G^{\mathcal{I}}$, it includes two edges $(x_i, I_{ij}), (x_j, I_{ij})$; examples can be found in Section 6.

Treewidth and Acyclicity of a Query. We briefly review the definition of the treewidth of a graph and a query.

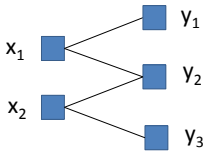
▶ **Definition 3** (Treewidth). A *tree decomposition* [19] of a graph $G(V, E)$ is a tree $T = (I, F)$, with a set $X(u) \subseteq V$ associated with each vertex $u \in I$ of the tree, such that the following conditions are satisfied:

1. For each $v \in V$, there is a $u \in I$ such that $v \in X(u)$,
2. For all edges $(v_1, v_2) \in E$, there is a $u \in I$ with $v_1, v_2 \in X(u)$,
3. For each $v \in V$, the set $\{u \in I \ : \ v \in X(u)\}$ induces a connected subtree of T.

The width of the tree decomposition $T = (I, F)$ is $\max_{u \in I} |X(u)| - 1$. The *treewidth* of G is the width of the tree decomposition of G having the minimum width.

Chekuri and Rajaraman defined the *treewidth of a query* q as the treewidth of the query incidence graph G^q [6]. A query can be viewed as a *hypergraph* where every hyperedge corresponds to an atom in the query and comprises the variables as vertices that belong to the relational atom. The *GYO-reduction* [11, 24] of a query repeatedly removes *ears* from the query hypergraph (hyperedges having at least one variable that does not belong to any other hyperedge) until no further ears exist. A query is *acyclic* if its GYO-reduction is the empty hypergraph, otherwise it is cyclic. For example, the query $P^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$ is acyclic, whereas the query $C^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_1)$ is cyclic.

There is another notion of width of a query called *querywidth* qw defined in terms of *query decomposition* such that the decomposition tree has relational atoms from the query instead of variables [6]; The relation between the querywidth qw and treewidth tw of a query is given by the inequality $tw/a \leq qw \leq tw + 1$, where a is the maximum arity of an atom in q. A query is acyclic if and only if its querywidth is 1; the treewidth of an acyclic query can be > 1 [6]. The notion of *hypertreewidth* has been defined by Gottlob *et al.* in [10]. A query can be evaluated in poly-time combined complexity if its treewidth, querywidth, or hypertreewidth is bounded [23, 6, 10, 14, 9].



(a) The bipartite graph $\mathcal{H}_0$

$$R = \{(1,1), (1,2), (1,4), (1,8), (2,1), (2,2), (2,3), (2,4), (3,2), (5,2), (10,2)\}$$

(b) The instance of $R(x_1, x_2)$

Figure 1 The running example (Example 5) for Section 3.

3 Main Techniques

In this section, we present the main techniques used to prove Theorem 2 with the help of a simple query q_2 that computes the cross product of two relations and projects onto the empty set. In particular, we consider the query $(q_2, \mathcal{I})$ with an arbitrary set of inequalities $\mathcal{I}$, where $q_2(\ ) = R(x_1, \ldots, x_m), S(y_1, \ldots, y_\ell)$. A naïve way to evaluate the query $(q_2, \mathcal{I})$ is to iterate over all pairs of tuples from R and S, and check if any such pair satisfies the inequalities in $\mathcal{I}$. This algorithm runs in time $O(m\ell|R||S|)$. We will show instead how to evaluate $(q_2, \mathcal{I})$ in time $f(q_2, \mathcal{I})(|R| + |S|)$ for some function f that is independent of the relations R and S.

The key idea is to compress the information that we need from R to evaluate the inequalities by computing a representation R' of R of such that the size of R' only depends on $\mathcal{I}$ and not on R. Further, we must be able to compute R' in time $O(f'(\mathcal{I})|R|)$. Then, instead of iterating over the pairs of tuples from R, S, we can iterate over the pairs from R' and S, which can be done in time $f''(q_2, \mathcal{I})|S|$. The challenge is to show that such a representation R' exists and that we can compute it efficiently.

We now formalize the above intuition. Let $X = \{x_1, \cdots, x_m\}$, $Y = \{y_1, \cdots, y_\ell\}$. Let $\mathcal{H} = G^{\mathcal{I}}$ denote the inequality graph; since q_2 has only two relations, $\mathcal{H}$ is a bipartite graph on X and Y. If a tuple t from S satisfies the inequalities in $\mathcal{I}$ when paired with at least one tuple in R, we say that t is $\mathcal{H}$-accepted by R, and it contributes to the answer of $(q_2, \mathcal{I})$. For a variable x_i and a tuple t, let $t[x_i]$ denotes the value of the attribute of t that corresponds to variable x_i.

▶ **Definition 4** ($\mathcal{H}$-accepted Tuples)**.** Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph. We say that a tuple t over Y is *$\mathcal{H}$-accepted* by a relation R if there exists some tuple $t_R \in R$ such that for every $(x_i, y_j) \in E$, we have $t_R[x_i] \neq t[y_j]$.

Notice that $(q_2, \mathcal{I})$ is true if and only if there exists a tuple $t_S \in S$ that is $\mathcal{H}$-accepted by R.

▶ **Example 5** (Running Example)**.** Let us define $\mathcal{H}_0 = (X, Y, E)$ with $X = \{x_1, x_2\}$, $Y = \{y_1, y_2, y_3\}$ and $E = \{(x_1, y_1), (x_1, y_2), (x_2, y_2), (x_2, y_3)\}$ (see Figure 1(a)) and consider the instance for R as depicted in Figure 1(b). This setting will be used as our running example.

Observe that the tuple $t = (2, 1, 3)$ is $\mathcal{H}_0$-accepted by R. Indeed consider the tuple $t' = (3, 2)$ in R: it is easy to check that all inequalities are satisfied by t, t'. In contrast, the tuple $(2, 1, 2)$ is not $\mathcal{H}_0$-accepted by R.

▶ **Definition 6** ($\mathcal{H}$-Equivalence)**.** Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph. Two relations R_1, R_2 of arity $m = |X|$ are *$\mathcal{H}$-equivalent* if for any tuple t of arity $\ell = |Y|$, the tuple t is $\mathcal{H}$-accepted by R_1 if and only if t is $\mathcal{H}$-accepted by R_2.

$\mathcal{H}$-equivalent relations form an equivalence class comprising instances of the same arity m. The main result in this section shows that for a given R, an $\mathcal{H}$-equivalent instance $R' \subseteq R$ of size independent of R can be efficiently constructed.

▶ **Theorem 7.** *Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph ($|Y| = \ell$) and R be a relation of arity $m = |X|$. Let $\phi(\mathcal{H}) = \ell! \prod_{j \in [\ell]} d_{\mathcal{H}}(y_j)$, where $d_{\mathcal{H}}(v)$ is the degree of a vertex v in $\mathcal{H}$. There exists an instance $R' \subseteq R$ such that:*

1. *R' is $\mathcal{H}$-equivalent with R*
2. *$|R'| \leq e \cdot \phi(\mathcal{H})$*
3. *R' can be computed in time $O(\phi(\mathcal{H})|R|)$.*

To describe how the algorithm that constructs R' works, we need to introduce another notion that describes the tuples of arity ℓ that are *not* $\mathcal{H}$-accepted by R. Let $\perp$ be a value that does not appear in the active domain $\texttt{Dom}$.

▶ **Definition 8** ($\mathcal{H}$-Forbidden Tuples)**.** Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph and R be a relation of arity $m = |X|$. A tuple t over Y with values in $\texttt{Dom} \cup \{\perp\}$ is *$\mathcal{H}$-forbidden* for R if for any tuple $t_R \in R$ there exist $y_j \in Y$ and $(x_i, y_j) \in E$ such that $t[y_j] = t_R[x_i]$.

▶ **Example 9** (Continued)**.** The reader can verify from Figure 1 that tuples of the form $(1, 2, x)$, where x can be any value, are $\mathcal{H}_0$-forbidden for R. Furthermore, notice that the tuple $(1, 2, \perp)$ is also $\mathcal{H}_0$-forbidden (in our construction $(1, 2, \perp)$ being $\mathcal{H}_0$-forbidden implies that any tuple of the form $(1, 2, x)$ is $\mathcal{H}_0$-forbidden).

Next we formalize the intuition of the above example. We say that a tuple t_1 defined over Y *subsumes* another tuple t_2 defined over Y if for any $y_j \in Y$, either $t_1[y_j] = \perp$ or $t_1[y_j] = t_2[y_j]$. Observe that if t_1 subsumes t_2 and t_1 is $\mathcal{H}$-forbidden, t_2 must be $\mathcal{H}$-forbidden as well. A tuple is *minimally $\mathcal{H}$-forbidden* if it is $\mathcal{H}$-forbidden and is not subsumed by any other $\mathcal{H}$-forbidden tuple. In our example, $(1, 2, 1)$ is subsumed by $(1, 2, \perp)$, so it is not minimally $\mathcal{H}_0$-forbidden, but the tuple $(1, 2, \perp)$ is. Lemma 10 stated below will be used to prove Theorem 7:

▶ **Lemma 10.** *Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph, and R be a relation defined on X. Then, the set of all minimally $\mathcal{H}$-forbidden tuples of R has size at most $\phi(\mathcal{H}) = \ell! \prod_{j \in [\ell]} d_{\mathcal{H}}(y_j)$ and it can be computed in time $O(\phi(\mathcal{H})|R|)$.*

To prove the above lemma, we present an algorithm that encodes all the minimally $\mathcal{H}$-forbidden tuples of R in a rooted tree $T_{\mathcal{H}}(R)$. The tree has labels for both the nodes and the edges. More precisely, the label $L(v)$ of some node v is either a tuple in R or a special symbol $\perp^*$ (only the leaves can have label $\perp^*$), while the label of an edge of the tree is a pair of the form (y_j, a), where $y_j \in Y$ and $a \in \texttt{Dom}$. The labels of the edges are used to construct a set of $\mathcal{H}$-forbidden tuples that includes the set of all minimally $\mathcal{H}$-forbidden tuples as follows:

For each leaf node v with label $L(v) = \perp^*$, let $(y_{j_1}, a_{j_1}), \ldots, (y_{j_m}, a_{j_m})$ be the edge labels in the order they appear from the root to the leaf. Then, the tuple $\texttt{tup}(v)$ defined on Y as follows is an $\mathcal{H}$-forbidden tuple (but not necessarily minimally $\mathcal{H}$-forbidden):

$$\texttt{tup}(v)[y_j] = \begin{cases} a_j & \text{if } j \in \{j_1, \ldots, j_m\} \\ \perp & \text{otherwise} \end{cases}$$

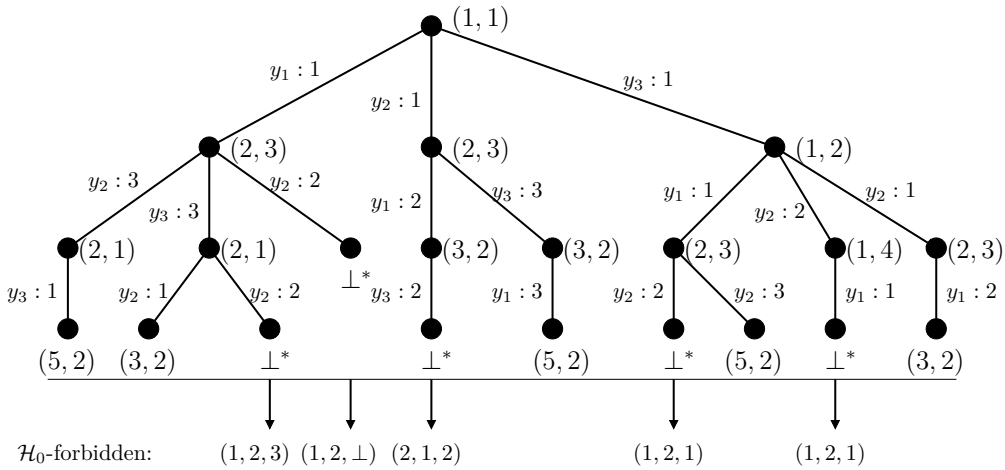


Figure 2 The tree $T_{\mathcal{H}}(R)$ of the running example. The diagram also presents how the $\mathcal{H}_0$-forbidden tuples are encoded by the tree.

Construction of $T_{\mathcal{H}}(R)$. We construct $T_{\mathcal{H}}(R)$ inductively by scanning through the tuples of R in an arbitrary order. As we read the next tuple t from R, we need to ensure that the $\mathcal{H}$-forbidden tuples that have been so far encoded by the tree are not $\mathcal{H}$-accepted by t: we achieve this by expanding some of the leaves and adding new edges and nodes to the tree. Therefore, after the algorithm has consumed a subset $R'' \subseteq R$, the partially constructed tree will be $T_{\mathcal{H}}(R'')$.

For the base of the induction, where $R'' = \emptyset$, we define $T_{\mathcal{H}}(\emptyset)$ as a tree that contains a single node (the root r) with label $L(r) = \perp^*$.

For the inductive step, let $T_{\mathcal{H}}(R'')$ be the current tree and let $t \in R$ be the next scanned tuple. The algorithm processes (in arbitrary order) all the leaf nodes v of the tree with $L(v) = \perp^*$. Let $(y_{j_1}, a_{j_1}), \ldots, (y_{j_p}, a_{j_p})$ be the edge labels in the order they appear on the path from root r to v. We distinguish two cases (for tuple t and a fixed leaf node v):

1. There exists $j \in \{j_1, \ldots, j_p\}$ and edge $(x_i, y_j) \in E$ such that $t[x_i] = a_j$. In this case, $\texttt{tup}(v)$ will be $\mathcal{H}$-forbidden in $R'' \cup \{t\}$; therefore, nothing needs to be done for this v.
2. Otherwise (*i.e.*, there is no such j), $\texttt{tup}(v)$ is not a $\mathcal{H}$-forbidden tuple for $R'' \cup \{t\}$. We set $L(v) = t$ (therefore, we never reassign the label of a node that has already been assigned to some tuple in R). There are two cases:
   a. If $p = \ell$, we cannot expand further from v (and will not expand in the future because now $L(v) \neq \perp^*$), since all y_j-s have been already set.
   b. If $p < \ell$, we expand the tree at node v. For every edge $(x_i, y_j) \in E$ such that $j \notin \{j_1, \ldots, j_p\}$, we add a fresh node $v^{i,j}$ with $L(v^{i,j}) = \perp^*$ and an edge $(v, v^{i,j})$ with label $(y_j, t[x_i])$. Notice that the tuples $\texttt{tup}(v^{i,j})$ will be now $\mathcal{H}$-forbidden in $R'' \cup \{t\}$.

The algorithm stops when either (a) all the tuples from R are scanned or (b) there exists no leaf node with label $\perp^*$.

▶ **Example 11** (Continued). We now illustrate the steps of the algorithm through the running example. After reading the first tuple, $t_1 = (1, 1)$, the algorithm expands the root node r to three children (for y_1, y_2, y_3), labels $L(r) = (1, 1)$ and labels the new edges as $(y_1, 1), (y_2, 1), (y_3, 1)$ and the new three leaves as $\perp^*$.

Suppose the second tuple $t_2 = (1, 2)$ is read next. First consider the leaf node with label $\perp^*$ that is reached from the root through the edge $(y_1, 1)$. At this point, the node

represents the tuple $(1, \perp, \perp)$. Observe that are in case (1) of the algorithm, and so the node is not expanded ($t_2[x_1] = 1$ and $m = 1 < 3 = \ell$). Consider now the third leaf node with label $\perp^*$, reached through the edge $(y_3, 1)$. We are now in case (2), and we have to expand the node. The available edges (since we have already assigned a value to y_3) are $(x_1, y_1), (x_1, y_2), (x_2, y_2)$. Hence, the node is labeled $(1, 2)$, and expands into three children, one for each of the above edges. These edges are labeled by $(y_1, 1), (y_2, 1), (y_2, 2)$ respectively; then the algorithm continues and at the end the tree in Figure 2 is obtained.

The $\mathcal{H}$-forbidden tuples encoded by the tree are not necessarily minimally $\mathcal{H}$-forbidden. However, for every minimally $\mathcal{H}$-forbidden tuple there exists a node in the tree that encodes it. In the running example, we find only two minimally $\mathcal{H}_0$-forbidden tuples for R: $(1, 2, \perp)$ and $(2, 1, 2)$. Furthermore, the constructed tree is not unique for R and depends on the order in which the tuples in R are scanned. The following lemma sums up the properties of the tree construction, and directly implies Lemma 10; the proof is deferred to the full version of the paper [15].

▶ **Lemma 12.** *$T_{\mathcal{H}}(R)$ satisfies the following properties:*

1. *The number of leaves is at most $\phi(\mathcal{H}) = \ell! \prod_{j \in [\ell]} d_{\mathcal{H}}(y_j)$.*
2. *Every leaf of $T_{\mathcal{H}}(R)$ with label $\perp^*$ encodes a $\mathcal{H}$-forbidden tuple.*
3. *Every minimally $\mathcal{H}$-forbidden tuple is encoded by some leaf of the tree with label $\perp^*$.*

For our running example, $\phi(\mathcal{H}_0) = 3! \cdot (1 \cdot 2 \cdot 1) = 12$, whereas the tree $T_{\mathcal{H}_0}(R)$ has only 10 leaves. We should note here that the bound $\phi(\mathcal{H})$ is tight, *i.e.* there exists an instance for which the number of minimally $\mathcal{H}$-forbidden tuples is exactly $\phi(\mathcal{H})$. [2]

We now discuss how we can use the tree $T_{\mathcal{H}}(R)$ to find a small $\mathcal{H}$-equivalent relation to R. It turns out that the connection is immediate: it suffices to collect the labels of all the nodes (not only leaves) of the tree $T_{\mathcal{H}}(R)$ that are not $\perp^*$. More formally:

$$\mathcal{E}_{\mathcal{H}}(R) = \{L(v) \mid v \in T_{\mathcal{H}}(R), L(v) \neq \perp^*\} \tag{1}$$

We can now show the following result, which completes the proof of Theorem 7:

▶ **Lemma 13.** *The set $\mathcal{E}_{\mathcal{H}}(R)$ is $\mathcal{H}$-equivalent to R and has size $|\mathcal{E}_{\mathcal{H}}(R)| \leq e \cdot \phi(\mathcal{H})$.*

▶ **Example 14** (Continued)**.** For our running example, the small $\mathcal{H}_0$-equivalent relation will be: $\mathcal{E}_{\mathcal{H}_0}(R) = \{(1,1), (1,2), (1,4), (2,1), (2,3), (3,2), (5,2)\}$. In other words, the tuples $(1,8), (2,2), (2,4), (10,2)$ are redundant and can be removed without affecting the answer to the query $(q_2, \mathcal{I})$.

Although the set of minimally $\mathcal{H}$-forbidden tuples is the same irrespective of the order by which the algorithm scans the tuples, the relation $\mathcal{E}_{\mathcal{H}}(R)$ depends on this order. It is an open problem to find the smallest possible $\mathcal{H}$-equivalent relation for R.

4 Query Plans for Inequalities

In this section, we use the techniques presented in the previous section as building blocks and prove Theorem 2. A *Select-Project-Join (SPJ) query plan* refers to a relational algebra expression that uses only selection (σ), projection (Π), and join ($\bowtie$) operators. Let $\mathcal{P}_q$ be

[2] For example, for $\mathcal{H}_0$ consider the instance $\{(1,2), (3,4), (5,6)\}$. The reader can check that the resulting tree has 12 leaves with label $\perp^*$, and that every leaf leads to a different minimally $\mathcal{H}$-forbidden tuple.

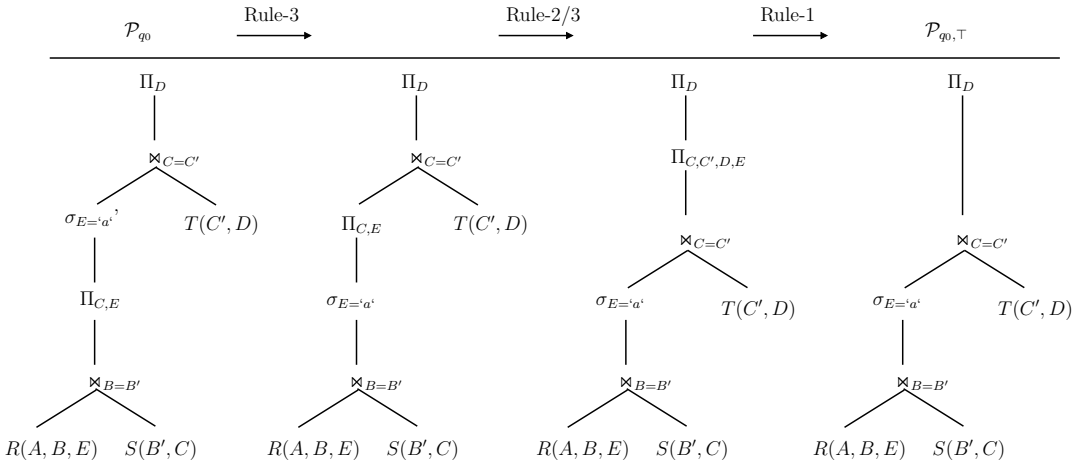


Figure 3 The relational plan $\mathcal{P}_{q_0}$ for Example 15, and the transformation to the plan $\mathcal{P}_{q_0,\top}$.

any SPJ query plan that computes a CQ q (without inequalities) on a database instance D in time $T(|q|, |D|)$. We will show how to transform $\mathcal{P}_q$ into a plan $\mathcal{P}_{q,\mathcal{I}}$ that computes $(q, \mathcal{I})$ in time $g(q, \mathcal{I}) \cdot \max(T(|q|, |D|))$. Without loss of generality, we assume that all the relation names and attributes in the base and derived relations (at intermediate steps in the plan) are distinct. Our running example for this section is given below:

▶ **Example 15.** Consider the query $(q_0, \mathcal{I})$, and the query plan $\mathcal{P}_{q_0}$ that computes q_0:

$$q_0(w) = R(x, y, `a`), S(y, z), T(z, w), \quad \mathcal{I} = \{x \neq z, y \neq w, x \neq w\}$$
$$\mathcal{P}_{q_0} = \Pi_D(\sigma_{E=`a`}(\Pi_{C,E}(R(A, B, E) \bowtie_{B=B'} S(B', C))) \bowtie_{C=C'} T(C', D))$$

The query plan $\mathcal{P}_{q_0}$ is depicted in Figure 3.

Clearly, this plan by itself does not work for $(q_0, \mathcal{I})$ as it is losing information that is essential to evaluate the inequlities, *e.g.*, $B(= B')$ is being projected out and it is used later in the inequality $x \neq w$ with the attribute C of T. To overcome this problem while keeping the same structure of the plan, we define a new projection operator that allows us to perform valid algebraic transformations, even in the presence of inequalities. Let $\text{att}(R)$ be the set of attributes that appear in a base or derived relation R; a query plan or sub-plan $\mathcal{P}$ is a derived relation with attributes $\text{att}(\mathcal{P})$. If $X \subseteq \text{att}(R)$, let $\bar{X}^R = \text{att}(R) \setminus X$.

▶ **Definition 16** ($\mathcal{H}$-Projection)**.** Let R be a base or a derived relation in $\mathcal{P}$. Let $X \subseteq \text{att}(R)$ and $\mathcal{H} = (\bar{X}^R, \text{att}(\mathcal{P}) \setminus \text{att}(R), E)$ be a bipartite graph. Then, the $\mathcal{H}$-projection of R on X, denoted $\Pi^{\mathcal{H}}_X(R)$, is defined as

$$\Pi^{\mathcal{H}}_X(R) = \bigcup_{\alpha \in \Pi_X(R)} \mathcal{E}_{\mathcal{H}}(\sigma_{X=\alpha}(R)) \tag{2}$$

where $\mathcal{E}_{\mathcal{H}}$ denotes an $\mathcal{H}$-equivalent subrelation as defined and constructed in equation (1).

Intuitively, $\mathcal{H}$ contains the inequalities between the attributes in $\bar{X}^R$ (that are being projected out) and the attributes of the rest of the query plan. The operator $\Pi^{\mathcal{H}}_X$ first groups the tuples from R according to the values of the X-attributes, but then instead of projecting out the values of the attributes in $\bar{X}^R$ for each such group, it computes a small $\mathcal{H}$-equivalent subrelation according to the graph $\mathcal{H}$.

▶ Observation 17. The $\mathcal{H}$-projection of a relation R on X satisfies the following properties:

1. $\Pi_X(R) = \Pi_X(\Pi_X^{\mathcal{H}}(R))$
2. $|\Pi_X^{\mathcal{H}}(R)| \leq e \cdot \phi(\mathcal{H}) \cdot |\Pi_X(R)|$ (ref. Lemma 7)

First step. To construct the plan $\mathcal{P}_{q,\mathcal{I}}$ from $\mathcal{P}_q$, we first create an equivalent query plan $\mathcal{P}_{q,\top}$ by pulling all the projections in $\mathcal{P}_q$ to the top of the plan. The equivalence of $\mathcal{P}_q$ and $\mathcal{P}_{q,\top}$ is maintained by the following standard algebraic rules regarding projections:

(Rule-1) Absorption: If $X \subseteq Y$, then $\Pi_X(R) = \Pi_X(\Pi_Y(R))$.
(Rule-2) Distribution: If $X_1 \subseteq \text{att}(R_1)$ and $X_2 = \text{att}(R_2)$, then $\Pi_{X_1 \cup X_2}(R_1 \times R_2) = \Pi_{X_1}(R_1) \times R_2$.
(Rule-3) Commutativity with Selection: If the selection condition θ is over a subset of X, then $\sigma_\theta(\Pi_X(R)) = \Pi_X(\sigma_\theta(R))$.

Figure 3 depicts how each rule is applied in our running example to transform the initial query plan $\mathcal{P}_{q_0}$ to $\mathcal{P}_{q_0,\top}$, where the only projection occurs in the top of the query plan. Observe that to distribute a projection over a join $R_1 \bowtie_{A_1=A_2} R_2$ (and not a cartesian product), we can write it as $\sigma_{A_1=A_2}(R_1 \times R_2)$, use both (Rule-2) and (Rule-3) to push the projection, and then write it back in the form as $R_1 \bowtie_{A_1=A_2} R_2$.

The plan $\mathcal{P}_{q,\top}$ will be of the form $\mathcal{P}_{q,\top} = \Pi_X(\mathcal{P}_0)$, where $\mathcal{P}_0$ is a query plan that contains only selections and joins. Notice that the plan $\Pi_X(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ correctly computes $(q, \mathcal{I})$, since it applies the inequalities before projecting out any attributes.[3] However, the running time is not comparable with that of $\mathcal{P}_q$ since the structures of the plans $\mathcal{P}_q$ and $\Pi_X(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ are very different. To achieve comparable running time, we modify $\Pi_X(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ by applying the corresponding rules of (Rule-1), (Rule-2), (Rule-3) for $\mathcal{H}$-projection in the reverse order.

Second step. To convert projections to $\mathcal{H}$-projections, first, we replace Π_X with $\Pi_X^{\mathcal{H}_0}$, where $\mathcal{H}_0 = (\text{att}(\mathcal{P}_0) \setminus X, \emptyset, \emptyset)$. Notice that $\Pi_X^{\mathcal{H}_0}$ is essentially like Π_X, but instead of removing the attributes that are not in X, the operator keeps an arbitrary witness. Thus, if we compute $\Pi_X^{\mathcal{H}_0}(\sigma_{\mathcal{I}}(\mathcal{P}_0))$, we not only get all tuples t in $(q, \mathcal{I})$, but for every such tuple we obtain a tuple t' from $(q^f, \mathcal{I})$ such that $t = t'[X]$. For our running example, $X = \{D\}$, and therefore, $\mathcal{H}_0 = (\{A, B, B', C, C', E\}, \emptyset, \emptyset)$ (see the rightmost plan in Figure 4).

Third step. We next present the rules for $\mathcal{H}$-projections to convert $\Pi_X^{\mathcal{H}_0}(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ to the desired plan $\mathcal{P}_{q,\mathcal{I}}$. To show that the rules are algebraically correct, we need a weaker version of plan equivalence.

▶ **Definition 18** (Plan Equivalence)**.** Two plans $\mathcal{P}_1, \mathcal{P}_2$ are equivalent under $\Pi_X^{\mathcal{H}}$, denoted $\mathcal{P}_1 \equiv_X^{\mathcal{H}} \mathcal{P}_2$, if for every tuple α, $\mathcal{E}_{\mathcal{H}}(\sigma_{X=\alpha}(\mathcal{P}_1))$ and $\mathcal{E}_{\mathcal{H}}(\sigma_{X=\alpha}(\mathcal{P}_2))$ are $\mathcal{H}$-equivalent.

In other words, we do not need to have the same values of the attributes that are being projected out by Π_X in the small sub-relations $\mathcal{E}_{\mathcal{H}}$. We write $\mathcal{I}[X_1, X_2] \subseteq \mathcal{I}$ to denote the inequalities between attributes in subsets X_1 and X_2. For convenience, we also write $\mathcal{I}[X, X] = \mathcal{I}[X]$. We use $E[X_1, X_2]$ in a similar fashion, where E is the set of edges in a bipartite graph. Let $\mathbf{A} = \text{att}(\mathcal{P}_0)$. We apply the transformation rules for a sub-plan that is of the form $\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(S))$, where $\mathcal{I}$ is defined on $\text{att}(S)$ and $\mathcal{H} = (\bar{X}^S, \mathbf{A}, E)$. [4] The rules are:

[3] From here on we let $\mathcal{I}$ denote inequalities on attributes and not variables.
[4] For the sake of simplicity, we do not write the bipartite graph as $\mathcal{H} = (\bar{X}^S, \mathbf{A} \setminus \text{att}(S), E)$. However, the transformation rules ensure that the edges E in the bipartite graph are always between $\bar{X}^S$ and $\mathbf{A} \setminus \text{att}(S)$.

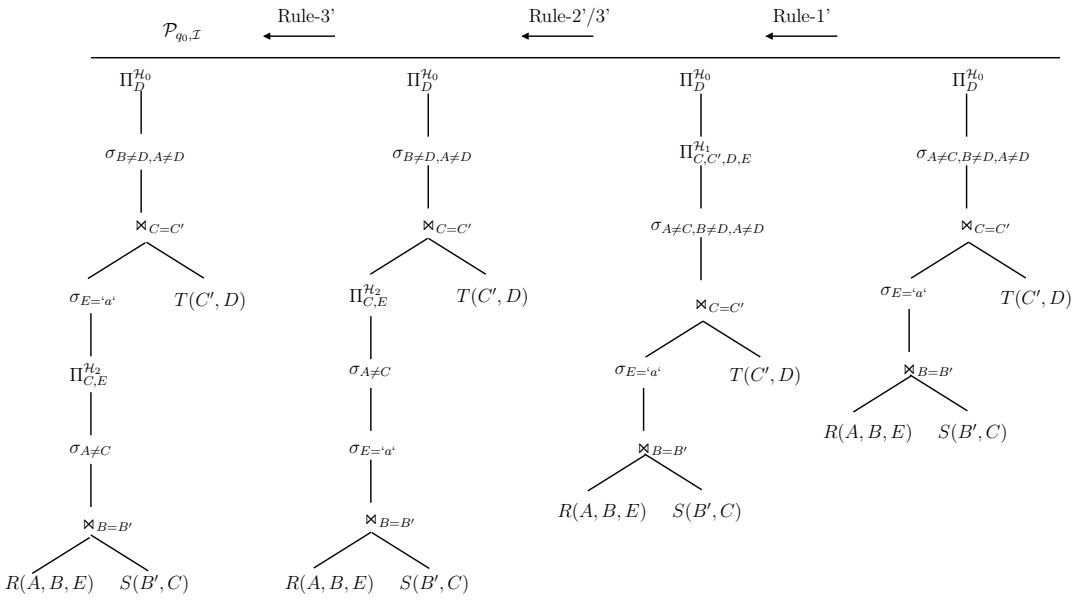


Figure 4 The reverse application of rules for Example 15. The bipartite graphs defined have edge sets $E(\mathcal{H}_0) = \emptyset$, $E(\mathcal{H}_1) = \emptyset$ and $E(\mathcal{H}_2) = \{(A, D), (B, D)\}$.

(Rule-1'). If $X \subseteq Y$ and $\mathcal{H}' = (\bar{Y}^S, \mathbf{A}, E[\bar{Y}^S, \mathbf{A}])$, then

$$\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(S)) \quad \equiv_X^{\mathcal{H}} \quad \Pi_X^{\mathcal{H}}(\Pi_Y^{\mathcal{H}'}(\sigma_{\mathcal{I}}(S)))$$

In the running example, we have $X = \{D\}$, $Y = \{C, C', D, E\}$, and $\text{att}(S) = \mathbf{A} = \{A, B, B', C, C', D, E\}$. The new bipartite graph for Rule-1' in Figure 4 (corresponding to Rule-1 in Figure 3) is $\mathcal{H}_1 = (\{A, B, B'\}, \mathbf{A}, \emptyset)$.

(Rule-2'). Let $S = R_1 \times R_2$, and $X = X_1 \cup Z_2$, where $X_1 \subseteq \text{att}(R_1) = Z_1$ and $Z_2 = \text{att}(R_2)$. If we define $\mathcal{H}' = (Z_1 \setminus X_1,\ \mathbf{A},\ E[Z_1 \setminus X_1, \mathbf{A}] \cup \mathcal{I}[Z_1 \setminus X_1, Z_2])$, then

$$\Pi_{X_1 \cup Z_2}^{\mathcal{H}}(\sigma_{\mathcal{I}}(R_1 \times R_2)) \quad \equiv_X^{\mathcal{H}} \quad \sigma_{\mathcal{I} \setminus \mathcal{I}[Z_1]}(\Pi_{X_1}^{\mathcal{H}'}(\sigma_{\mathcal{I}[Z_1]}(R_1)) \times R_2)$$

This rule adds new edges to the bipartite graph (which is initially empty) from the set of inequalities $\mathcal{I}$. In the running example, we have $X_1 = \{C, E\} \subseteq \{A, B, B', C, E\} = Z_1$ and $Z_2 = \{C', D\}$. Since $E(\mathcal{H}_1) = \emptyset$, to construct the edge set of the new bipartite graph $\mathcal{H}_2$, we need to find the inequalities that have one attribute in $Z_1 \setminus X_1 = \{A, B, B'\}$ and the other in $Z_2 = \{C', D\}$: these are $A \neq D$ and $B \neq D$. Hence, $\mathcal{H}_2 = (\{A, B, B'\}, \mathbf{A}, \{(A, D), (B, D)\})$, and the application of the rule is depicted in Figure 4.

(Rule-3'). If θ is defined over a subset of X, and $S = \sigma_\theta(R)$:

$$\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(\sigma_\theta(R))) \quad \equiv_X^{\mathcal{H}} \quad \sigma_\theta(\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(R)))$$

In the running example, we move the selection operator $\sigma_{E='a'}$ before the projection operator $\Pi_{C,E}^{\mathcal{H}_2}$ as the last step of the transformation.

The proof of correctness of these transformations (*i.e.*, (Rule-1'), (Rule-2'), (Rule-3') preserve the equivalence of the plans under $\Pi_X^{\mathcal{H}}$) is deferred to the full version of the paper [15]. After applying the above transformations in the reverse order, the following lemma holds:

▶ **Lemma 19.** *Let $\mathcal{P}_q$ be an SPJ plan for q. For a set of inequalities $\mathcal{I}$, the transformed plan $\mathcal{P}_{q,\mathcal{I}}$ has the following properties:*

1. *If $\mathcal{P}_{q,\top} = \Pi_X(\mathcal{P}_0)$, the plan $\Pi_X(\mathcal{P}_{q,\mathcal{I}})$ computes $(q,\mathcal{I})$ (after projecting out the attributes that served as witness from $\mathcal{P}_{q,\mathcal{I}}$).*
2. *For every Π_X operator in $\mathcal{P}_q$, there exists a corresponding $\Pi_X^{\mathcal{H}}$ operator in $\mathcal{P}_{q,\mathcal{I}}$ for some appropriately constructed $\mathcal{H}$.*
3. *Every intermediate relation R in $\mathcal{P}_{q,\mathcal{I}}$ has size at most $e \cdot \max_{\mathcal{H}}\{\phi(\mathcal{H})\} \cdot |R'|$, where R' is the corresponding intermediate relation in $\mathcal{P}_q$.*
4. *If $T(|q|,|D|)$ is the time to evaluate $\mathcal{P}_q$, the the time to evaluate $\mathcal{P}_{q,\mathcal{I}}$ increases by a factor of at most $(e \cdot \max_{\mathcal{H}}\{\phi(\mathcal{H})\})^2$.*

Theorem 2 directly follows from the above lemma. To prove the bound on the running time, we use the fact that each operator (selection, projection or join) can be implemented in at most quadratic time in the size of the input (*i.e.*, $T(MN) \leq cM^2T(N)$). Additionally, notice that, if k is the vertex size of the inequality graph, then $\max_{\mathcal{H}}\{\phi(\mathcal{H})\} \leq k!k^k$. Hence, the running time can increase at most by a factor of $2^{O(k \log k)}$ when inequalities are added to the query. In our running example, $\phi(\mathcal{H}_0) = 1$, $\phi(\mathcal{H}_0) = 1$ and $\phi(\mathcal{H}_2) = 2$, hence the resulting intermediate relations in will be at most $2e$ times larger than the ones in $\mathcal{P}_{q_0}$.

The following query with inequalities is an example where our algorithm gives much better running time than the color-coding-based or treewidth-based techniques described in the subsequent sections.

▶ **Example 20.** Consider $P^k() = R_1(x_1,x_2), R_2(x_2,x_3), \cdots, R_k(x_k,x_{k+1})$ with inequalities $\mathcal{I} = \{x_i \neq x_{i+2} \mid i \in [k-1]\}$. Let $\mathcal{P}$ be the SPJ plan that computes this acyclic query in time $O(k|D|)$ by performing joins from left to right and projecting out the attributes as soon as they join. Then, the plan $\mathcal{P}_{\mathcal{I}}$ that is constructed has constant $\max_{\mathcal{H}}\{\phi(\mathcal{H})\}$; thus, $(P^k,\mathcal{I})$ can be evaluated in time $O(k|D|)$ as well.

▶ Remark. In this section we compared the running time of queries with inequalities with SPJ plans that compute the query without the inequalities. However, optimal algorithms that compute CQs may not use SPJ plans, as the recent worst-case optimal algorithms in [17, 22] show. These algorithms apply to conjunctive queries without projections, where any inequality can be applied at the end without affecting the asymptotic running time. However, there are cases where nonstandard algorithms for Boolean CQs run faster than SPJ algorithms, *e.g.* $q() = R(x_1,x_2), R(x_2,x_3), \ldots, R(x_{2k},x_1)$, can be computed in time $O(N^{2-1/k})$, where $N = |R|$. We show in the full version [15] that our techniques can be applied in this case as well. However, it is an open whether we can use them for any black-box algorithm.

5 Color-coding Technique and Generalization of Theorem 1

In this section, we will review the color-coding technique from [4] and use it to generalize Theorem 1 for arbitrary CQs with inequalities (*i.e.*, not necessarily acyclic queries)[5].

▶ **Theorem 21.** *Let q be a CQ that can be evaluated in time $T(|q|,|D|)$. Then, $(q,\mathcal{I})$ can be computed in time $2^{O(k \log k)} \cdot log(|D|) \cdot T(|q|,|D|)$ where k is the number of variables in $\mathcal{I}$.*

[5] The $\log^2(|D|)$ factor in Theorem 1 is reduced to $\log(|D|)$ in Theorem 21, but this is because one log factor was due to sorting the relations in the acyclic query, and now this hidden in the term $T(|q|,|D|)$.

First, we state the original randomized color-coding technique to describe the intuition: randomly color each value of the active domain by using a hash function h, use these colors to check the inequality constraints, and use the actual values to check the equality constraints.

For a CQ q, let q^f denote the *full query* (without inequalities), where every variable in the body appears in the head of the query q. For a variable x_i and a tuple t, $t[x_i]$ (or simply $t[i]$ where it is clear from the context) denotes the value of the attribute of t that corresponds to variable x_i. Let $t \in q^f(D)$. We say that t *satisfies the inequalities* $\mathcal{I}$, denoted by $t \models \mathcal{I}$, if for each $x_i \neq x_j$ in $\mathcal{I}$, $t[x_i] \neq t[x_j]$. We say that t *satisfies the inequalities* $\mathcal{I}$ *with respect to the hash function* h, denoted by $t \models_h \mathcal{I}$, if for each such inequaity $h(t[x_i]) \neq h(t[x_j])$.

Recall that k is the number of variables that appear in $\mathcal{I}$. Let h be a perfectly random hash function $h : \mathtt{Dom} \to [p]$ (where $p \geq k$). For any $t \in q^f(D)$ if t satisfies $\mathcal{I}$, then with high probability it also satisfies $\mathcal{I}$ with respect to h, *i.e.*, $\mathbf{Pr}_h[t \models_h \mathcal{I} \mid t \models \mathcal{I}] \geq \frac{p(p-1)\cdots(p-k+1)}{p^k}$ $\geq e^{-2\sum_{i=1}^{k-1}(i/p)} \geq e^{-k}$, where we used the fact that $1 - x \geq e^{-2x}$ for $x \leq \frac{1}{2}$. Therefore, by repeating the experiment $2^{O(k)}$ times we can evaluate a Boolean query with constant probability.

This process can be derandomized leading to a deterministic algorithm (for evaluating any CQ, not necessarily Boolean) by selecting h from a family $\mathcal{F}$ of k-perfect hash functions. A k-perfect family guarantees that for every tuple of arity at most k (with values from the domain $\mathtt{Dom}$), there will be some $h \in \mathcal{F}$ such that for all $i, j \in [k]$, if $t[i] \neq t[j]$, then $h(t[i]) \neq h(t[j])$ (and thus if $t \models \mathcal{I}$, then $t \models_h \mathcal{I}$) It is known (see [4]) that we can construct a k-perfect family of size $|\mathcal{F}| = 2^{O(k)} \log(|\mathtt{Dom}|) = 2^{O(k)} \log |D|$.[6]

A coloring $\mathbf{c}$ of the vertices of the inequality graph $G^{\mathcal{I}}$ with k colors is called a *valid k-coloring*, if for each $x_i \neq x_j$ we have that $c_i \neq c_j$ where c_i denotes the color of variable x_i under $\mathbf{c}$. Let $\mathcal{C}(G^{\mathcal{I}})$ denote all the valid colorings of $G^{\mathcal{I}}$. For each such coloring $\mathbf{c}$ and any given hash function $h : \mathtt{Dom} \to [k]$, we can define a subinstance $D[\mathbf{c}, h] \subseteq D$ such that for each relation R, $R^{D[\mathbf{c},h]} = \{t \in R^D \mid \forall x_i \in vars(R), h(t[x_i]) = c_i\}$. In other words, the subinstance $D[\mathbf{c}, h]$ picks only the tuples that under the hash function h agree with the coloring $\mathbf{c}$ of the inequality graph. Then the algorithm can be stated as follows:

> **Deterministic Algorithm:** *For every hash function $h : \mathtt{Dom} \to [k]$ in a k-perfect hash family $\mathcal{F}$, for every valid k-coloring $\mathbf{c} \in \mathcal{C}(G^{\mathcal{I}})$ of the variables, evaluate the query q on the sub-instance $D[\mathbf{c}, h]$. Output $\bigcup_{h \in \mathcal{F}} \bigcup_{\mathbf{c} \in \mathcal{C}(G^{\mathcal{I}})} q(D[\mathbf{c}, h])$.*

The correctness argument for the above algorithm is presented in [15]. The running time of the algorithm is $O(|\mathcal{F}| \cdot |\mathcal{C}(G^{\mathcal{I}})| \cdot T(q, |D|))$. Since $|\mathcal{F}| \leq 2^{O(p)} \log |D|$ and $|\mathcal{C}(G^{\mathcal{I}})| \leq k^k$, Theorem 21 follows.

Comparison of Theorem 2 with Theorem 21. The factors dependent on the query in these two theorems ($g(q, \mathcal{I})$ in Theorem 2 and $f(k)$ in Theorem 21) are both bounded by $2^{O(k \log k)}$. However, our technique outperforms the color-coding technique in several respects. First, the randomized color-coding technique is simple and elegant, but is unsuitable to implement in a database system that typically aims to find deterministic answers. On the other hand, apart from the additional $\log(|D|)$ factor, the derandomized color-coding technique demands the construction of a new k-perfect hash family for every database instance and query, and therefore may not be efficient for practical purposes. Our algorithm requires no preprocessing and can be applied in a database system by maintaining the same query plan and using

[6] Assuming $\mathtt{Dom}$ includes only the attributes that appear as variables in the query q, $|\mathtt{Dom}| \leq |D||q|$.

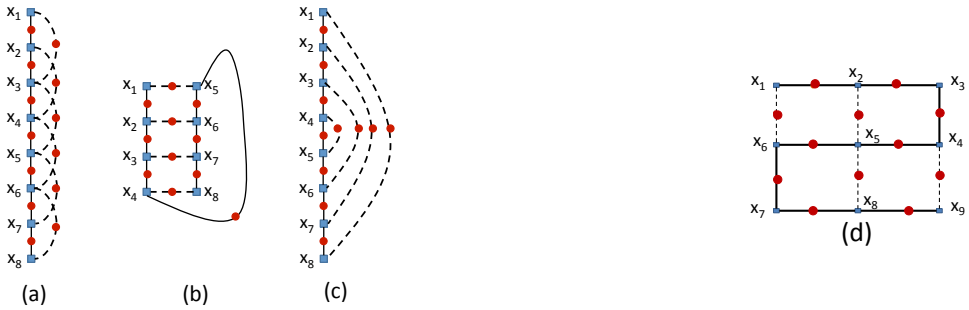


Figure 5 Augmented graphs for Example 22 ($k = 7$) and Example 23 ($k = 8$). The solid and dotted edges come from the query and inequalities respectively; the blue squares denote variables, and red circles denote (unnamed) relational atoms: (a) $(P^7, \mathcal{I}_1)$, (b) $(P^7, \mathcal{I}_2)$, (c) $(P^7, \mathcal{I}_3)$, (d) $(P^7, \mathcal{I}_4)$.

a more sophisticated projection operation. More importantly, the color coding technique is *oblivious* of the combined structure of the query and the inequalities. As an example, consider the path query P^k, together with the inequalities $\mathcal{I}_1 = \{x_i \neq x_{i+2} : i \in [k-1]\}$. The color-coding-based algorithm has a running time of $2^{O(k \log k)}|D| \log |D|$. However, as discussed in Section 4, we can compute this query in time $O(k|D|)$, thus the exponential dependence on k is eliminated.

6 CQs and Inequalities with Polynomial Combined Complexity

In this section, we investigate classes of queries and inequalities that entail a poly-time combined complexity for $(q, \mathcal{I})$ in terms of the treewidths of query graph G^q, inequality graph $G^{\mathcal{I}}$, and augmented graph $G^{q,\mathcal{I}}$. If the augmented graph $G^{q,\mathcal{I}}$ has bounded treewidth, then $(q, \mathcal{I})$ can be answered in poly-time combined complexity [23, 6]. We give examples of such q and $\mathcal{I}$ below:

▶ **Example 22.** Consider the path query: $P^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$, which is acyclic, and consider the following inequality patterns (see Figure 5): (i) $(P^k, \mathcal{I}_1)$ where $\mathcal{I}_1 = \{x_i \neq x_{i+2} : i \in [k-1]\}$ has treewidth 2. (ii) $(P^k, \mathcal{I}_2)$ where $\mathcal{I}_2 = \{x_i \neq x_{i+\frac{k}{2}} : i \in [\frac{k+1}{2}]\}$ has treewidth 3 (k is odd). (iii) $(P^k, \mathcal{I}_3)$ where $\mathcal{I}_3 = \{x_i \neq x_{k-i+1} : i \in [\frac{k+1}{2}]\}$ has treewidth 2 (k is odd).

However, for certain inputs our algorithm in Section 4 can outperform the treewidth-based techniques since it considers the inequality structure more carefully. For instance, even though the augmented graph of $(P^k, \mathcal{I}_1)$ has treewidth 2 (see Figure 5 (a)), the techniques of [23] will give an algorithm with running time $O(poly(k)|D|^2)$, whereas the algorithm in Section 4 gives a running time of $O(k|D|)$.

Indeed, the treewidth of $G^{q,\mathcal{I}}$ is at least as large as the treewidth of G^q and $G^{\mathcal{I}}$. As mentioned earlier, when $G^{\mathcal{I}}$ is the complete graph on $k+1$ variables (with treewidth $= k+1$), answering $(P^k, \mathcal{I})$ is as hard as finding if a graph on $k+1$ vertices has a Hamiltonian path, and therefore is NP-hard in k. Interestingly, even when both G^q and $G^{\mathcal{I}}$ have bounded treewidths, $G^{q,\mathcal{I}}$ may have unbounded treewidth as illustrated by the following example:

▶ **Example 23.** Consider $(P^k, \mathcal{I}_4)$ (see Figure 5(d)), where $k+1 = p^2$ for some p. Algebraically, we can write $\mathcal{I}_4$ as: $\mathcal{I}_4 = \{x_i \neq x_{\lfloor i/p \rfloor + 1 + 2p - (i \mod p)} \mid i = 1, \ldots, p(p-1)\}$. The edges for P^k are depicted in the figure as an alternating path on the grid with solid edges, whereas

the remaining edges are dotted and correspond to the inequalities. Here both G^{P^k} and $G^{\mathcal{I}_4}$ have treewidth 1, but $G^{P^k,\mathcal{I}_4}$ has treewidth $\Theta(\sqrt{k})$.

However, this does not show that evaluation of the query $(P^k, \mathcal{I}_4)$ is NP-hard in k, which we prove below by a reduction from the *list coloring problem*:

▶ **Definition 24** (List Coloring)**.** Given an undirected graph $G = (V, E)$, and a list of admissible colors $L(v)$ for each vertex $v \in V$, list coloring asks whether there exists a coloring $c(v) \in L(v)$ for each vertex v such that the adjacent vertices in G have different colors.

The list coloring problem generalizes the coloring problem, and therefore is NP-hard. List coloring is NP-hard even on grid graphs with 4 colors and where $2 \leq |L(v)| \leq 3$ for each vertex v [7]; we show NP-hardness for $(P^k, \mathcal{I}_4)$ by a reduction from list coloring on grids.

▶ Proposition 25. The combined complexity of evaluating $(P^k, \mathcal{I}_4)$ is NP-hard, where both the query P^k and the inequality graph G are acyclic (have treewidth 1).

In fact, the above proposition can be generalized as follows: *if the graph $G^{q,\mathcal{I}}$ is NP-hard for list coloring for a query q where each relation has arity 2, then evaluation of the query $(q, \mathcal{I})$ is also NP-hard in the size of the query.*

On the contrary, $(q, \mathcal{I})$ may not be hard in terms of combined complexity if the treewidth of $G^{q,\mathcal{I}}$ is unbounded, which we also show with the help of the list coloring problem. Consider the queries $F^k(\ ) = R_1(x_1), R_2(x_2), \ldots, R_k(x_k)$. Given inequalities $\mathcal{I}$, the evaluation of $(F^k, \mathcal{I})$ is *equivalent* to the list coloring problem on the graph $G^{\mathcal{I}}$ when the available colors for each vertex x_i are the tuples in $R_i(x_i)$. Since list coloring is NP-hard:

▶ Proposition 26. The evaluation of $(F^k, \mathcal{I})$ is NP-hard in k for arbitrary inequalities $\mathcal{I}$.

Therefore, answering $(F^k, \mathcal{I})$ becomes NP-hard in k even for this simple class of queries if we allow arbitrary set of inequalities $\mathcal{I}$ (this also follows from Theorem 29). However, list coloring can be solved in polynomial time for certain graphs $G^{\mathcal{I}}$: (i) **Trees** (the problem can be solved in time $O(|V|)$ independent of the available colors[13]), and in general graphs of constant treewidth. (ii) **Complete graphs** (by a reduction to *bipartite matching*).[7] In general, if the connected components of G are either complete graphs or have constant treewidth, list coloring can be solved in polynomial time. Therefore, on such graphs as $G^{\mathcal{I}}$, the query $(F^k, \mathcal{I})$ can be computed in poly-time in k and $|D|$. Here we point out that none of the other algorithms given in this paper can give a poly-time algorithm in $k, |D|$ for $(F^k, \mathcal{I})$ when $G^{\mathcal{I}}$ is the complete graph (and therefore has treewidth k). The following proposition generalizes this property:

▶ Proposition 27. Let q be a Boolean CQ, where each relational atom has arity at most 2. If q has a *vertex cover* (a set of variables that can cover all relations in q) of constant size and the list coloring problem on $G^{\mathcal{I}}$ can be solved in poly-time, then $(q, \mathcal{I})$ can be answered in poly-time combined complexity.

The proof is given in the full version of the paper. To see an example, consider the star query $Z^n(\ ) = R_1(y, x_1), \ldots, R_n(y, x_n)$ which has a vertex cover $\{y\}$ of size 1. We iterate over all possible values of y: for each such value $\alpha \in \texttt{Dom}$, the query $R_1(\alpha, x_1), \ldots, R_n(\alpha, x_n)$ is equivalent to F^n, and therefore $(Z^n, \mathcal{I})$ can be evaluated in poly-time in combined complexity when $G^{\mathcal{I}}$ is an easy instance of list coloring.

[7] We can construct a bipartite graph where all vertices v appear on one side, the colors appear on the other side, and there is an edge (v, c) if $c \in L(v)$. Then the list coloring problem on complete graph is solvable if and only if there is a perfect matching in the graph.

7 CQs with Polynomial Combined Complexity for All Inequalities

This section aims to find CQs q such that computing $(q, \mathcal{I})$ has poly-time combined complexity, no matter what the choice of $\mathcal{I}$ is. Here we present a sufficient condition for this, and a stronger necessary condition.

A *fractional edge cover* of a CQ q assigns a number v_R to each relation $R \in q$ such that for each variable x, $\sum_{R:x\in vars(R)} v_R \geq 1$. A *fractional vertex packing* (or, *independent set*) of q assigns a number u_x to each variable x, such that $\sum_{x\in vars(R)} u_x \leq 1$ for every relation $R \in q$. By duality, the minimum fractional edge cover is equal to the maximum fractional vertex packing. When each $v_R \in \{0, 1\}$ we get an *integer edge cover*, and when each $u_x \in \{0, 1\}$ we get an *integer vertex packing*.

▶ **Definition 28.** A family $\mathcal{Q}$ of Boolean CQs has *unbounded fractional (resp. integer) vertex packing* if there exists a function $T(n)$ such that for every integer $n > 0$ it can output in time $poly(n)$ a query $q \in \mathcal{Q}$ that has a fractional (resp. integer) vertex packing of size at least n (counting relational atoms as well as variables).

A family $\mathcal{Q}$ of Boolean CQs has *bounded fractional (resp. integer) vertex packing* if there exists a constant $b > 0$ such that for any $q \in \mathcal{Q}$, the size of any fractional (resp. integer) vertex packing is $\leq b$.

Path queries $P^k( ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$ and cycle queries $C^k( ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_1)$ are examples of classes of unbounded vertex packing.

▶ **Theorem 29.**

1. *If a family of Boolean CQs $\mathcal{Q}$ has unbounded integer vertex packing, the combined complexity of $(q, \mathcal{I})$ for $q \in \mathcal{Q}$ is NP-hard.*
2. *If a family of CQs $\mathcal{Q}$ has bounded fractional vertex packing, then for each $q \in \mathcal{Q}$, $(q, \mathcal{I})$ can be evaluated in poly-time combined complexity for any $\mathcal{I}$.*

The NP-hardness in this theorem follows by a reduction from 3-Coloring, whereas the poly-time algorithm uses the bound given by Atserias-Grohe-Marx [12, 5] in terms of the size of minimum fractional edge cover of the query, and the duality between minimum fractional edge cover and maximum fractional vertex packing. The formal proof of the above theorem will appear in the full version of the paper.

In this paper, we illustrate the properties with examples. Consider the family $S^k( ) = R(x_1, \ldots, x_k)$ for $k \geq 1$: this has vertex packing of size $= 1$ and therefore can be answered trivially in poly-time in combined complexity for any inequality pattern $\mathcal{I}$. On the other hand, the class of path queries P^k mentioned earlier has unbounded vertex packing (has a vertex packing of size $\approx \frac{k}{2}$), and therefore for certain set of inequalities (*e.g.*, when $G^{\mathcal{I}}$ is a complete graph), the query evaluation of $(P^k, \mathcal{I})$ is NP-hard in k. Similarly, the class $F^k( ) = R_1(x_1), R_2(x_2), \ldots, R_k(x_k)$ mentioned earlier has unbounded vertex packing, and is NP-hard in k with certain inequality patterns (see Proposition 26).

Theorem 29 is not a dichotomy or a characterization of easy CQs w.r.t. inequalities, since there is a gap between the maximum fractional and integer vertex packing.[8]

[8] For example, for the complete graph on k vertices, the maximum integer vertex packing is of size 1 whereas the maximum fractional vertex packing is of size $\frac{k}{2}$.

8 Conclusion

We studied the complexity of CQs with inequalities and compared the complexity of query answering with and without the inequality constraints. Several questions remain open: Is there a property that gives a dichotomy of query evaluation with inequalities both for the class of CQs, and for the class of CQs along with the inequality graphs? What can be said about unions of conjunctive queries (UCQ) and recursive datalog programs? Can our techniques be used as a black-box to extend any algorithm for CQs, *i.e.*, not necessarily based on SPJ query plans, to evaluate CQs with inequalities?

References

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

2 Foto Afrati, Chen Li, and Prasenjit Mitra. Answering queries using views with arithmetic comparisons. In *PODS*, pages 209–220, 2002.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

5 Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *FOCS*, pages 739–748, 2008.

6 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

7 Marc Demange and Dominique De Werra. On some coloring problems in grids. *Theor. Comput. Sci.*, 472:9–27, February 2013.

8 Arnaud Durand and Etienne Grandjean. The complexity of acyclic conjunctive queries revisited. *CoRR*, abs/cs/0605008, 2006.

9 Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, November 2002.

10 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. In *PODS*, pages 21–32, 1999.

11 M.H. Graham. On the universal relation. *Technical Report, University of Toronto, Ontario, Canada*, 1979.

12 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

13 Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997.

14 Phokion G. Kolaitis, David L. Martin, and Madhukar N. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *PODS*, pages 197–204, 1998.

15 Paraschos Koutris, Tova Milo, Sudeepa Roy, and Dan Suciu. Answering conjunctive queries with inequalities. *CoRR*, abs/1412.3869, 2014.

16 B. Monien. How to find long paths efficiently. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 239 – 254. North-Holland, 1985.

17 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 37–48, 2012.

18 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. In *PODS*, pages 12–19, 1997.

19 Neil Robertson and P.D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

20 Riccardo Rosati. The limits of querying ontologies. In *ICDT*, pages 164–178, 2007.

21 Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.*, 54(1):113–135, February 1997.

22 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 96–106, 2014.

23 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94. IEEE Computer Society, 1981.

24 C.T. Yu and M. Z. Ozsoyoglu. An algorithm for tree-query membership of a distributed query. In *COMPSAC*, pages 306–312, 1979.

25 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discrete Math.*, 10(2):209–222, 1997.

SQL's Three-Valued Logic and Certain Answers

Leonid Libkin

School of Informatics, University of Edinburgh

Abstract

SQL uses three-valued logic for evaluating queries on databases with nulls. The standard theoretical approach to evaluating queries on incomplete databases is to compute certain answers. While these two cannot coincide, due to a significant complexity mismatch, we can still ask whether the two schemes are related in any way. For instance, does SQL always produce answers we can be certain about?

This is not so: SQL's and certain answers semantics could be totally unrelated. We show, however, that a slight modification of the three-valued semantics for relational calculus queries can provide the required certainty guarantees. The key point of the new scheme is to fully utilize the three-valued semantics, and classify answers not into certain or non-certain, as was done before, but rather into certainly true, certainly false, or unknown. This yields relatively small changes to the evaluation procedure, which we consider at the level of both declarative (relational calculus) and procedural (relational algebra) queries. We also introduce a new notion of certain answers with nulls, which properly accounts for queries returning tuples containing null values.

1998 ACM Subject Classification H.2.4 Query Processing

Keywords and phrases Null values, incomplete information, query evaluation, three-valued logic, certain answers

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.94

1 Introduction

SQL's query evaluation engine uses three-valued logic when it comes to handling incomplete information: comparisons involving null values have the truth value *unknown* [7]. This results in a number of well known paradoxes. Consider, for instance, two relations R and S with a single numerical attribute A, and assume that S contains a single row with a null value in it. Then

```
select S.A from S where S.A <= 0 or S.A > 0
```
(1)

returns nothing despite the condition in the `where` clause being a tautology. This is because both `null <= 0` and `null > 0` evaluate to *unknown* and so does their disjunction. Worse yet, for the same reason, the query computing $R - S$:

```
select R.A from R where R.A not in (select S.A from S)
```
(2)

returns nothing if S contains a single null, no matter what R is, telling us that might well have $|R| > |S|$ and $R - S = \emptyset$ at the same time.

However unintuitive these answers are (which led to very severe criticism of the design of null-related features of SQL [6, 7]) they at least seem not to give us any false positives. To understand what it means, we appeal to the standard theoretical notion of query answering in the presence of incompleteness, *certain answers* [1, 12]. Each incomplete database D has an associated *semantics* $[\![D]\!]$. We can think of $[\![D]\!]$ as the set of possible complete databases that D can represent, i.e., all databases obtained by substituting values for nulls. Then

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 94–109

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

certain answers contain tuples that will be in the answer to Q over all possible complete databases represented by D:

$$\mathsf{certain}(Q, D) \;=\; \bigcap\{Q(D') \mid D' \in [\![D]\!]\} \tag{3}$$

How does SQL evaluation of queries relate to certain answers? There is a simple argument that they cannot coincide for relational calculus queries: SQL's evaluation is tractable (*very* tractable, in fact, of AC^0 data complexity), but data complexity of certain answers is intractable: at least coNP-complete for commonly considered semantics [2]. Examples (1) and (2) seem to suggest that we at least get a subset of certain answers, but this is not the case: false positives are possible. Consider the query:

```
select R.A from R
where R.A not in (select R1.A from R R1                    (4)
                  where R1.A not in (select * from S))
```

expressing $R - (R - S)$ and a database $R = \{1\}$ and $S = \{\bot\}$. SQL's evaluation results in $\{1\}$. At the same time the certain answer is empty: if $\bot$ is interpreted as any value other than 1, the query produces $\emptyset$.

Can we remedy this? Clearly we cannot modify SQL's evaluation rules to generate certain answers due to the complexity mismatch. So the best we can hope for is a *reasonable approximation without false positives*. The idea itself is not new: in fact for the first time it was expressed in [22], even before complexity bounds for certain answers were known. Despite this, we do not yet have such approximation schemes for SQL query evaluation. Providing them is our goal here. Specifically, we want to achieve the following:

- find query answers fast, without a significant modification of the existing evaluation techniques, and at the same time
- guarantee that no false positives occur, i.e., every returned tuple is a certain answer.

We achieve this by providing a small modification to the three-valued logic approach of SQL that restores correctness guarantees: query evaluation no longer produces false positives, and all returned results are guaranteed to be certain answers.

To understand the idea of the modification, notice that SQL's query evaluation actually *mixes* three- and two-valued logic. Three-valued logic is used to evaluate conditions, but then query results return only those tuples for which conditions evaluate to *true*, effectively collapsing *unknown* and *false*. This works fine for positive queries, but once negation, especially negation in subqueries (e.g., `not in` or `not exists`) enters the picture, we have a problem, as it flips truth values. Now *true* flips to *false*, but both *unknown* and *false* (which were collapsed to one value when a subquery was evaluated) flip to *true*! This is how unintended tuples end up in the answer.

So to get correctness guarantees, we just need to be faithful to the three-valued approach. This means that there will be three possible outcomes for each candidate answer tuple: it can be either

- certainly in the answer (truth value *true*); or
- certainly *not* in the answer (truth value *false*); or
- possibly in the answer, or possibly not (truth value *unknown*).

The second modification that we need is using *marked*, or *naïve* nulls [1, 12] in tables. Such nulls can appear multiple times in tables, and they are often required by applications such as data integration and exchange [3, 13]. In fact they have already been implemented in

connection with such applications [11, 19]. Generally, SQL's nulls can be modeled with naïve nulls, simply by forbidding repetition. The reason we need marked nulls is twofold. Firstly, we want to produce more general results. Secondly, we need to overcome an additional (and quite unreasonable) deficiency of SQL's handling of nulls: even comparing whether a null value equals *itself* produces truth value *unknown*. Indeed, consider a table `T(A,B)` with a single tuple `(1,null)` and a query `select T1.A from T T1, T T2 where T1.A=T2.A and T1.B=T2.B`, i.e., $\pi_A(T \cap T)$. Instead of the expected 1, it gives the empty result, as comparing a value with itself does not evaluate to true.

We remark that using the multi-valued approach has proved very useful in two closely related areas: model-checking [4, 10], and knowledge representation [14, 18]. In fact the procedure of [14] that uses three-valued reasoning with knowledge bases is similar in spirit with the modification of SQL query evaluation that we propose (although the technical details of our procedure are quite different from [14]), and its modifications to achieve tractable reasoning [18] relied on database query evaluation techniques. In the database field the three-valued approach has, by and large, belonged to the practice rather than the theory.

Organization. In Section 2 we present basic definitions. Section 3 describes the evaluation procedure for relational calculus and SQL's three-valued approach in the presence of nulls. Section 4 presents the modified evaluation procedure and states its correctness. In Section 5 we prove a generalization of that result, relying on a new notion of certain answers with nulls. This generalization properly accounts for all three possible outcomes of query evaluation (certainly true, certainly false, unknown). In Section 6 we look at certainty guarantees for relational algebra queries. Concluding remarks are in Section 7. Due to space limitations, only proof sketches are presented here; complete proofs are available in the full version.

2 Preliminaries

Incomplete databases. We begin with some standard definitions [1, 12]. Incomplete databases are populated by *constants* and *nulls*. The sets of constants and nulls are countably infinite sets denoted by Const and Null respectively. Nulls are denoted by $\bot$, sometimes with sub- or superscripts.

A relational schema (vocabulary) is a set of relation names with associated arities. An incomplete relational instance D assigns to each k-ary relation symbol S from the vocabulary a k-ary relation S^D over $\mathsf{Const} \cup \mathsf{Null}$, i.e., a finite subset of $(\mathsf{Const} \cup \mathsf{Null})^k$. When the instance is clear from the context we shall write S, rather than S^D, for the relation itself as well.

The sets of constants and nulls that occur in D are denoted by $\mathsf{Const}(D)$ and $\mathsf{Null}(D)$. If $\mathsf{Null}(D)$ is empty, we refer to D as *complete*. That is, complete databases are those without nulls. The *active domain* of D is $\mathrm{adom}(D) = \mathsf{Const}(D) \cup \mathsf{Null}(D)$.

Homomorphisms, valuations, and semantics. Given two relational structures D and D', a homomorphism $h : D \to D'$ is a map from the active domain of D to the active domain of D' such that:

1. for every relation symbol S, if a tuple $\bar{u}$ is in relation S in D, then the tuple $h(\bar{u})$ is in the relation S in D'; and
2. $h(c) = c$ for every $c \in \mathsf{Const}(D)$.

By $h(D)$ we denote the image of D, i.e., the set of all tuples $S(h(\bar{u}))$ where $S(\bar{u})$ is in D. If $h : D \to D'$ is a homomorphism, then $h(D)$ is a subinstance of D'.

A homomorphism $h : D \to D'$ is called a *valuation* if $h(x)$ is a constant for every $x \in \text{adom}(D)$; in other words, it provides a valuation of nulls as constant values. If h is a valuation, then $h(D)$ is complete. We now define the semantics of incomplete databases by means of valuations:

$$[\![D]\!] \;=\; \{h(D) \mid h \text{ is a valuation}\}.$$

This is often referred to as the closed-world assumption, or CWA semantics of incompleteness [12, 21]. Another common semantics uses the open-world assumption, or OWA, and allows adding complete tuples to $h(D)$. In the study of incompleteness, the closed-world semantics is a bit more common [1, 2, 12] since it is better behaved. We shall offer some comments on the OWA semantics in Section 5.2.

Query languages. As our basic query languages we consider relational calculus and its fragments. Relational calculus has exactly the power of *first-order logic*, or FO. Its formulae are built from relational atoms $R(\bar{x})$, equality atoms $x = y$, by closing them under conjunction $\wedge$, disjunction $\vee$, negation $\neg$, existential $\exists$ and universal $\forall$ quantifiers. If $\bar{x}$ is the list of free variables of a formula φ, we write $\varphi(\bar{x})$ to indicate this. We write $|\bar{x}|$ for the length of $\bar{x}$.

Conjunctive queries (CQs, also known as select-project-join queries) are defined as queries expressed in the $\exists, \wedge$-fragment of FO. The class UCQ of *unions of conjunctive queries* is the class of formulae of the form $\varphi_1 \vee \ldots \vee \varphi_m$, where each φ_i is a conjunctive query. In terms of its expressive power, this is the existential-positive fragment of FO, i.e., the $\exists, \vee, \wedge$-fragment.

We shall use relational algebra, the procedural language equivalent to FO, that has operations of selection σ, projection π, cartesian product $\times$, union $\cup$, and difference $-$. We use the unnamed perspective of relational algebra which does not require the renaming operator [1] (more on this in Section 6, where we shall add explicit intersection to relational algebra). The fragment without the difference operator is referred to as positive relational algebra; it has the same expressiveness as existential positive formulae (and thus unions of conjunctive queries).

3 Evaluation procedures for FO queries

We shall look at different query evaluation procedures. Each such procedure Eval will take a query (an FO formula) $\varphi(\bar{x})$, a database D, and an assignment ν of values to the free variables $\bar{x}$. The output $\mathsf{Eval}(\varphi, D, \nu)$ is a *truth value*. For the standard Boolean logic, the domain of truth values is $\{0, 1\}$, with 0 meaning *false* and 1 meaning *true*. For three-valued logic, the domain is $\{0, \frac{1}{2}, 1\}$, with $\frac{1}{2}$ interpreted as *unknown*.

An assignment ν maps each free variable to an element of $\text{adom}(D)$. Note that such an element could be a constant or a null; assignments thus are *not* valuations. We write $\nu[a/x]$ for the assignment that changes ν by mapping x to a. Also, given a tuple $\bar{x} = (x_1, \ldots, x_n)$ of free variables, and a tuple $\bar{a} = (a_1, \ldots, a_n)$, we write simply $\mathsf{Eval}(\varphi, D, \bar{a})$ if the assignment ν is such that $\nu(x_i) = a_i$ for all $i \leq n$.

Given an evaluation procedure Eval, the outcome of query evaluation for $\varphi(\bar{x})$ with $|\bar{x}| = k$ is

$$\mathsf{Eval}(\varphi, D) \;=\; \{\bar{a} \in \text{adom}(D)^k \mid \mathsf{Eval}(\varphi, D, \bar{a}) = 1\}\,.$$

For all of the evaluation procedures that we use (except two in Subsection 5.2), the

evaluation of the Boolean connectives and quantifiers is completely standard:

$$\begin{array}{rcl} \mathsf{Eval}(\varphi \vee \psi, D, \nu) & = & \max(\mathsf{Eval}(\varphi, D, \nu), \mathsf{Eval}(\psi, D, \nu)) \\ \mathsf{Eval}(\varphi \wedge \psi, D, \nu) & = & \min(\mathsf{Eval}(\varphi, D, \nu), \mathsf{Eval}(\psi, D, \nu)) \\ \mathsf{Eval}(\neg\varphi, D, \nu) & = & 1 - \mathsf{Eval}(\varphi, D, \nu) \\ \mathsf{Eval}(\exists x\varphi, D, \nu) & = & \max\{\mathsf{Eval}(\varphi, D, \nu[a/x]) \mid a \in \mathrm{adom}(D)\} \\ \mathsf{Eval}(\forall x\varphi, D, \nu) & = & \min\{\mathsf{Eval}(\varphi, D, \nu[a/x]) \mid a \in \mathrm{adom}(D)\} \end{array} \tag{5}$$

Thus, from now we only explain the valuation of *atomic* formulae $R(\bar{x})$ and equalities $x = y$. The classical FO evaluation gives us the procedure $\mathsf{Eval}_{\mathsf{FO}}$ with the range $\{0, 1\}$ defined by (5) and:

$$\mathsf{Eval}_{\mathsf{FO}}(R(\bar{x}), D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(\bar{x}) \in R^D \\ 0 & \text{if } \nu(\bar{x}) \notin R^D \end{cases}$$

$$\mathsf{Eval}_{\mathsf{FO}}(x = y, D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(x) = \nu(y) \\ 0 & \text{if } \nu(x) \neq \nu(y) \end{cases}$$

SQL's evaluation has $\{0, \frac{1}{2}, 1\}$ as the range of values. Again it uses rules (5), and the rule for $\mathsf{Eval}_{\mathsf{SQL}}(R(\bar{x}), D, \nu)$ is exactly the same as for $\mathsf{Eval}_{\mathsf{FO}}$, but for equality atoms the rule differs:

$$\mathsf{Eval}_{\mathsf{SQL}}(x = y, D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(x) = \nu(y) \text{ and } \nu(x), \nu(y) \in \mathsf{Const} \\ 0 & \text{if } \nu(x) \neq \nu(y) \text{ and } \nu(x), \nu(y) \in \mathsf{Const} \\ \frac{1}{2} & \text{if } \nu(x) \in \mathsf{Null} \text{ or } \nu(y) \in \mathsf{Null} \end{cases}$$

Indeed, SQL's approach is to declare every comparison as *unknown* if a null is involved. Note that over complete databases, $\mathsf{Eval}_{\mathsf{FO}}$ and $\mathsf{Eval}_{\mathsf{SQL}}$ coincide. Also, over incomplete databases, $\mathsf{Eval}_{\mathsf{FO}}$ is usually referred to as naïve evaluation [1, 12].

How do these relate to certain answers? We now examine FO and SQL evaluation. But first note that the definition (3) ensures that only tuples of constants are present in certain answers. There is no such restriction on the standard evaluation procedures. So to do a fair comparison we only compare sets of constant tuples returned by evaluation procedures (this will be relaxed later in the paper).

▶ **Definition 1.** *Given a class $\mathcal{Q}$ of queries, an evaluation procedure* Eval *has* certainty guarantees for $\mathcal{Q}$ *if for every query $\varphi(\bar{x}) \in \mathcal{Q}$, every database D, and every tuple $\bar{a}$ of constants with $|\bar{a}| = |\bar{x}|$, we have*

$$\bar{a} \in \mathsf{Eval}(\varphi, D) \;\Rightarrow\; \bar{a} \in \mathsf{certain}(\varphi, D).$$

In other words,

$$\mathsf{Eval}(\varphi, D) \cap \mathsf{Const}^{|\bar{x}|} \;\subseteq\; \mathsf{certain}(\varphi, D).$$

Certain answers and $\mathsf{Eval}_{\mathsf{FO}}$. The first observation is immediate:

$$\mathsf{certain}(\varphi, D) \;\subseteq\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, D).$$

The converse in general is not true, we can have $\mathsf{Eval}_{\mathsf{FO}}(\varphi(\bar{x}), D) \cap \mathsf{Const}^{|\bar{x}|} \not\subseteq \mathsf{certain}(\varphi, D)$. Consider for instance $\varphi(x) = R(x) \wedge \neg S(x)$ expressing the difference of R and S. Let D contain $R^D = \{1\}$ and $S^D = \{\bot\}$; then $\mathsf{Eval}(\varphi, D) = \{1\}$ while $\mathsf{certain}(\varphi, D) = \emptyset$.

However, sometimes certainty guarantees can be established. It has long been known [12] that we get them by excluding universal quantification and negation from first-order logic: $\mathsf{Eval}_{\mathsf{FO}}$ has certainty guarantees for the class UCQ. This was recently extended in [8] which showed that the same is true for queries from a rather significant expansion of the class UCQ, by adding universal quantification and a limited form of implication. More precisely, we look at the class $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ defined as follows:

- atomic formulae $R(\bar{x})$ and $x = y$ are in $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$;
- if $\varphi, \psi \in \mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ then so are $\varphi \vee \psi$ and $\varphi \wedge \psi$;
- if $\varphi \in \mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ then so are $\exists x \varphi$ and $\forall x \varphi$;
- if $\varphi(\bar{x}, \bar{y})$ is in $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$, then so is $\forall \bar{x}\ (R(\bar{x}) \to \varphi(\bar{x}, \bar{y}))$, where R is a relation symbol in the schema, and $\bar{x}$ does not have a repetition of variables.

Then $\mathsf{Eval}_{\mathsf{FO}}$ has certainty guarantees for $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ queries [8]. From the point of view of relational algebra, the class $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ corresponds to operations $\sigma, \pi, \cup, \times$ and the division operation $Q \div Q'$, where Q' is written in the $\pi, \cup, \times$-fragment of relational algebra, see [16].

Certain answers and $\mathsf{Eval}_{\mathsf{SQL}}$. How does SQL change things? Actually, it changes them for the *worse*: now there is no connection between $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D)$ and $\mathsf{certain}(\varphi, D)$ whatsoever. Indeed, we saw that for the query $\varphi(x) = R(x) \wedge \neg(R(x) \wedge \neg S(x))$ and database D with $R^D = \{1\}$ and $S^D = \{\bot\}$, the certain answer is empty while $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D) = \{1\}$, and for $\psi(x) = R(x) \wedge (S(x) \vee \neg S(x))$, the certain answer is $\{1\}$, while $\mathsf{Eval}_{\mathsf{SQL}}(\psi, D) = \emptyset$.

In a restricted case we provide correctness guarantees:

▶ **Proposition 2.** $\mathsf{Eval}_{\mathsf{SQL}}$ *has certainty guarantees for unions of conjunctive queries.*

Proof sketch. This follows from the fact for unions of conjunctive queries, $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D, \nu) = 1$ implies $\mathsf{Eval}_{\mathsf{FO}}(\varphi, D, \nu) = 1$ (shown by induction), and known results for FO evaluation for unions of conjunctive queries [12]. ◀

4 Evaluation procedures with certainty guarantees

We now introduce an evaluation procedure that comes with certainty guarantees for *all* relational calculus queries. For that, we have to explain what is wrong with FO and SQL evaluation procedures shown above, particularly for evaluation of atomic formulae.

Atomic relational formulae $R(\bar{x})$. For both SQL and FO, one simply checks, for a given assignment ν, whether $\nu(\bar{x})$ belongs to R. However, returning 0 if $\nu(\bar{x}) \notin R$ is too strong if we view 0 as saying that the tuple certainly *cannot* belong to R.

Indeed, consider $R = \{(\bot_1, 1), (2, \bot_2)\}$ and let ν be the identity (recall that the range of ν is the whole active domain). Consider a tuple $\bar{x} = (\bot_1, \bot_2)$. It is not in R, but can it be in R under some valuation h? Of course it can: if $h(\bot_1) = 2$ and $h(\bot_2) = 1$, then $h(\bar{x}) = (2, 1)$ and $h(R) = \{(2, 1)\}$, i.e., $h(\bar{x}) \in h(R)$. On the other hand, if $h'(\bot_1) = 1$ and $h'(\bot_2) = 2$, then $h'(\bar{x}) = (1, 2)$ and $h'(R) = \{(1, 1), (2, 2)\}$, so $h'(\bar{x}) \notin h'(R)$. Thus, the correct value for evaluating the membership of $\bar{x}$ in R seems to be $\frac{1}{2}$, not 0. Value 0 should be reserved for cases when no valuation h makes $h(\bar{x}) \in h(R)$ possible.

The $\mathsf{Eval}_{\mathsf{FO}}$ and $\mathsf{Eval}_{\mathsf{SQL}}$ procedures return 0 too eagerly, and this becomes a problem when negation is applied to a formula, as 0 becomes a 1, and suddenly we have a false positive answer that in fact is not certain at all. If the value is kept at $\frac{1}{2}$, applying negation still results in $1 - \frac{1}{2} = \frac{1}{2}$, and thus no false 'certain answers' appear.

Equality formulae $x = y$ FO evaluation results in 0 if $\nu(x)$ and $\nu(y)$ are different nulls, but they could still be mapped to the same constant, so the right value should be $\frac{1}{2}$, not 0. On the other hand, SQL evaluation produces $\frac{1}{2}$ if one of $\nu(x)$ or $\nu(y)$ is a null. But if we know $\nu(x) = \nu(y)$, then for every valuation h we will have $h(\nu(x)) = h(\nu(y))$, so the evaluation procedure must return 1 and not $\frac{1}{2}$ in this case, or else it will miss some certain answers.

Now with this in mind, we introduce a proper *3-valued evaluation procedure* $\mathsf{Eval}_{3\mathsf{v}}$. For this, we need one additional concept. Given two tuples $\bar{t}_1$ and $\bar{t}_2$ of the same length over $\mathsf{Const} \cup \mathsf{Null}$, we say that they *unify* if there is a homomorphism h such that $h(\bar{t}_1) = h(\bar{t}_2)$. We then write $\bar{t}_1 \Uparrow \bar{t}_2$.

It is easy to see that we can define $\bar{t}_1 \Uparrow \bar{t}_2$ by asking for a valuation h so that $h(\bar{t}_1) = h(\bar{t}_2)$. By classical results on unification, it is known that $\bar{t}_1 \Uparrow \bar{t}_2$ can be tested in linear time [20].

Now the evaluation procedure is as follows. It uses rules (5) and the following rules for atomic formulae:

$$\mathsf{Eval}_{3\mathsf{v}}(R(\bar{x}), D, \nu) = \begin{cases} 1 & \text{if } \nu(\bar{x}) \in R^D \\ 0 & \text{if there is no } \bar{t} \in R^D \text{ such that } \nu(\bar{x}) \Uparrow \bar{t} \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

$$\mathsf{Eval}_{3\mathsf{v}}(x = y, D, \nu) = \begin{cases} 1 & \text{if } \nu(x) = \nu(y) \\ 0 & \text{if } \nu(x), \nu(y) \in \mathsf{Const} \text{ and } \nu(x) \neq \nu(y) \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

Coming back to the example in the beginning of the section, if we have a database D with $R^D = \{(\bot_1, 1), (2, \bot_2)\}$ and $\nu : (x, y) \mapsto (\bot_1, \bot_2)$, then $\mathsf{Eval}_{3\mathsf{v}}(R(x, y), D, \nu) = \frac{1}{2}$. Indeed, even though $(\bot_1, \bot_2)$ is not in R^D, there are valuations h so that $h(\bot_1, \bot_2) \in h(R^D)$. On the other hand, no valuation h makes $(1, 2) \in h(R^D)$ possible, so for $\nu' : (x, y) \mapsto (1, 2)$ we have $\mathsf{Eval}_{3\mathsf{v}}(R(x, y), D, \nu') = 0$.

These modifications turn out to be sufficient to ensure certainty guarantees for all relational calculus queries.

▶ **Theorem 3.** $\mathsf{Eval}_{3\mathsf{v}}$ *has certainty guarantees for all* FO *queries.*

As an example, consider again query (4), or $\varphi(x) = R(x) \wedge \neg(R(x) \wedge \neg S(x))$ over D with $R^D = \{1\}$ and $S^D = \{\bot\}$. It produced a false positive since $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D) = \{1\}$ but the certain answer is empty. But now we have $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D) = \emptyset$. Indeed, we had $\mathsf{Eval}_{\mathsf{SQL}}(R(x) \wedge \neg S(x), D, 1) = 0$, and thus $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D, 1) = 1$, but now $\mathsf{Eval}_{3\mathsf{v}}(R(x) \wedge \neg S(x), D, 1) = \frac{1}{2}$ and hence $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, 1) = \frac{1}{2}$.

As another remark, note that the result of $\mathsf{Eval}_{3\mathsf{v}}$ need not be contained in the result of $\mathsf{Eval}_{\mathsf{SQL}}$, i.e., $\mathsf{Eval}_{3\mathsf{v}}$ can produce results that SQL evaluation misses. For instance, given a database D with $R^D = \{(\bot, \bot)\}$ and a query $\psi = \exists x, y\ R(x, y) \wedge x = y$, one can easily check that $\mathsf{Eval}_{3\mathsf{v}}(\psi, D, \nu) = 1$ (for the only possible valuation over a singleton active domain), while $\mathsf{Eval}_{\mathsf{SQL}}(\psi, D, \nu) = \frac{1}{2}$.

Theorem 3 will be a consequence of a more general result (Theorem 6), that does not restrict us to constant tuples. But for this we first need to define certain answers with nulls.

5 Certain answers with nulls

While the definition of certain answers (3) has been with us for 30+ years [17], recently it has been questioned [15, 16]. One of the problems with this definition is that it only returns tuples containing constants. Consider a database D with a relation $R^D = \{(1,2),(3,\bot)\}$ and a query $\psi(x,y) = R(x,y)$. Then $\mathsf{certain}(\psi, D) = \{(1,2)\}$ but intuitively we should return the entire relation R^D since we are certain its tuples are in the answer. The reason we are certain about it is that for every valuation h, the tuple $(3, h(\bot))$ is in $h(D)$. We turn this reasoning into a definition.

▶ **Definition 4.** Given an incomplete database D and a k-ary query Q defined over complete databases, *certain answers with nulls* $\mathsf{certain}_\bot(Q, D)$ is defined as the set of all tuples $\bar{u} \in \mathrm{adom}(D)^k$ such that $h(\bar{u}) \in Q(h(D))$ for all valuations h.

For instance, if a query is given by an FO formula with k free variables, then

$$\mathsf{certain}_\bot(\varphi, D) \;=\; \{\bar{u} \in \mathrm{adom}(D)^k \;\mid\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1 \text{ for every valuation } h\}.$$

Returning to the above example, we have $\mathsf{certain}_\bot(\psi, D) = \{(1,2),(3,\bot)\}$, so the tuple $(3,\bot)$ is no longer omitted.

We now summarize properties of certain answers with nulls. The usual certain answers can be obtained from certain answers with nulls by dropping tuples containing nulls, and certain answers with nulls are always contained in the result of the simple FO evaluation of formulae. Sometimes, but not always, they may coincide with the result of such an evaluation.

Formally, we have the following.

▶ **Proposition 5.** *The following hold:*

- $\mathsf{certain}(\varphi(\bar{x}), D) = \mathsf{certain}_\bot(\varphi, D) \cap \mathsf{Const}^{|\bar{x}|}$.
- $\mathsf{certain}_\bot(\varphi, D) \subseteq \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$ *for every* FO *query* φ.
- *If* $\varphi \in \mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$, *then* $\mathsf{certain}_\bot(\varphi, D) = \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$.
- *There exist* FO *queries* φ *so that* $\mathsf{certain}_\bot(\varphi, D) \neq \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$.

We can now state a more general description of the evaluation procedure $\mathsf{Eval}_{3\mathsf{v}}$: the output value 1 guarantees that a tuple belongs to certain answers with nulls for query φ, the output value 0 guarantees that it belongs to certain answers with nulls for the negation $\neg\varphi$, and output value $\frac{1}{2}$ comes with no guarantees.

▶ **Theorem 6.** *For every* FO *query* $\varphi(\bar{x})$ *and every database* D,

$$\mathsf{Eval}_{3\mathsf{v}}(\varphi, D) \subseteq \mathsf{certain}_\bot(\varphi, D).$$

Moreover, if $\bar{a} \in \mathit{adom}(D)^{|\bar{x}|}$ *and* $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \bar{a}) = 0$, *then* $\bar{a} \in \mathsf{certain}_\bot(\neg\varphi, D)$.

Theorem 3 is now an immediate corollary: if $\bar{a}$ is a tuple of constants and $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \bar{a}) = 1$, then by Theorem 6, $\bar{a} \in \mathsf{certain}_\bot(\varphi, D)$, and by Proposition 5, $\bar{a} \in \mathsf{certain}(\varphi, D)$.

Proof sketch. We first show an auxiliary result that $\bar{u} \in \mathsf{certain}_\bot(\varphi, D)$ if and only if $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1$ for every homomorphism h (rather than every valuation h). Then the theorem is a consequence of the following:

$$\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 1 \;\Rightarrow\; \forall \text{ homomorphism } h:\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 1 \qquad (*)$$

$$\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 0 \;\Rightarrow\; \forall \text{ homomorphism } h:\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 0 \qquad (**)$$

This is shown by induction on φ; we provide the proof for the case of atomic formulae (for which $\mathsf{Eval}_{3\mathsf{v}}$ differs from $\mathsf{Eval}_{\mathsf{FO}}$) here.

If $\varphi(\bar{x})$ is a relational atom $R(\bar{x})$, then:

(*) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 1$ then $\nu(\bar{x}) \in R^D$; in particular, $h(\nu(\bar{x})) \in h(R^D)$ for every homomorphism h, showing $h(\nu(\bar{x})) \in \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D))$.

(**) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 0$ then for each tuple $\bar{t} \in R^D$ we have that $\nu(\bar{x}) \Uparrow \bar{t}$ does not hold. Thus for each homomorphism h, and each tuple $\bar{t} \in R^D$, we have $h(\nu(\bar{x})) \neq h(\bar{t})$. This means that $h(\bar{x}) \notin h(R^D)$, and thus for each homomorphism h we have $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 0$.

If $\varphi(x, y)$ is an equational atom $x = y$, then:

(*) If $\mathsf{Eval}_{3\mathsf{v}}(x = y, D, \nu) = 1$ then $\nu(x) = \nu(y)$, and thus for for every homomorphism h, we have $h(\nu(x)) = h(\nu(y))$; in particular, $\mathsf{Eval}_{\mathsf{FO}}(x = y, h(D), \nu) = 1$.

(**) If $\mathsf{Eval}_{3\mathsf{v}}(x = y, D, \nu) = 0$, then both $\nu(x)$ and $\nu(y)$ are constants and $\nu(x) \neq \nu(y)$. Since they are constants, every homomorphism leaves them intact, and thus $\mathsf{Eval}_{\mathsf{FO}}(x = y, h(D), \nu) = 0$. ◀

Another corollary says that we can use $\mathsf{Eval}_{3\mathsf{v}}$ to find *overapproximations* of certain answers:

▶ **Corollary 7.** *For every* FO *query* $\varphi(\bar{x})$ *we have*

$$\mathsf{certain}_{\perp}(\varphi, D) \;\subseteq\; adom(D)^{|\bar{x}|} - \mathsf{Eval}_{3\mathsf{v}}(\neg\varphi, D).$$

As for the complexity of the procedure, one can easily show the following.

▶ **Proposition 8.** *For each relational vocabulary* σ *and* $\alpha \in \{0, \frac{1}{2}, 1\}$*, from every* FO *query* $\varphi(\bar{x})$ *one can compute* FO *queries* $\varphi^{\alpha}(\bar{x})$ *in the vocabulary that extends* σ *with a unary predicate* $\mathsf{const}(\cdot)$ *interpreted as the set of constants, such that, for every database* D*,*

$$\{\bar{a} \in adom(D)^{|\bar{x}|} \mid \mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \bar{a}) = \alpha\} \quad = \quad \mathsf{Eval}_{\mathsf{FO}}(\varphi^{\alpha}, D).$$

Consequently, data complexity of computing $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D)$ *is in* AC^0*.*

This gives us a complexity argument showing that there are cases when $\mathsf{Eval}_{3\mathsf{v}}$ fails to produce *all* certain answers. A concrete example of strict containment of $\mathsf{Eval}_{3\mathsf{v}}$ in $\mathsf{certain}_{\perp}$ will be shown below in Section 5.1.

5.1 CQs and UCQs with inequalities

A common extension of conjunctive queries and their unions is by adding inequalities [1]. This is a very mild form of negation; essentially, we only allow negation to be applied to equality atoms. Instead of writing them as $\neg(x = y)$, it is common to use $x \neq y$ in formulae, and refer to them as inequality atoms. Then the $\exists, \wedge$-closure of relational, equality and inequality atoms is referred to as CQs with inequalities, and the $\exists, \wedge, \vee$-closure as *UCQs with inequalities*. This class of queries is denoted by $\mathsf{UCQ}^{\neq}$.

We now present a particularly easy evaluation procedure that correctly accounts for $\mathsf{Eval}_{3\mathsf{v}}$ producing value 1 for UCQs with inequalities, and thus gives us correctness guarantees for those queries. This procedure uses two-valued, rather than three-valued, logic and only one rule that separates it from $\mathsf{Eval}_{\mathsf{FO}}$. To understand it, note for an inequality atom $x \neq y$, FO

evaluation returns true if x and y are assigned different values – even if they are different nulls. But actually the evaluation of conditions such as $\bot_1 \neq \bot_2$ must be false, since $\bot_1$ and $\bot_2$ can be mapped, by a valuation, to the same element. For $\mathsf{UCQ}^{\neq}$, there is no risk with assigning *false* rather than *unknown*, since negation will never be applied further on. This lets us define the evaluation procedure for $\mathsf{UCQ}^{\neq}$ by adding the following explicit rule for $\neq$ formulae to the $\mathsf{Eval}_{\mathsf{FO}}$ rules:

$$\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(x \neq y, D, \nu) \quad = \quad \begin{cases} 1 & \text{if } \nu(x), \nu(y) \in \mathsf{Const} \text{ and } \nu(x) \neq \nu(y) \\ 0 & \text{otherwise} \end{cases}$$

This evaluation is particularly easy to implement in SQL with the usual `is not null` conditions in the `where` clause. And it has the desired correctness guarantees.

▶ **Theorem 9.** *For every* $\mathsf{UCQ}^{\neq}$ *query* φ*, we have*

$$\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi, D) \;=\; \mathsf{Eval}_{3\mathsf{v}}(\varphi, D) \;\subseteq\; \mathsf{certain}_{\bot}(\varphi, D).$$

In particular, $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}$ *has certainty guarantees for* $\mathsf{UCQ}^{\neq}$ *queries.*

Proof sketch. We prove, by induction on the formulae, that $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi, D, \nu) = 1$ iff $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 1$ for $\mathsf{UCQ}^{\neq}$ queries. ◀

One cannot capture $\mathsf{certain}_{\bot}(\varphi, D)$ precisely with the $\mathsf{UCQ}^{\neq}$ evaluation procedure. Indeed, consider the query $\psi = \exists x \exists y\ R(x, y) \wedge x \neq y$ and a database D with $R^D = \{(\bot, 1), (\bot, 2)\}$. One easily checks $\mathsf{certain}_{\bot}(\psi, D) = \mathsf{certain}(\psi, D) = \text{true}$ but at the same time $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\psi, D) = 0$. By Theorem 9, this also means that $\mathsf{Eval}_{3\mathsf{v}}(\psi, D)$ fails to capture $\mathsf{certain}_{\bot}(\varphi, D)$; this is the example promised at the end of the last section.

In fact there could be no polynomial-time evaluation procedure for finding certain answers for $\mathsf{UCQ}^{\neq}$ queries since they have coNP-complete data complexity, even without free variables. Indeed, suppose we have a graph $G = \langle V, E\rangle$ where the set of vertices is $\{a_1, \ldots, a_n\}$. Create a binary relation D_G with $\mathrm{adom}(D_G) = \{\bot_1, \ldots, \bot_n\}$ and pairs $(\bot_i, \bot_j)$ for every edge $(a_i, a_j) \in E$. Let $\varphi \in \mathsf{UCQ}^{\neq}$ be given by $\exists x\ D_G(x, x) \vee \exists x, y, z, u\ (x \neq y \wedge x \neq z \wedge x \neq u \wedge y \neq z \wedge y \neq u \wedge z \neq u)$. Then $\mathsf{certain}(\varphi, D_G)$ is true iff G is not 3-colorable.

5.2 Open world semantics

Another commonly used semantics of incompleteness is based on the *open-world assumption*, or owa [1, 12, 21]. Under this assumption, after applying a valuation h to a database, finitely many complete tuples can be added to it. That is,

$$[\![D]\!]_{\mathrm{owa}} \;=\; \{h(D) \cup D' \mid h \text{ is a valuation and } D' \text{ is complete}\}.$$

Certain answers under owa are defined as $\mathsf{certain}_{\mathrm{owa}}(Q, D) = \bigcap\{Q(D') \mid D' \in [\![D]\!]_{\mathrm{owa}}\}$.

The evaluation procedure $\mathsf{Eval}_{3\mathsf{v}}$ no longer has certainty guarantees under owa. To see this, consider D with relations $R^D = \{(1, 2)\}$ and $S^D = \{(\bot_1, 1), (2, \bot_2)\}$. Let $\varphi(x, y) = R(x, y) \wedge \neg S(x, y)$. Since the tuple $(1, 2)$ does not unify with either tuple in S^D, we have $(1, 2) \in \mathsf{Eval}_{3\mathsf{v}}(\varphi, D)$. However, under owa, it is not a certain answer: for instance, the database D' with $R^{D'} = \{(1, 2)\}$ and $S^{D'} = \{(1, 1), (2, 2), (1, 2)\}$ is in $[\![D]\!]_{\mathrm{owa}}$, and $\mathsf{Eval}_{\mathsf{FO}}(\varphi, D')$ is empty.

Thus, our question is whether the approach of $\mathsf{Eval}_{3\mathsf{v}}$, guaranteeing correctness for all FO queries under cwa, can be extended to owa. Of course there is always a trivial positive answer: the evaluation procedure that always returns 0 vacuously has correctness guarantees.

Since $[\![D]\!]_{\text{CWA}} \subseteq [\![D]\!]_{\text{OWA}}$, certain answers under OWA will be included in certain answers under CWA, so the question really is how much we eliminate from the latter so that the result is still meaningful, and provides certainty guarantees under OWA. Note also that finding certain answers under OWA is undecidable [2] (even for data complexity [9]) which ties our hands even more in terms of finding suitable approximations.

To understand the changes that need to be made under OWA, consider again relational atoms. For them, there is no way to assert with certainty that a tuple does not belong to a relation, since each relation can be expanded under OWA. Hence, the case when evaluation produces 0 must go.

Next, look at existential formulae. Again we cannot state with certainty that the result of evaluation of those is 0, as perhaps in some extension of the database there is a witness for the existential formula, so the lowest value for evaluating such a formula is $\frac{1}{2}$, not 0. Likewise, for universal formulae, one cannot state with certainty that the result of evaluation is 1, as it requires checking the universal conditions in all extensions of the database, which is an undecidable problem. Hence, the highest value in this case is $\frac{1}{2}$ and not 1.

This explains the three changes that we make for the evaluation procedure. The procedure $\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}$ has the range $\{0, \frac{1}{2}, 1\}$ and differs from $\mathsf{Eval}_{3\mathsf{v}}$ in three rules:

$$\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(R(\bar{x}), D, \nu) = \begin{cases} 1 & \text{if } \nu(\bar{x}) \in R^D \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

$$\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(\exists x\varphi, D, \nu) = \max\left\{\tfrac{1}{2},\ \max\{\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(\varphi, D, \nu[a/x]) \mid a \in \text{adom}(D)\}\right\}$$

$$\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(\forall x\varphi, D, \nu) = \min\left\{\tfrac{1}{2},\ \min\{\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(\varphi, D, \nu[a/x]) \mid a \in \text{adom}(D)\}\right\}$$

Note that this procedure is the only one that modifies rules (5). These modifications are sufficient for correctness under OWA.

▶ **Proposition 10.** *The evaluation algorithm* $\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}$ *has correctness guarantees under* OWA.

Returning to the example from the beginning of the subsection, note that the value of $\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(S(x,y), D, (1,2))$ is $\frac{1}{2}$ (for $\mathsf{Eval}_{3\mathsf{v}}$ it would have been 0), and thus the result $\mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(R(x,y) \wedge \neg S(x,y), D, (1,2))$ is $\frac{1}{2}$ as well; in particular, $(1,2) \notin \mathsf{Eval}_{3\mathsf{v}}^{\text{OWA}}(\varphi, D)$ while we had $(1,2) \in \mathsf{Eval}_{3\mathsf{v}}(\varphi, D)$.

A remark on equivalence of queries under $\mathsf{Eval}_{3\mathsf{v}}$. Under the usual FO semantics, called $\mathsf{Eval}_{\mathsf{FO}}$ here, we are used to a number of equivalences that are *not* necessarily true when $\mathsf{Eval}_{3\mathsf{v}}$ is used instead. Consider, for instance, a formula $\varphi(x) = \exists y\big(R(x,y) \wedge (y = 1 \vee y \neq 1)\big)$. Of course we expect it to be equivalent to $\varphi'(x) = \exists y R(x,y)$. However, under the three-valued semantics these are not equivalent: if $R^D = \{(1, \bot)\}$ and $\nu : x \mapsto 1$, then $\mathsf{Eval}_{\mathsf{FO}}(\varphi, D, \nu) = \mathsf{Eval}_{\mathsf{FO}}(\varphi', D, \nu) = 1 = \mathsf{Eval}_{3\mathsf{v}}(\varphi', D, \nu)$, but at the same time $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = \frac{1}{2}$. This point will be important for us in the next section, where we present an evaluation procedure of relational algebra for databases with nulls.

6 Evaluation procedure for relational algebra

Queries that get executed in a DBMS are procedural queries, in particular, in the relational case, they are written in relational algebra, or some of its extensions. We now present an algorithm that provides an evaluation with correctness guarantees for relational algebra expressions. Even though from the point of view of expressiveness, relational algebra is

equivalent to FO, the equivalence itself, established under the standard two-valued semantics, is not yet a guarantee that it will provide us with a desired evaluation procedure in the three-valued world.

To expand on this, note that by Proposition 8, for every FO query $\varphi(\bar{x})$ we have a relational algebra expression e_φ which has access to the extra predicate $\mathsf{const}(\cdot)$ so that e_φ faithfully implements $\mathsf{Eval}_{3v}(\varphi, \cdot)$. So it seems that starting with a relational algebra query Q, we could find an equivalent FO query φ_Q and then consider e_{φ_Q} to evaluate Q.

Reasoning of this sort, however, mixes the equivalence of FO and relational algebra (that is true with respect to the usual two-valued FO evaluation) with the three-valued evaluation. Still, from the equivalence of $\mathsf{Eval}_{\mathsf{FO}}(\varphi_Q, \cdot)$ and Q one can easily derive $e_{\varphi_Q}(D) = \mathsf{Eval}_{3v}(\varphi_Q, D) \subseteq \mathsf{certain}_\perp(Q, D)$, so we do in fact get correctness guarantees with this approach. Nonetheless, it not satisfactory for two reasons. First, the detour via translation into FO and back to algebra may produce unnecessarily complicated expressions. Second, this approach assumes a particular translation between relational algebra and FO (which of course is not unique), and the quality of the resulting query depends on that translation. For instance, we view expressions R and $\sigma_{A=1}(R) \cup \sigma_{A\neq 1}(R)$ as equivalent, but using the latter in e_{φ_Q} can miss some answers with certainty guarantees due to the presence of nulls.

The bottom line is that it is better to have a *direct* evaluation procedure for relational algebra that gives us correctness guarantees without going through both algebra-to-FO and FO-to-algebra translations.

In the two-valued world sound translations for relational algebra have been considered in the past [22]. Our goal is a bit different though as we have to provide specific correctness guarantees, and relate them to SQL's way of evaluating queries; in fact we shall produce approximations for sets of tuples on which Eval_{3v} returns 1 and 0.

We now explain the procedure for correct evaluation of relational algebra queries. First, recall the operations of relational algebra. These are selection σ, projection π, cartesian product $\times$, union $\cup$, intersection $\cap$, and difference $-$. To avoid the clutter, and in particular to avoid renaming, we use the *unnamed* perspective for presenting relational algebra [1], that is, for each expression returning an m-attribute relation, we simply assume that the names of those attributes are $\sharp 1, \ldots, \sharp m$. As conditions θ in selections, we use positive Boolean combinations of equalities and inequalities between attribute values and constants. For instance, $(\sharp 1 \neq \sharp 2) \vee (\sharp 3 = 1)$ is a condition that can be used in selection. Note that such conditions are closed under negation, simply by propagating it all the way to (in)equalities, so we shall also refer sometimes to conditions $\neg\theta$, meaning the result of such a propagation. We refer to this standard relational algebra as RA.

We also consider an extension called $\mathsf{RA}_{\mathsf{null}}$. In this extension, conditions θ are positive Boolean combinations of

- equalities and inequalities between attributes, and
- conditions $\mathsf{const}(\sharp n)$ and $\mathsf{null}(\sharp n)$ stating that the value of attribute $\sharp n$ is a constant or a null, respectively.

Our goal is to provide a translation $\mathsf{RA} \to \mathsf{RA}_{\mathsf{null}}$ that associates with each query Q of RA a query Q^+ of $\mathsf{RA}_{\mathsf{null}}$ such that $Q^+(D) \subseteq \mathsf{certain}_\perp(Q, D)$.

As noticed already, due to coNP-data complexity of $\mathsf{certain}_\perp(Q, D)$, we cannot hope for equality, so this correctness guarantee is the best we can count on.

We shall actually produce more. Let $\bar{Q}$ be the query that computes the complement of Q, i.e., for an n-ary Q, the result of $\bar{Q}(D)$ is $\mathrm{adom}(D)^n - Q(D)$. Then we actually provide a translation

$$Q \mapsto (Q^+, Q^-)$$

$$
\begin{array}{rclrcl}
R^+ & = & R & R^- & = & R^{\ominus} \\
(Q_1 \cup Q_2)^+ & = & Q_1^+ \cup Q_2^+ & (Q_1 \cup Q_2)^- & = & Q_1^- \cap Q_2^- \\
(Q_1 \cap Q_2)^+ & = & Q_1^+ \cap Q_2^+ & (Q_1 \cap Q_2)^- & = & Q_1^- \cup Q_2^- \\
(Q_1 - Q_2)^+ & = & Q_1^+ \cap Q_2^- & (Q_1 - Q_2)^- & = & Q_1^- \cup Q_2^+ \\
(\sigma_\theta(Q))^+ & = & \sigma_{\theta^*}(Q^+) & (\sigma_\theta(Q))^- & = & Q^- \cup \sigma_{(\neg\theta)^*}(\mathrm{adom}^{\mathrm{ar}(Q)}) \\
(Q_1 \times Q_2)^+ & = & Q_1^+ \times Q_2^+ & (Q_1 \times Q_2)^- & = & Q_1^- \times \mathrm{adom}^{\mathrm{ar}(Q_2)} \\
 & & & & & \cup\ \mathrm{adom}^{\mathrm{ar}(Q_1)} \times Q_2^- \\
(\pi_{\boldsymbol{\alpha}}(Q))^+ & = & \pi_{\boldsymbol{\alpha}}(Q^+) & (\pi_{\boldsymbol{\alpha}}(Q))^- & = & \pi_{\boldsymbol{\alpha}}(Q^-) - \pi_{\boldsymbol{\alpha}}(\mathrm{adom}^{\mathrm{ar}(Q)} - Q^-)
\end{array}
$$

Figure 1 Relational algebra translations.

of RA queries into a pair of $\mathsf{RA}_{\mathsf{null}}$ queries such that

$$Q^+(D) \subseteq \mathsf{certain}_\perp(Q, D) \quad \text{and} \quad Q^-(D) \subseteq \mathsf{certain}_\perp(\bar{Q}, D).$$

When this happens, we say that the translation $Q \mapsto (Q^+, Q^-)$ *provides correctness guarantees.*

Since $\mathsf{certain}_\perp(Q, D) \cap \mathsf{certain}_\perp(\bar{Q}, D) = \emptyset$, this also means that $Q^+(D) \cap Q^-(D) = \emptyset$. One can think of Q^+ and Q^- as analogs of finding tuples for which $\mathsf{Eval}_{3\mathsf{v}}$ produces 0 or 1. Everything that does not fall into the results of these two, is essentially 'unknowns'.

We now provide the translations. We need three auxiliary elements: a translation $\theta \mapsto \theta^*$ from RA conditions to $\mathsf{RA}_{\mathsf{null}}$ conditions, one RA query, and one $\mathsf{RA}_{\mathsf{null}}$ query. These are given as follows:

The translation $\theta \mapsto \theta^*$ is defined inductively. We assume that in conditions $\sharp n = \sharp m$ or $\sharp n \neq \sharp m$, attributes $\sharp n$ and $\sharp m$ are different (otherwise they are easily eliminated).

- If θ is $(\sharp n = \sharp m)$ or $(\sharp m = c)$, where c is a constant, then $\theta^* = \theta$.
- $(\sharp n \neq \sharp m)^* = (\sharp n \neq \sharp m) \wedge \mathsf{const}(\sharp n) \wedge \mathsf{const}(\sharp m)$.
- $(\sharp n \neq c)^* = (\sharp n \neq c) \wedge \mathsf{const}(\sharp n)$.
- $(\theta_1 \vee \theta_2)^* = \theta_1^* \vee \theta_2^*$.
- $(\theta_1 \wedge \theta_2)^* = \theta_1^* \wedge \theta_2^*$.

Active domain query. We use adom as an RA query that returns the active domain of a database; clearly it can be written as a $\pi, \cup$-query, that takes the union of all projections of all relations in the database.

Relative complement query. The *relative complement* of a k-ary relation R in database D is

$$R^{\ominus} = \{\bar{u} \in \mathrm{adom}(D)^k \mid \neg\exists \bar{t} \in R : \ \bar{u} \Uparrow \bar{t}\,\}.$$

It is not hard to see that $R^{\ominus}$ is expressible in $\mathsf{RA}_{\mathsf{null}}$. We show this formally in the proof of Theorem 11. In fact this is the only expression where conditions $\mathsf{null}(\sharp n)$ are used in selections.

With these, translations of relational algebra are given by inductive rules presented in Figure 1. We use abbreviation $\mathrm{ar}(Q)$ for the arity of Q, and $\boldsymbol{\alpha}$ refers to a list of attributes.

▶ **Theorem 11.** *The translation $Q \mapsto (Q^+, Q^-)$ in Figure 1 provides correctness guarantees.*

Proof sketch. Again, we show the following, by induction on relational algebra expressions:

$$\bar{u} \in Q^+(D) \;\Rightarrow\; \forall \text{ homomorphism } h:\; h(\bar{u}) \in Q(h(D)) \qquad (\checkmark)$$

$$\bar{u} \in Q^-(D) \;\Rightarrow\; \forall \text{ homomorphism } h:\; h(\bar{u}) \notin Q(h(D)) \qquad (\checkmark\checkmark)$$

We provide a couple of sample cases. Consider, for instance, the case when Q is R. Then $Q^- = R^{\ominus}$. Assume $\bar{u} \in R^{\ominus}$ and let h be a homomorphism. By definition, $\bar{u}$ does not unify with any of $\bar{t} \in R$, in particular, $h(\bar{u})$ cannot equal $h(\bar{t})$, thus implying $h(\bar{u}) \notin h(R)$.

Let $\theta = (\sharp n \neq \sharp m)$, and assume $Q = \sigma_\theta(Q_1)$ (so that $Q^+ = \sigma_{\theta^*}(Q_1^+)$). Suppose $\bar{u} \in \sigma_{\theta^*}(Q_1^+(D))$, and let h be a homomorphism. Since $\bar{u} \in Q_1^+(D)$, we see, by the hypothesis, that $h(\bar{u}) \in Q_1(h(D))$. Furthermore, since θ^* holds, we know that u_n and u_m, the nth and the mth components of $\bar{u}$, are constants, and $u_n \neq u_m$. This means $h(u_n) \neq h(u_m)$, proving $h(\bar{u}) \in \sigma_\theta(Q_1(h(D)))$. ◀

The translation in Figure 1 is not just one translation but rather a family of translations, due to the following observation. A translation can be viewed as a mapping $\mathcal{F}$ that assigns to each relational algebra operation ω (including nullary operations for base relations) two queries F_ω^+ and F_ω^-. These queries are simply the queries that appear on the right in the translation; for instance, for the translation scheme we used, $F_\cap^+$ is the intersection (since the result of $(Q_1 \cap Q_2)^+$ is the intersection of Q_1^+ and Q_2^+) and $F_\cap^-$ is the union (since the result of $(Q_1 \cap Q_2)^-$ is the union of Q_1^- and Q_2^-).

Such a mapping $\mathcal{F}$ results in a translation $Q \mapsto \mathcal{F}_Q^+, \mathcal{F}_Q^-$, where $\mathcal{F}_Q^+$ and $\mathcal{F}_Q^-$ are queries of the same type as Q (i.e., they operate on databases of the same schema and have the same arity). Intuitively, these are analogs of Q^+ and Q^- that we had for the translation in Figure 1.

Formally, they are defined as follows.

- If ω is a base relation R, then F_R^+ and F_R^- take no arguments and $\mathcal{F}_R^+ = F_R^+$ and $\mathcal{F}_R^- = F_R^-$.

  That is, F_R^+ and F_R^- are queries that give us certainly positive and certainly negative information about R.
- If ω is a unary operation (σ or π), then F_ω^+ and F_ω^- take two arguments and $\mathcal{F}_{\omega(Q)}^+ = F_\omega^+(\mathcal{F}_Q^+, \mathcal{F}_Q^-)$ and $\mathcal{F}_{\omega(Q)}^- = F_\omega^-(\mathcal{F}_Q^+, \mathcal{F}_Q^-)$.

  That is, if we already have queries $\mathcal{F}_Q^+$ and $\mathcal{F}_Q^-$ describing certainly positive and certainly negative answers for Q, the queries describing such answers for $\omega(Q)$ are obtained by applying F_ω^+ and F_ω^- to those.
- If ω is a binary operation ($\cup, \cap, -, \times$), then F_ω^+ and F_ω^- take four arguments and $\mathcal{F}_{\omega(Q_1,Q_2)}^+ = F_\omega^+(\mathcal{F}_{Q_1}^+, \mathcal{F}_{Q_2}^+, \mathcal{F}_{Q_1}^-, \mathcal{F}_{Q_2}^-)$ and $\mathcal{F}_{\omega(Q_1,Q_2)}^- = F_\omega^-(\mathcal{F}_{Q_1}^+, \mathcal{F}_{Q_2}^+, \mathcal{F}_{Q_1}^-, \mathcal{F}_{Q_2}^-)$.

  That is, if we already have queries $\mathcal{F}_{Q_i}^+$ and $\mathcal{F}_{Q_i}^-$ describing certainly positive and certainly negative answers for Q_i, with $i = 1, 2$, the queries describing such answers for $\omega(Q_1, Q_2)$ are again obtained by applying F_ω^+ and F_ω^- to those.

Given a translation $\mathcal{F}$ and another translation $\mathcal{G}$ that assigns to each operation ω queries G_ω^+ and G_ω^-, we say that $\mathcal{F}$ is *contained* in $\mathcal{G}$ if $F_\omega^+ \subseteq G_\omega^+$ and $F_\omega^- \subseteq G_\omega^-$, where $\subseteq$ refers to the usual query containment.

▶ **Proposition 12.** *Every translation that is contained in the translation of Figure 1 provides correctness guarantees.*

This proposition lets us adjust translations for the sake of efficiency without having to worry about correctness guarantees. For instance, consider the rule

$$(Q_1 \times Q_2)^- \;=\; Q_1^- \times \mathrm{adom}^{\mathrm{ar}(Q_2)} \;\cup\; \mathrm{adom}^{\mathrm{ar}(Q_1)} \times Q_2^-$$

in Figure 1. This results in a rather expensive query, as one needs to compute a power of the active domain. But we can replace it with the much simpler rule $(Q_1 \times Q_2)^- = Q_1^- \times Q_2^-$, since $Q_1^- \times Q_2^-$ is contained in the above query, giving us a more efficient query. Another possible replacement is of the rule

$$(\sigma_\theta(Q))^- \;=\; Q^- \cup \sigma_{(\neg\theta)^*}(\mathrm{adom}^{\mathrm{ar}(Q)})$$

that again requires computing the active domain with the very simple rule $(\sigma_\theta(Q))^- = Q^-$. In both cases the result is that the translated queries are significantly more efficient and they still guarantee correctness of the overall translation in the sense that they produce subsets of certain answers with nulls, or the usual certain answers if tuples with nulls are removed. There is a price to pay for the efficiency though: we can get fewer answers in the result. Hence one should decide how to resolve the efficiency vs the quality of approximation tradeoff.

Another corollary concerns positive relational algebra, even extended with inequalities, and it just follows from examining the basic translation of Figure 1. Define $\mathsf{PosRA}^{\neq}$ as the positive fragment of RA (i.e., $\sigma, \pi, \times, \cup$) where conditions in selections are allowed to use inequalities. In terms of its expressiveness, this fragment corresponds to $\mathsf{UCQ}^{\neq}$.

▶ **Corollary 13.** *Let Q be a $\mathsf{PosRA}^{\neq}$ query, and let Q^* be obtained from it by changing each selection condition θ to θ^*. Then, for every database D, we have $Q^*(D) \subseteq \mathsf{certain}_\perp(Q, D)$.*

7 Conclusions

We have shown that small changes to the 3-valued query evaluation used in SQL produce sound query answers, i.e., answers without false positives. We have presented such evaluation procedures at the levels of both relational calculus and algebra, and also specialized them for unions of conjunctive queries with inequalities.

The theoretical complexity of these procedures is very low, in fact it is as low as evaluating relational calculus and algebra themselves, in terms of data complexity. The next obvious step is to implement these algorithms to study their real-life applicability. As indicated at the end of the last section, our translations – especially at the procedural level – are really families of algorithms, with the efficiency vs quality of approximation tradeoff, so there is a lot to play with, to find those that provide a good combination of both. Another natural question is to consider other features of SQL. They include not only such common features as aggregation and grouping, but also derived operations of relational algebra that are used in implementation of SQL queries: for instance, the division operation for the implementation of some universal queries, or semi-joins and anti-joins that can be used for implementing subqueries.

Acknowledgment. I thank Cristina Sirangelo and the reviewers for their helpful comments and suggestions. Work partially supported by EPSRC grant J015377.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2 S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.
3 M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.
4 G. Bruns and P. Godefroid. Model checking with multi-valued logics. In *ICALP*, pages 281–293, 2004.
5 K. Compton. Some useful preservation theorems. *Journal of Symbolic Logic*, 48(2):427–440, 1983.
6 C. J. Date. *Database in Depth - Relational Theory for Practitioners*. O'Reilly, 2005.
7 C. J. Date and H. Darwen. *A Guide to the SQL Standard*. Addison-Wesley, 1996.
8 A. Gheerbrant, L. Libkin, and C. Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems*, 39(4): 34 (2014).
9 A. Gheerbrant, L. Libkin, and T. Tan. On the complexity of query answering over incomplete XML documents. In *ICDT*, pages 169–181, 2012.
10 A. Gurfinkel and M. Chechik. Multi-valued model checking via classical model checking. In *CONCUR*, pages 263–277, 2003.
11 L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD*, pages 805–810, 2005.
12 T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
13 M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, pages 233–246, 2002.
14 H. J. Levesque. A completeness result for reasoning with incomplete first-order knowledge bases. In *Principles of Knowledge Representation and Reasoning (KR)*, pages 14–23, 1998.
15 L. Libkin. Certain answers as objects and knowledge. In *Principles of Knowledge Representation and Reasoning (KR)*, 2014.
16 L. Libkin. Incomplete information: what went wrong and how to fix it. In *PODS*, pages 1–13, 2014.
17 W. Lipski. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):262–296, 1979.
18 Y. Liu and H. J. Levesque. A tractability result for reasoning with incomplete first-order knowledge bases. In *IJCAI*, pages 83–88, 2003.
19 B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an opensource tool for second-generation schema mapping and data exchange. *PVLDB*, 4(12):1438–1441, 2011.
20 M. Paterson and M. N. Wegman. Linear unification. *J. Comput. Syst. Sci.*, 16(2):158–167, 1978.
21 R. Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.
22 R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33(2):349–347, 1986.

A Trichotomy in the Complexity of Counting Answers to Conjunctive Queries

Hubie Chen[1] **and Stefan Mengel***[2]

1 **Universidad del País Vasco, E-20018 San Sebastián, Spain,** ***and*** **IKERBASQUE, Basque Foundation for Science, E-48011 Bilbao, Spain**
2 **LIX UMR 7161, École Polytechnique, France**

Abstract

Conjunctive queries are basic and heavily studied database queries; in relational algebra, they are the select-project-join queries. In this article, we study the fundamental problem of counting, given a conjunctive query and a relational database, the number of answers to the query on the database. In particular, we study the complexity of this problem relative to sets of conjunctive queries. We present a trichotomy theorem, which shows essentially that this problem on a set of conjunctive queries is either tractable, equivalent to the parameterized CLIQUE problem, or as hard as the parameterized counting CLIQUE problem; the criteria describing which of these situations occurs is simply stated, in terms of graph-theoretic conditions.

1998 ACM Subject Classification H.2.3 [Database Management]: Languages – Query languages, F.1.3 [Computation by Abstract Devices]: Complexity measures and classes, G.2.2 [Discrete Mathematics]: Graph Theory – Hypergraphs

Keywords and phrases database theory, query answering, conjunctive queries, counting complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.110

1 Introduction

Conjunctive queries are the most basic and most heavily studied database queries. They can be formalized logically as formulas consisting of a sequence of existentially quantified variables, followed by a conjunction of atomic formulas; in relational algebra, they are the *select-project-join* queries (see e.g. [1]). Ever since the landmark 1977 article of Chandra and Merlin [2], complexity-theoretic aspects of conjunctive queries have been a research subject of persistent and enduring interest which continues to the present day (as a sampling, we point to the works [13, 16, 10, 12, 18, 14, 17, 5]; see the discussions and references therein for more information). The problem of evaluating a Boolean (closed) conjunctive query on a relational database is equivalent to a number of well-known problems, including conjunctive query containment, the homomorphism problem on relational structures, and the constraint satisfaction problem [2, 13]. That this evaluation problem appears in many equivalent guises attests to the foundational and primal nature of this problem, and it has correspondingly been approached and studied from a wide variety of perspectives and motivations.

In this article, we study the fundamental problem of counting, given a conjunctive query and a relational database, the number of query answers, that is, the number of assignments that make the query true with respect to the database; we denote this problem by #CQ.

* Partially supported by a Qualcomm grant administered by École Polytechnique.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 110–126

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In addition to being a natural problem in its own right, let us remark that all practical query languages supported by database management systems have a counting operator. We study the complexity of #CQ relative to sets of conjunctive queries, that is, we study a problem family: each set of conjunctive queries gives rise to a restricted version of the general problem. Our objective is to determine on which sets of conjunctive queries #CQ is tractable, and more broadly, to understand the complexity behavior of the problem family at hand. Throughout, we assume that each considered set of conjunctive queries is of *bounded arity*, by which we mean that there is a constant bounding the arity of all relation symbols in all queries in the set.[1]

Surprisingly, despite the natural and basic character of the counting problem #CQ, the project of understanding its complexity behavior over varying sets of queries has been carried out in previous work for only two particular types of conjunctive queries. In the case of Boolean conjunctive queries, the problem #CQ specializes to the problem of deciding whether or not such a query evaluates to *true* or *false* on a database. A classification of sets of Boolean conjunctive queries was given by Grohe [12], showing essentially that this decision problem is either polynomial-time tractable, or is hard under a typical complexity-theoretic assumption from parameterized complexity, namely, that the parameterized Clique problem is not fixed-parameter tractable (see Theorem 9).[2] It is well-known that a conjunctive query can be naturally mapped to a relational structure (see Definition 7); the tractable sets of Grohe's classification are such queries whose corresponding structures have *cores* of *bounded treewidth*. The *core* of a structure $\mathbf{A}$ is, intuitively, the smallest structure that is (in a certain sense) equivalent to the structure $\mathbf{A}$, and *bounded treewidth* is a graph-theoretical condition that can intuitively be taken as a notion of tree similitude. Following Grohe's work, Dalmau and Jonsson [6] studied the case of quantifier-free conjunctive queries (which they phrase as the problem of counting homomorphisms between a given pair of relational structures). They proved that bounded treewidth is the property that determines polynomial-time tractability for this case; in contrast to Grohe's theorem, the statement of their classification (Theorem 10) does not refer to the notion of core, and is proved under the assumption that the *counting* version #Clique of the parameterized Clique problem is not fixed-parameter tractable.

In this article, we present a trichotomy theorem describing the complexity of the counting problem #CQ on each possible set of conjunctive queries. Our trichotomy theorem unifies, generalizes, and directly implies the two discussed prior classifications. This trichotomy yields that counting on a set of conjunctive queries is polynomial-time tractable, is interreducible with the Clique problem, or admits a reduction from and is thus as hard as the counting problem #Clique. In order to prove and state our trichotomy theorem, we work with a notion of *core* of a (structure associated with a) conjunctive query whose definition crucially takes into account which variables of the conjunctive query are free (Definition 12). We also use a notion of hypergraph of a conjunctive query whose vertices are the free variables of the query (Definition 21). The properties of a query set that determine which case of our trichotomy theorem applies are whether or not the cores have bounded treewidth, and whether or not the just-mentioned hypergraphs have bounded treewidth; these two conditions correspond, respectively, to the conditions that describe the dichotomies for the Boolean case

[1] It is known that when no such bound on the arity is assumed, the complexity of query evaluation can be highly sensitive to the representation of database relations [4]. In contrast, natural representations are equivalent under polynomial-time translation; thus, the study of bounded arity queries can be viewed as the investigation of the representation-independent case.

[2] This is equivalent to the assumption that the parameterized complexity class W[1] is not contained in the parameterized complexity class FPT, which is the phrasing that Grohe employs.

and the quantifier-free case. The proof of our trichotomy draws on recent work of Durand and Mengel [8], who presented a classification for the problem #CQ based on hypergraphs (see Theorem 11).

Note that it is readily verified that the classes of queries for which we show that #CQ is tractable are equivalent to those described in recent work by Greco and Scarcello [11]. In contrast to their work, we show that the promise version of #CQ is not only fixed-parameter tractable on these classes of queries but can even be solved in polynomial time. Moreover, and more importantly, we show that these classes are the *only* classes of queries for which #CQ can be solved efficiently because all other classes of queries are intractable under standard complexity assumptions.

In order to prove and present our trichotomy, we introduce a version of the *case complexity* framework [3] which is suitable for dealing with counting problems. Among other features, this framework facilitates the presentation of reductions between parameterized problems which are restricted in terms of the permitted parameters (or *slices*); this is the type of restriction we deal with here, as the parameter of an instance of #CQ is taken to be the query, and we consider #CQ with respect to various query sets. This framework also allows the straightforward derivation of complexity consequences as a function of the computability assumption placed on the query sets; witness the derivation of Theorems 23 and 24 from Theorem 22.

2 Preliminaries

For an integer $i \geq 1$, we define π_i to be the operator that, given a tuple, returns the value in the ith coordinate.

We assume that the reader is familiar with basic graph theoretic notions. In particular, we will use some very basic properties of treewidth, which can be found, for example, in [9, Chapter 11] or in [15, Section 2.3].

2.1 Structures, homomorphisms and cores

A relational vocabulary is defined to be a set of relation symbols $\tau := \{R_1, R_2, \ldots, R_\ell\}$ where each R_i has an arity r_i. A relational structure $\mathbf{A}$ over τ is a tuple $(A, R_1^{\mathbf{A}}, \ldots, R_\ell^{\mathbf{A}})$ where A is a set called the *domain* of $\mathbf{A}$ and $R_i^{\mathbf{A}} \subseteq A^{r_i}$ is a relation of arity r_i. All vocabularies and structures in this article are assumed to be relational. We assume each structure in this article to be *finite* in that it has a finite domain. We denote structures by the bold letters, $\mathbf{A}, \mathbf{B}, \ldots$, and their corresponding domains by $A, B, \ldots$.

We assume each class of structures in this article to be of bounded arity, that is, for each such class we assume there exists a constant $c \geq 1$ such that the arity of each relation of a structure in the class is at most c. Since in the bounded arity setting the sizes of all reasonable encodings of a structure are polynomially related, we do not fix a specific encoding but assume that all structures are encoded in any reasonable way.

▶ **Definition 1.** Let $\mathbf{A}$ and $\mathbf{B}$ be two structures over the same vocabulary τ. A *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ is a function $h : A \to B$ such that for each relation symbol $R \in \tau$ and each $t = (t_1, \ldots, t_\ell) \in R^{\mathbf{A}}$ we have $(h(t_1), \ldots, h(t_\ell)) \in R^{\mathbf{B}}$. A homomorphism h from $\mathbf{A}$ to $\mathbf{B}$ is called an *isomorphism* if h is bijective and h^{-1} is a homomorphism from $\mathbf{B}$ to $\mathbf{A}$; when such an isomorphism exists, we say that $\mathbf{A}$ and $\mathbf{B}$ are *isomorphic*. An isomorphism from a structure to itself is called an *automorphism*.

▶ **Definition 2.** Two structures $\mathbf{A}$ and $\mathbf{B}$ are *homomorphically equivalent* if there are homomorphisms from $\mathbf{A}$ to $\mathbf{B}$ and from $\mathbf{B}$ to $\mathbf{A}$.

A structure is a *core* if it is not homomorphically equivalent to a proper substructure of itself. A structure $\mathbf{B}$ is a *core of a structure* a structure $\mathbf{A}$ if $\mathbf{B}$ is a substructure of $\mathbf{A}$, $\mathbf{B}$ is homomorphically equivalent to $\mathbf{A}$, and $\mathbf{B}$ is a core.

We state two basic properties of cores of structures; due to the first, we will speak of *the* core of a structure instead of *a* core. The second seems to be folklore; a proof can be found, for example, in [9].

▶ **Lemma 3.** *Every structure* $\mathbf{A}$ *has at least one core. Furthermore, every two cores* $\mathbf{B}_1$ *and* $\mathbf{B}_2$ *of* $\mathbf{A}$ *are isomorphic.*

▶ **Lemma 4.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two homomorphically equivalent structures, and let* $\mathbf{A}'$ *and* $\mathbf{B}'$ *be cores of* $\mathbf{A}$ *and* $\mathbf{B}$*, respectively. Then* $\mathbf{A}'$ *and* $\mathbf{B}'$ *are isomorphic.*

2.2 Complexity theory background

Throughout, we use Σ to denote an alphabet over which strings are formed. All problems to be considered are viewed as counting problems. So, a *problem* is a mapping $Q : \Sigma^* \to \mathbb{N}$. We view decision problems as problems where, for each $x \in \Sigma^*$, it holds that $Q(x)$ is equal to 0 or 1. We use FP (as usual) to denote the class of problems (which, again, are mappings $\Sigma^* \to \mathbb{N}$) that can be computed in polynomial time. A *parameterization* is a mapping $\kappa : \Sigma^* \to \Sigma^*$. A parameterized problem is a pair (Q, κ) consisting of a problem Q and a parameterization κ. A partial function $T : \Sigma^* \to \mathbb{N}$ is *polynomial-multiplied* with respect to a parameterization κ if there exists a computable function $f : \Sigma^* \to \mathbb{N}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that, for each $x \in \mathsf{dom}(T)$, it holds that $T(x) \leq f(\kappa(x))p(|x|)$.

▶ **Definition 5.** Let $\kappa : \Sigma^* \to \Sigma^*$ be a parameterization. A partial mapping $r : \Sigma^* \to \Sigma^*$ is *FPT-computable* with respect to κ if there exist a polynomial-multiplied function $T : \Sigma^* \to \mathbb{N}$ (with respect to κ) with $\mathsf{dom}(T) = \mathsf{dom}(r)$ and an algorithm A such that, for each string $x \in \mathsf{dom}(r)$, the algorithm A computes $r(x)$ within time $T(x)$; when this holds, we also say that r is *FPT-computable* with respect to κ *via* A.

As is standard, we may and do freely interchange among elements of Σ^*, $\Sigma^* \times \Sigma^*$, and $\mathbb{N}$. We define FPT to be the class that contains a parameterized problem (Q, κ) if and only if Q is FPT-computable with respect to κ.

We now introduce a notion of reduction for counting problems, which is a form of Turing reduction. We use $\wp_{\mathsf{fin}}(A)$ to denote the set containing all finite subsets of A.

▶ **Definition 6.** A *counting FPT-reduction* from a parameterized problem (Q, κ) to a second parameterized problem (Q', κ') consists of a computable function $h : \Sigma^* \to \wp_{\mathsf{fin}}(\Sigma^*)$, and an algorithm A such that:

- on an input x, A may make oracle queries of the form $Q'(y)$ with $\kappa'(y) \in h(\kappa(x))$, and
- Q is FPT-computable with respect to κ via A.

We use Clique to denote the decision problem where (k, G) is a yes-instance when G is a graph that contains a clique of size $k \in \mathbb{N}$. By #Clique we denote the problem of counting, given (k, G), the number of k-cliques in the graph G. The parameterized versions of these problems, denoted by p-Clique and p-#Clique, are defined via the parameterization π_1. We will make tacit use of the following well-known facts: FPT is closed under counting FPT-reduction; p-Clique is complete for $\mathsf{W}[1]$ under counting FPT-reduction; and, p-#Clique complete for $\#\mathsf{W}[1]$ under counting FPT-reduction.

A *promise problem* is a pair $\langle Q, I\rangle$ where Q is a problem and $I \subseteq \Sigma^*$. When C is a complexity class, define prom-C to contain a promise problem $\langle Q, I\rangle$ when there exists $P \in C$ such that, for all $x \in I$, it holds that $P(x) = Q(x)$. A *parameterized promise problem* is a pair $(\langle Q, I\rangle, \kappa)$ consisting of a promise problem $\langle Q, I\rangle$ and a parameterization κ; such a problem will also be notated by $\langle (Q, \kappa), I\rangle$. When C is a parameterized complexity class, define prom-C to contain a promise problem $(\langle Q, I\rangle, \kappa)$ when there exists a problem P such that $(P, \kappa) \in C$ and for all $x \in I$, it holds that $P(x) = Q(x)$.

3 Conjunctive Queries and Computational Problems

A *conjunctive query* is a relational first-order formula (possibly with free variables) of the form $\exists v_1 \dots \exists v_n \bigwedge_{i=1}^{m} \alpha_i$ where each α_i is a predicate application, that is, an atomic formula of the form $R(\vec{u})$ where R is a relation symbol and $\vec{u}$ is a tuple of variables. Since the only type of queries that we are concerned with in this article are conjunctive queries, we will sometimes simply use *query* to refer to a conjunctive query.

▶ **Definition 7.** To a conjunctive query ϕ over the vocabulary τ, we assign a structure $\mathbf{A} = \mathbf{A}_\phi$, called the *natural model*, as follows: the domain of $\mathbf{A}$ is $\mathsf{var}(\phi)$; $\mathbf{A}$ is over the vocabulary τ; and for each relation symbol $R \in \tau$, we set $R^{\mathbf{A}} := \{\vec{a} \mid R(\vec{a}) \text{ is an atom of } \phi\}$.

To each conjunctive query ϕ we assign the pair $(\mathbf{A}, S)$ where $\mathbf{A}$ is the natural model of ϕ and S the set of its free variables. From such a pair $(\mathbf{A}, S)$, it is easy to reconstruct the corresponding query ϕ: each tuple of a relation of $\mathbf{A}$ is made into an atom, and then, one existentially quantifies the elements of A not in S to obtain ϕ. Because of this easy correspondence between queries and pairs $(\mathbf{A}, S)$, in a slight abuse of notation, we do not differentiate between pairs $(\mathbf{A}, S)$ and queries throughout. In particular, we will call a pair $(\mathbf{A}, S)$ a query, and we will use $\mathcal{C}$ interchangably for classes of queries and of pairs $(\mathbf{A}, S)$.

Let ϕ be a conjunctive query with assigned pair $(\mathbf{A}, S)$ and let $\mathbf{B}$ be a structure. Then a function $h : S \to B$ is a satisfying assignment of ϕ if and only if it can be extended to a homomorphism from $\mathbf{A}$ to $\mathbf{B}$; we denote the set of such functions by $\hom(\mathbf{A}, \mathbf{B}, S)$. In this article, we are interested in the following counting problem.

#CQ
Input: A query $(\mathbf{A}, S)$ and a structure $\mathbf{B}$.
Problem: Compute $\lvert\hom(\mathbf{A}, \mathbf{B}, S)\rvert$.

In the case of a conjunctive query $(\mathbf{A}, \emptyset)$ without free variables, the problem #CQ amounts to deciding whether or not there exists a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. We define this case as the problem CQ.

CQ
Input: A query $(\mathbf{A}, \emptyset)$ and a structure $\mathbf{B}$.
Problem: Decide if there exists a homomorphism from $\mathbf{A}$ to $\mathbf{B}$.

We define p-#CQ to be the parameterized problem $(\#\text{CQ}, \pi_1)$, that is, we take the parameter of each instance $((\mathbf{A}, S), \mathbf{B})$ to be $(\mathbf{A}, S)$. (Formally, we view each instance of p-#CQ as a pair of strings, where the first component encodes the query, and the second component encodes the structure.) In analogy to #CQ, we define p-CQ to be the parameterized problem (CQ, π_1).

We define the hypergraph of a query $(\mathbf{A}, S)$ to be the hypergraph $\mathcal{H} = (V, E)$ where V is the domain of $\mathbf{A}$ and $E := \{\mathsf{dom}(t) \mid t \in R^{\mathbf{A}},\ R^{\mathbf{A}} \text{ is a relation of } \mathbf{A}\}$ where $\mathsf{dom}(t)$

denotes the set of elements appearing in t. The treewidth of $(\mathbf{A}, S)$ is defined to be that of its hypergraph. Dalmau, Kolaitis and Vardi [7] proved that CQ can be solved efficiently, when the treewidth of the cores of the queries is bounded.

▶ **Theorem 8** ([7]). *Let $k \in \mathbb{N}$ be a fixed constant. Let $\mathcal{C}_k$ be the class of all structures with cores of treewidth at most k. Then the promise problem $\langle \mathrm{CQ}, \mathcal{C}_k \times \Sigma^* \rangle$ is in* prom-FP.

Grohe [12] showed that this result is optimal.

▶ **Theorem 9** ([12]). *Let $\mathcal{C}$ be a recursively enumerable class of structures of bounded arity. Assume* FPT $\neq$ W[1]. *Then the following statements are equivalent:*
1. $\langle \mathrm{CQ}, \mathcal{C} \times \Sigma^* \rangle \in$ prom-FP.
2. $\langle p\text{-}\mathrm{CQ}, \mathcal{C} \times \Sigma^* \rangle \in$ prom-FPT.
3. *There is a constant c such that the cores of the structures in $\mathcal{C}$ have treewidth at most c.*

Dalmau and Jonsson [6] considered the analogous question for #CQ for quantifier free queries and found that cores do not help in this setting.

▶ **Theorem 10** ([6]). *Let $\mathcal{Q}$ be the class of all quantifier free conjunctive queries, i.e., queries of the form $(\mathbf{A}, A)$. Let $\mathcal{C}$ be a recursively enumerable class of structures of bounded arity in $\mathcal{Q}$. Assume* FPT $\neq$ #W[1]. *Then the following statements are equivalent:*
1. $\langle \#\mathrm{CQ}, \mathcal{C} \times \Sigma^* \rangle \in$ prom-FP.
2. $\langle p\text{-}\#\mathrm{CQ}, \mathcal{C} \times \Sigma^* \rangle \in$ prom-FPT.
3. *There is a constant c such that the structures in $\mathcal{C}$ have treewidth at most c.*

It is common to consider classes of queries defined by restricting their associated hypergraph. For #CQ it turns out to be helpful to also encode which vertices of a hypergraph correspond to free variables, which is formalized in the following definition. A pair $(\mathcal{H}, S)$ where $\mathcal{H}$ is a hypergraph and S is a subset of the vertices of $\mathcal{H}$ is called an S-hypergraph. The S-hypergraph of a query $(\mathbf{A}, S)$ is $(\mathcal{H}, S)$ where $\mathcal{H}$ is the hypergraph of $\mathbf{A}$.

In [8], the following version of #CQ is considered.

$\#\mathrm{CQ}_{hyp}$
Input: An S-hypergraph $(\mathcal{H}, S)$ and an instance $((\mathbf{A}, S), \mathbf{B})$ of #CQ where $\mathcal{H}$ is the hypergraph of $\mathbf{A}$.
Problem: Compute $\lvert \hom(\mathbf{A}, \mathbf{B}, S) \rvert$.

We define $p\text{-}\#\mathrm{CQ}_{hyp}$ to be the parameterized problem $(\#\mathrm{CQ}_{hyp}, \pi_1)$; here, an instance of $\#\mathrm{CQ}_{hyp}$ is viewed as a pair $((\mathcal{H}, S), ((\mathbf{A}, S), \mathbf{B}))$, on which the operator π_1 returns $(\mathcal{H}, S)$.

It turns out that for $\#\mathrm{CQ}_{hyp}$ a parameter called S-star size is of critical importance. Let $\mathcal{H} = (V, E)$ be a hypergraph and $S \subseteq V$. Let C be the vertex set of a connected component of $\mathcal{H}[V - S]$. Let E_C be the set of hyperedges $\{e \in E \mid e \cap C \neq \emptyset\}$ and $V_C := \bigcup_{e \in E_C} e$. Then $\mathcal{H}[V_C]$ is called an *S-component* of $\mathcal{H}$. The size of a biggest independent set in $\mathcal{H}[V_C \cap S]$ is called the S-star size of the S-component $\mathcal{H}[V_C]$. The S-star size of $(\mathcal{H}, S)$ is then defined to be the maximum S-star size taken over all S-components of $(\mathcal{H}, S)$. By the *quantified star size* of a query $(\mathbf{A}, S)$ we refer to the S-star size of the S-hypergraph associated to $(\mathbf{A}, S)$. For more explanations on these notions and examples see [15, Section 3.2].

▶ **Theorem 11** ([8]). *Let $\mathcal{G}$ be a recursively enumerable class of S-hypergraphs of bounded arity. Assume that* W[1] $\neq$ FPT. *Then the following statements are equivalent:*
1. $\langle \#\mathrm{CQ}_{hyp}, \mathcal{G} \times \Sigma^* \rangle \in$ prom-FP.
2. $\langle p\text{-}\#\mathrm{CQ}_{hyp}, \mathcal{G} \times \Sigma^* \rangle \in$ prom-FPT.

3. *There is a constant c such that for each S-hypergraph $(\mathcal{H}, S)$ in $\mathcal{G}$ the treewidth of $\mathcal{H}$ and the S-star size of $\mathcal{H}$ are at most c.*

We have seen that for the problem CQ, cores of structures are crucial, while in the classification due to Dalmau and Jonsson, they do not matter at all. Thus we introduce a notion of cores for conjunctive queries that interpolates between these two extreme cases. The idea behind the definition is that we require the homomorphisms between $(\mathbf{A}, S)$ and its core to be the identity on the free variables, while they may map the quantified variables in any way that leads to a homomorphism. This is formalized as follows.

▶ **Definition 12.** For a conjunctive query $(\mathbf{A}, S)$ where $\mathbf{A}$ is defined on vocabulary τ, we define the *augmented structure*, denoted by $\mathsf{aug}(\mathbf{A}, S)$, to be the structure over the vocabulary $\tau \cup \{R_a \mid a \in S\}$ where $R_a^{\mathsf{aug}(\mathbf{A},S)} := \{a\}$. We define the *core of a conjunctive query* of $(\mathbf{A}, S)$ to be the core of $\mathsf{aug}(\mathbf{A}, S)$.

▶ **Example 13.** Let $(\mathbf{A}, S)$ be a query without free variables, that is, where $S = \emptyset$; then the core of $(\mathbf{A}, S)$ is the core of $\mathbf{A}$. If $(\mathbf{A}, S)$ is quantifier-free, that is, where $S = A$, then the core of $(\mathbf{A}, S)$ equals $\mathbf{A}$.

The cores of conjunctive queries were essentially already studied by Chandra and Merlin in a seminal paper [2] although the notation used there is different. We give a fundamental result on conjunctive queries. We call two queries $(\mathbf{A}_1, S)$ and $(\mathbf{A}_2, S)$ *equivalent queries* if for each structure $\mathbf{B}$ we have $\hom(\mathbf{A}_1, \mathbf{B}, S) = \hom(\mathbf{A}_2, \mathbf{B}, S)$.

▶ **Theorem 14** ([2])**.** *If two conjunctive queries $(\mathbf{A}_1, S)$ and $(\mathbf{A}_2, S)$ have the same core (up to isomorphism), then they are equivalent.*

4 Case Complexity

In this section we develop a version of the case complexity framework advocated in [3] which is suitable for classifying counting problems. A main motivation for this framework is the growing amount of research on parameterized problems which are restricted by the permitted values of the parameter. In particular, this kind of problem arises naturally in query answering problems where one often restricts the admissible queries for the inputs (see e.g. [12, 6, 3]). An aim of the case complexity framework as introduced in [3] is to facilitate reductions between the considered restricted parameterized problems and to show results independent of computability assumptions for the parameter.

The central notion for our framework is the following: A *case problem* consists of a problem $Q : \Sigma^* \times \Sigma^* \to \mathbb{N}$ and a subset $S \subseteq \Sigma^*$, and is denoted $Q[S]$. When $Q[S]$ is a case problem, we define the following:

- param-$Q[S]$ is the parameterized problem (P, π_1) where $P(s, x)$ is defined as equal to $Q(s, x)$ if $s \in S$, and as 0 otherwise.
- prom-$Q[S]$ is the promise problem $\langle Q, S \times \Sigma^* \rangle$.
- param-prom-$Q[S]$ is the parameterized promise problem $(\mathsf{prom}\text{-}Q[S], \pi_1)$.

The case problem we consider in this paper will nearly exclusively be $\#\mathrm{CQ}[\mathcal{C}]$ where $\mathcal{C}$ is a class a class of conjunctive queries. Nevertheless, we stress the fact that our framework is fully generic and we believe that it will in the future also be useful for presenting and proving complexity classifications for other problems.

We now introduce a reduction notion for case problems.

▶ **Definition 15.** A *counting slice reduction* from a case problem $Q[S]$ to a second case problem $Q'[S']$ consists of

- a computably enumerable language $U \subseteq \Sigma^* \times \wp_{\mathsf{fin}}(\Sigma^*)$, and
- a partial function $r : \Sigma^* \times \wp_{\mathsf{fin}}(\Sigma^*) \times \Sigma^* \to \Sigma^*$ that has domain $U \times \Sigma^*$ and is FPT-computable with respect to (π_1, π_2) via an algorithm A that, on input (s, T, y), may make queries of the form $Q'(t, z)$ where $t \in T$,

such that the following conditions hold:

- (coverage) for each $s \in S$, there exists $T \subseteq S'$ such that $(s, T) \in U$, and
- (correctness) for each $(s, T) \in U$, it holds (for each $y \in \Sigma^*$) that

$$Q(s, y) = r(s, T, y).$$

As usual in counting complexity, it will often not be necessary to use the full generality of counting slice reductions. Therefore, we introduce a second, parsimonious notion of reductions for case problems which is often general enough but easier to deal with.

▶ **Definition 16.** A *parsimonious slice reduction* from a case problem $Q[S]$ to a second case problem $Q'[S']$ consists of

- a computably enumerable language $U \subseteq \Sigma^* \times \Sigma^*$, and
- a partial function $r : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^*$ that has domain $U \times \Sigma^*$ and is FPT-computable with respect to (π_1, π_2)

such that the following conditions hold:

- (coverage) for each $s \in S$, there exists $s' \in S'$ such that $(s, s') \in U$, and
- (correctness) for each $(t, t') \in U$, it holds (for each $y \in \Sigma^*$) that

$$Q(t, y) = Q'(t', r(t, t', y)).$$

We give some basic properties of counting slice reductions. Their proofs can be found in the full version of this paper.

▶ **Proposition 17.** *If $Q[S]$ parsimoniously slice reduces to $Q'[S']$, then $Q[S]$ counting slice reduces to $Q'[S']$.*

▶ **Theorem 18.** *Counting slice reducibility is transitive.*

The next two theorems give the connection between case complexity and parameterized complexity. In particular, they show that, from a counting slice reduction, one can obtain complexity results for the corresponding parameterized problems.

▶ **Theorem 19.** *Let $Q[S]$ and $Q'[S']$ be case problems. Suppose that $Q[S]$ counting slice reduces to $Q'[S']$, and that both S and S' are computable. Then* param-$Q[S]$ *counting FPT-reduces to* param-$Q'[S']$.

▶ **Theorem 20.** *Let $Q[S]$ be a case problem, and let $K : \Sigma^* \times \Sigma^* \to \mathbb{N}$ be a problem. Suppose that* param-prom-$Q[S]$ *is in* prom-FPT*, S is computably enumerable, and that the case problem $K[\Sigma^*]$ counting slice reduces to $Q[S]$. Then the parameterized problem (K, π_1) is in* FPT.

In the remainder of the paper, we will show all our reductions in the case complexity framework and then use Theorem 19 and Theorem 20 to derive parameterized complexity results. This approach lets us give results on $\#\mathrm{CQ}[\mathcal{C}]$ for different complexity assumptions on $\mathcal{C}$ without having to deal with these assumptions in the proofs. Thus we separate the technicalities of the reductions from the assumptions on $\mathcal{C}$ which in our opinion gives a far clearer presentation.

5 Statement of the main results

In this section we present the main results of this paper which we will then prove in the remainder of the paper. For the statement of the results we will use certain S-hypergraphs that we get as a contraction of the S-hypergraphs of conjunctive queries. When deleting a vertex v from a hypergraph, we delete v from the vertex set and all edges it appears in but keep all edges, unless they become empty after the deletion of v.

▶ **Definition 21.** To every S-hypergraph $(\mathcal{H}, S)$ we define an S-hypergraph $\mathsf{contract}(\mathcal{H}, S)$ as follows: We add an edge $\{u, v\}$ for any pair of vertices u, v that appears in a common S-component of $\mathcal{H}$. Then we delete the vertices in $V(\mathcal{H}) \setminus S$. To a class $\mathcal{G}$ of S-hypergraphs we define $\mathsf{contract}(\mathcal{G}) := \{\mathsf{contract}(\mathcal{H}, S) \mid (\mathcal{H}, S) \in \mathcal{G}\}$.

For a conjunctive query $(\mathbf{A}, S)$ let $\mathsf{contract}(\mathbf{A}, S)$ be $\mathsf{contract}(\mathcal{H}, S)$ where $(\mathcal{H}, S)$ is the S-hypergraph of the core of $(\mathbf{A}, S)$. For a class $\mathcal{C}$ of conjunctive queries, set $\mathsf{contract}(\mathcal{C}) := \{\mathsf{contract}(\mathbf{A}, S) \mid (\mathbf{A}, S) \in \mathcal{C}\}$.

We first present a version of our main result using the framework of case complexity.

▶ **Theorem 22.** *Let $\mathcal{C}$ be a class of conjunctive queries.*

1. *If the cores of $\mathcal{C}$ and $\mathsf{contract}(\mathcal{C})$ are of bounded treewidth, then prom-$\#\mathrm{CQ}[\mathcal{C}] \in \mathsf{prom}$-$\mathsf{FP}$, and hence param-prom-$\#\mathrm{CQ}[\mathcal{C}] \in \mathsf{prom}$-$\mathsf{FPT}$.*
2. *If the cores of $\mathcal{C}$ are of unbounded treewidth but $\mathsf{contract}(\mathcal{C})$ is of bounded treewidth, then $\#\mathrm{CQ}[\mathcal{C}]$ is equivalent to $\textsc{Clique}[\mathbb{N}]$ with respect to counting slice reductions.*
3. *If the treewidth of $\mathsf{contract}(\mathcal{C})$ is unbounded, then there is a counting slice reduction from $\#\textsc{Clique}[\mathbb{N}]$ to $\#\mathrm{CQ}[\mathcal{C}]$.*

We will prove Theorem 22 in Section 7.5. Using the results on case complexity, we derive from Theorem 22 two versions of the trichotomy phrased in terms of promise problems and of non-promise problems, depending on whether or not the class $\mathcal{C}$ of conjunctive queries is assumed to be recursively enumerable or computable, respectively.

▶ **Theorem 23.** *Let $\mathcal{C}$ be a class of conjunctive queries which is recursively enumerable. In the scope of this theorem, let us say that the class $\mathcal{C}$ is* tractable *if $\langle \#\mathrm{CQ}, \mathcal{C} \times \Sigma^* \rangle \in \mathsf{prom}$-$\mathsf{FP}$ and $\langle p\text{-}\#\mathrm{CQ}, \mathcal{C} \times \Sigma^* \rangle \in \mathsf{prom}$-$\mathsf{FPT}$.*

1. *If the cores of $\mathcal{C}$ and $\mathsf{contract}(\mathcal{C})$ have bounded treewidth, then $\mathcal{C}$ is tractable.*
2. *If the cores of $\mathcal{C}$ have unbounded treewidth, then $\mathcal{C}$ is not tractable, unless p-$\textsc{Clique}$ is in FPT (and hence $\mathsf{FPT} = \mathsf{W}[1]$).*
3. *If $\mathsf{contract}(\mathcal{C})$ has unbounded treewidth, then $\mathcal{C}$ is not tractable, unless p-$\#\textsc{Clique}$ is in FPT (and hence $\mathsf{FPT} = \#\mathsf{W}[1]$).*

Proof. (1) follows directly from item (1) of Theorem 22. (2) and (3) follow directly from the respective items of Theorem 22 and Theorem 20. ◀

From Theorem 23, one can immediately derive, as corollaries, Theorem 9 and Theorem 10.

Let us use $(p\text{-}\#\mathrm{CQ} \restriction I)$ to denote the parameterized problem which is equal to $\#\mathrm{CQ}$ on I, and is equal to 0 elsewhere (and has the parameterization of p-$\#\mathrm{CQ}$).

▶ **Theorem 24.** *Let $\mathcal{C}$ be a class of conjunctive queries which is computable.*

1. *If the cores of $\mathcal{C}$ and $\mathsf{contract}(\mathcal{C})$ have bounded treewidth, then $(p\text{-}\#\mathrm{CQ} \restriction \mathcal{C} \times \Sigma^*)$ is in FPT.*
2. *If the cores of $\mathcal{C}$ have unbounded treewidth, and $\mathsf{contract}(\mathcal{C})$ has bounded treewidth, then $(p\text{-}\#\mathrm{CQ} \restriction \mathcal{C} \times \Sigma^*)$ is equivalent to p-$\textsc{Clique}$ under counting FPT-reduction.*

3. *If* $\mathsf{contract}(\mathcal{C})$ *has unbounded treewidth,* $(p\text{-}\#\mathrm{CQ} \restriction \mathcal{C} \times \Sigma^*)$ *admits a counting FPT-reduction from* $p\text{-}\#\textsc{Clique}$.

Proof. For (1), the FPT algorithm is to first decide, given an instance $(\phi, \mathbf{B})$, whether or not $\phi \in \mathcal{C}$; if so, the algorithm invokes the algorithm of Theorem 22, otherwise, it returns 0. (2) and (3) follow immediately from Theorem 22 and Theorem 19. ◀

6 Positive complexity results

In this section, we will prove a counting version of Theorem 8. We will use a lemma that is probably well known, but as we could not find a reference, we give a proof for it in the full version of this paper.

▶ **Lemma 25.** *Let* $k \in \mathbb{N}$ *be a fixed constant. There exists a polynomial-time algorithm that, given a structure* $\mathbf{A}$ *whose core has treewidth at most* k*, outputs a core of* $\mathbf{A}$.

Lemma 25 yields a counting version of Theorem 8 as an easy corollary.

▶ **Corollary 26.** *Let* $\mathcal{C}$ *be a class of conjunctive queries such that the cores of the queries in* $\mathcal{C}$ *have bounded quantified star size and bounded treewidth. Then* $\mathsf{prom}\text{-}\#\mathrm{CQ}[\mathcal{C}] \in \mathsf{prom}\text{-}\mathsf{FP}$.

Proof. Let $((\mathbf{A}, S), \mathbf{B})$ be an instance of $\mathsf{prom}\text{-}\#\mathrm{CQ}[\mathcal{C}]$ with domain A. By the promise, there is a constant c such that the treewidth and the quantified star size of the core of $(\mathbf{A}, S)$ are at most c. We simply compute the core of $(\mathbf{A}, S)$ with Lemma 25 and delete from it the relations R_a introduced when constructing $\mathsf{aug}(\mathbf{A})$. Call the resulting query $(\hat{\mathbf{A}}, S)$. By construction, $(\mathbf{A}, S)$ and $(\hat{\mathbf{A}}, S)$ have the same core, so by Theorem 14 are equivalent. Moreover, the treewidth and quantified star size of $(\hat{\mathbf{A}}, S)$ are bounded by c and thus Theorem 11 lets us solve the instance in polynomial time. ◀

Let us discuss Corollary 26.

Theorem 8 and thus also Lemma 25 crucially depends on the fact that we know by an outside promise that the treewidth of the cores we consider is bounded. If this bound is not satisfied, then the algorithm of Theorem 8 may give false positive results. Consequently, the algorithm of Lemma 25 may compute a structure that is in fact *not* the core of the input and then the algorithm of Corollary 26 gives the wrong count. Unfortunately, deciding if the core of a conjunctive query has treewidth at most k is NP-complete [7] and even the problem of deciding if a fixed structure is the core of a given structure is NP-complete. Thus there is no efficient way of realizing that the core computed by the algorithm of Lemma 25 is wrong.

Consequently, while the result of Corollary 26 is very nice from a theoretical point of view (we will see in the next section that it is in fact optimal), it is probably of limited value from a more practical perspective. We see this as evidence that in fact parameterized complexity is a framework better suited for the type of problem discussed in this paper. Note that in this more relaxed setting of parameterized complexity, computing the core of a query by brute force can easily be done in the allowed time, because the core depends only on the query which is the parameter.

We now present a counting algorithm for $\#\mathrm{CQ}[\mathcal{C}]$ for certain classes $\mathcal{C}$ that has oracle access to CQ, the decision version of #CQ. Let ALL be the class of all conjunctive queries.

▶ **Lemma 27.** *Let* $\mathcal{C}$ *be a class of queries such that the treewidth of the* S*-hypergraphs in* $\mathsf{contract}(\mathcal{C})$ *is bounded by a constant* c*. Then there is a counting slice reduction from* $\#\mathrm{CQ}[\mathcal{C}]$ *to* $\mathrm{CQ}[\mathsf{ALL}]$.

The idea of the proof is as follows: Since the treewidth of $\mathsf{contract}(\mathcal{C})$ is bounded, we know that the unbounded treewidth of the cores does not originate from the structure of the free variables but only from the way the quantified variables interact in the S-components. We use the oracle for CQ[ALL] to solve subqueries of the original query in order to "contract" the quantified variables into one variable per S-component. This results in an instance with the same solutions that has bounded treewidth. We then solve this instance with the algorithm of Theorem 11. The complete proof of Lemma 27 can be found in the full version of this paper.

7 Hardness results

In this section we will prove the hardness results for Theorem 22. The main idea is reducing from the hard cases of Theorem 11 in several steps.

7.1 Simulating unary relations

In this section we show that for queries whose augmented structure is a core we can simulate unary relations on the variables of the query. These additional relations will later allow us to tell the variables apart such that we can later simulate the case in which all atoms of the queries have different relation symbols.

▶ **Lemma 28.** *Let $(\mathbf{A}, S)$ be a conjunctive query such that $\mathsf{aug}(\mathbf{A}, S)$ is a core. Then every homomorphism $h : \mathbf{A} \to \mathbf{A}$ with $h|_S = \mathrm{id}$ is a bijection.*

Proof. Clearly, h is also a homomorphism $h : \mathsf{aug}(\mathbf{A}, S) \to \mathsf{aug}(\mathbf{A}, S)$, because $h(a) = a \in R_a^{\mathsf{aug}(\mathbf{A},S)}$ for every $a \in S$. But by assumption $\mathsf{aug}(\mathbf{A}, S)$ is a core, so there is no homomorphism from $\mathsf{aug}(\mathbf{A}, S)$ to a proper substructure and thus h must be a bijection on $\mathsf{aug}(\mathbf{A}, S)$ and consequently also on $\mathbf{A}$. ◀

We assign a structure $\mathbf{A}^*$ to every structure $\mathbf{A}$:

▶ **Definition 29.** To a structure $\mathbf{A}$ we assign the structure $\mathbf{A}^*$ over the vocabulary $\tau \cup \{R_a \mid a \in A\}$ defined as $\mathbf{A}^* := \mathbf{A} \cup \bigcup_{a \in A} R_a^{\mathbf{A}^*}$ where $R_a^{\mathbf{A}^*} := \{a\}$.

Note that $\mathsf{aug}(\mathbf{A}, S)$ and $\mathbf{A}^*$ differ in which relations we add: For the structure $\mathsf{aug}(\mathbf{A}, S)$ we add $R_a^{\mathsf{aug}(\mathbf{A},S)}$ for variables $a \in S$ while for $\mathbf{A}^*$ we add $R_a^{\mathbf{A}^*}$ for all $a \in A$. Thus, $\mathbf{A}^*$ in general may have more relations than $\mathsf{aug}(\mathbf{A}, S)$.

We now formulate the main lemma of this section whose proof uses ideas from [6].

▶ **Lemma 30.** *Let $\mathcal{C}$ be a class of conjunctive queries such that for each $(\mathbf{A}, S) \in \mathcal{C}$ the augmented structure $\mathsf{aug}(\mathbf{A}, S)$ is a core. Let $\mathcal{C}^* := \{(\mathbf{A}^*, S) \mid (\mathbf{A}, S) \in \mathcal{C}\}$. Then there is a counting slice reduction from p-#CQ[$\mathcal{C}^*$] to p-#CQ[$\mathcal{C}$].*

Proof. Let $((\mathbf{A}^*, S), \mathbf{B})$ be an input for #CQ[$\mathcal{C}^*$]. Remember that $\mathbf{A}^*$ and $\mathbf{B}$ are structures over the vocabulary $\tau \cup \{R_a \mid a \in A\}$. For every query $(\mathbf{A}, S)$, the relation U of our counting slice reduction contains $((\mathbf{A}^*, S), (\mathbf{A}, S))$. Obviously, U is computable and satisfies the coverage property.

We now will reduce the computation of the size $|\hom(\mathbf{A}^*, \mathbf{B}, S)|$ to the computation of $|\hom(\mathbf{A}, \mathbf{B}', S)|$ for different structures $\mathbf{B}'$.

Let $D := \{(a,b) \in A \times B \mid b \in R_a^{\mathbf{B}}\}$ and define a structure $\mathbf{D}$ over the vocabulary τ with the domain D that contains for each relation symbol $R \in \tau$ the relation

$$R^{\mathbf{D}} := \{((a_1,b_1),\dots,(a_r,b_r)) \mid (a_1,\dots,a_r) \in R^{\mathbf{A}}, (b_1,\dots,b_r) \in R^{\mathbf{B}}, \\ \forall i \in [r] : (a_i,b_i) \in D\}.$$

Let again $\pi_1 : D \to A$ be the projection onto the first coordinate, i.e., $\pi_1(a,b) := a$. Observe that π_1 is by construction of $\mathbf{D}$ a homomorphism from $\mathbf{D}$ to $\mathbf{A}$.

We will several times use the following claim:

▶ **Claim 1.** *Let h be a homomorphism from $\mathbf{A}$ to $\mathbf{D}$ with $h(S) = S$. Then $\pi_1 \circ h$ is an automorphism of $\mathbf{A}$.*

Proof. Let $g := \pi_1 \circ h$. As the composition of two homomorphisms, g is a homomorphism from $\mathbf{A}$ to $\mathbf{A}$. Furthermore, by assumption $g|_S$ is a bijection from S to S. Since S is finite, there is $i \in \mathbb{N}$ such that $g^i|_S = \mathrm{id}$. But g^i is a homomorphism and thus, by Lemma 28, g^i is a bijection. It follows that g is a bijection.

Since A is finite, there is $j \in \mathbb{N}$ such that $g^{-1} = g^l$. It follows that g^{-1} is a homomorphism and thus g is an automporphism. ◀

Let $\mathcal{N}$ be the set of mappings $h : S \to D$ with $\pi_1 \circ h = \mathrm{id}$ that can be extended to a homomorphism $h' : \mathbf{A} \to \mathbf{D}$.

▶ **Claim 2.** *There is a bijection between $\hom(\mathbf{A}^*, \mathbf{B}, S)$ and $\mathcal{N}$.*

Proof. For each $h^* \in \hom(\mathbf{A}^*, \mathbf{B}, S)$ we define $P(h^*) := h$ by $h(a) := (a, h^*(a))$ for $a \in S$. From the extension of h^* to A we get an extension of h that is a homomorphism and thus $h \in \mathcal{N}$. Thus P is a mapping $P : \hom(\mathbf{A}^*, \mathbf{B}, S) \to \mathcal{N}$.

We claim that P is a bijection. Clearly, P is injective. We we will show that it is surjective as well. To this end, let $h : S \to D$ be a mapping in $\mathcal{N}$ and let h_e be a homomorphism from $\mathbf{A}$ to $\mathbf{D}$ that is an extension of h. By definition of $\mathcal{N}$ such a h_e must exist. By Claim 1 we have that $\pi_1 \circ h_e$ is an automorphism, and thus $(\pi_1 \circ h_e)^{-1}$ is a homomorphism. We set $h'_e := h_e \circ (\pi_1 \circ h_e)^{-1}$. Obviously, h'_e is a homomorphism from $\mathbf{A}$ to $\mathbf{D}$, because h'_e is the composition of two homomorphisms. Furthermore, for all $a \in S$ we have $h'_e(a) = (h_e \circ (\pi_1 \circ h_e))(a) = (h_e \circ (\pi_1 \circ h))(a) = h_e(a) = h(a)$, so h'_e is an extension of h. Moreover $\pi_1 \circ h'_e = (\pi_1 \circ h_e) \circ (\pi_1 \circ h_e)^{-1} = \mathrm{id}$. Hence, we have $h'_e = \mathrm{id} \times \hat{h}$ for a homomorphism $\hat{h} : \mathbf{A} \to \hat{\mathbf{B}}$, where $\hat{\mathbf{B}}$ is the structure we get from $\mathbf{B}$ by deleting the relations $R_a^{\mathbf{B}}$ for $a \in A$. But by definition $h'_e(a) \in D$ for all $a \in A$ and thus $\hat{h}(a) \in R_a^{\mathbf{B}}$. It follows that $\hat{h}$ is a homomorphism from $\mathbf{A}^*$ to $\mathbf{B}$. We set $h^* := \hat{h}|_S$. Clearly, $h^* \in \hom(\mathbf{A}^*, \mathbf{B}, S)$ and $P(h^*) = h$. It follows that P is surjective. This proves the claim. ◀

Let I be the set of mappings $g : S \to S$ that can be extended to an automorphism of $\mathbf{A}$. Let $\mathcal{N}'$ be the set of mappings $h : S \to D$ with $(\pi_1 \circ h)(S) = S$ that can be extended to homomorphisms $h' : \mathbf{A} \to \mathbf{D}$.

▶ **Claim 3.**

$$|\hom(\mathbf{A}^*, \mathbf{B}, S)| = \frac{|\mathcal{N}'|}{|I|}.$$

Proof. Because of Claim 2 it is sufficient to show that

$$|\mathcal{N}'| = |\mathcal{N}||I|. \tag{1}$$

We first prove that

$$\mathcal{N}' = \{f \circ g \mid f \in \mathcal{N}, g \in I\}. \tag{2}$$

The $\supseteq$ direction is obvious. For the other direction let $h \in \mathcal{N}'$. Let h' be the extension of h that is a homomorphism $h' : \mathbf{A} \to \mathbf{D}$. By Claim 1, we have that $g := \pi_1 \circ h'$ is an automorphism of $\mathbf{A}$. It follows that $g^{-1}|_S \in I$. Furthermore, $h \circ g^{-1}|_S$ is a mapping from S to D and $h' \circ g^{-1}$ is an extension that is a homomorphism from $\mathbf{A}$ to $\mathbf{D}$. Furthermore $(\pi_1 \circ h' \circ g^{-1}|_S)(a) = (g|_S \circ g^{-1}|_S)(a) = a$ for every $a \in S$ and hence $h' \circ g^{-1}|_S \in \mathcal{N}$ and $h = h \circ g^{-1}|_S \circ g|_S$ which proves the claim (2).

To show (1), we claim that for every $f, f' \in \mathcal{N}$ and every $g, g' \in I$, if $f \neq f'$ or $g \neq g'$, then $f \circ g \neq f' \circ g'$. To see this, observe that f can always be written as $f = \mathrm{id} \times f_2$ and thus $(f \circ g)(a) = (g(a), f_2(g(a))$. Thus, if g and g' differ, $\pi_1 \circ f \circ g \neq \pi_1 \circ f' \circ g'$ and thus $f \circ g \neq f' \circ g'$. Also, if $g = g'$ and $f \neq f'$, then clearly $f \circ g \neq f' \circ g'$. This completes the proof of (1) and the claim. ◀

Clearly, the set I depends only on $(\mathbf{A}, S)$ and thus it can be computed by an FPT-algorithm. Thus it suffices to show how to compute $|\mathcal{N}'|$ in the remainder of the proof.

For each set $T \subseteq S$ we define $\mathcal{N}_T := \{h \in \hom(\mathbf{A}, \mathbf{D}, S) \mid (\pi_1 \circ h)(S) \subseteq T\}$. We have by inclusion-exclusion

$$|\mathcal{N}'| = \sum_{T \subseteq S} (-1)^{|S \setminus T|} |\mathcal{N}_T|. \tag{3}$$

Observe that there are only $2^{|S|}$ summands in (3) and thus if we can reduce all of them to #CQ with the query $(\mathbf{A}, S)$ this will give us the desired counting slice reduction.

We will now show how to compute the $|\mathcal{N}_T|$ by interpolation. So fix a $T \subseteq S$. Let $\mathcal{N}_{T,i}$ for $i = 0, \ldots, |S|$ consist of the mappings $h \in \hom(\mathbf{A}, \mathbf{D}, S)$ such that there are exactly i elements $a \in S$ that are mapped to $h(a) = (a', b)$ such that $a' \in T$. Obviously, $\mathcal{N}_T = \mathcal{N}_{T,|S|}$ with this notation.

Now for each $j = 1, \ldots, |S|$ we construct a new structure $\mathbf{D}_{j,T}$ over the domain $D_{j,T}$. To this end, for each $a \in T$, let $a^{(1)}, \ldots, a^{(j)}$ be copies of a which are not in D. Then we set

$$D_{j,T} := \{(a^{(k)}, b) \mid (a, b) \in D, a \in T, k \in [j]\} \cup \{(a, b) \mid (a, b) \in D, a \notin T\}.$$

We define a mapping $B : D \to \wp(D_{j,T})$, where $\wp(D_{j,T})$ is the power set of $D_{j,T}$, by

$$B(a, b) := \begin{cases} \{(a^{(k)}, b) \mid k \in [j]\}\}, & \text{if } a \in T \\ \{(a, b)\}, & \text{otherwise.} \end{cases}$$

For every relation symbol $R \in \tau$ we define $R^{\mathbf{D}_{T,j}} := \bigcup_{(d_1, \ldots, d_s) \in R^D} B(d_1) \times \ldots \times B(d_s)$.

Then every $h \in \mathcal{N}_{T,i}$ corresponds to i^j mappings in $\hom(\mathbf{A}, \mathbf{D}_{j,T}, S)$. Thus for each j we get $\sum_{i=1}^{|S|} i^j |\mathcal{N}_{T,i}| = |\hom(\mathbf{A}, \mathbf{D}_{j,T}, S)|$. This is a linear system of equations and the corresponding matrix is a Vandermonde matrix, so $\mathcal{N}_T = \mathcal{N}_{T,|S|}$ can be computed with an oracle for #CQ on the instances $((\mathbf{A}, S), \mathbf{D}_{j,T})$. The size of the linear system depends only on $|S|$. Furthermore, $\|D^j\| \leq \|D\| j^s \leq \|D\|^{s+1}$ where s is the bound on the arity of the relations symbols in τ and thus a constant. It follows that the algorithm described above is a counting slice reduction. This completes the proof of Lemma 30. ◀

7.2 Reducing from hypergraphs to structures

In this section we show that we can in certain situations reduce from $\#\mathrm{CQ}_{hyp}$ to $\#\mathrm{CQ}$. This will later allow us to reduce from the hard cases in Theorem 11 to show the hardness results of Theorem 22.

We proceed in several steps. Let in this section $\mathcal{C}$ be a class of conjunctive queries of bounded arity. To every query $(\mathbf{A}, S)$ we construct a structure $\hat{\mathbf{A}}$ as follows; note that when we use this notation, S will be clear from the context. Construct the augmented structure $\mathsf{aug}(\mathbf{A}, S)$ of $\mathbf{A}$ and compute its core. We define $\hat{\mathbf{A}}$ to be the structure that we get by deleting the relations R_a for $a \in S$ that we added in the construction of $\mathsf{aug}(\mathbf{A}, S)$. We set $\hat{\mathcal{C}} := \{(\hat{\mathbf{A}}, S) \mid (\mathbf{A}, S) \in \mathcal{C}\}$.

Note that in any situation where we apply both the $\hat{}$- and *-operators, the $\hat{}$ is applied before the *.

▶ **Claim 4.** *There is a parsimonious slice-reduction from $\#\mathrm{CQ}[\hat{\mathcal{C}}]$ to $\#\mathrm{CQ}[\mathcal{C}]$.*

Proof. The relation U relates to every query $(\mathbf{A}, S)$ the query $(\hat{\mathbf{A}}, S)$. Certainly, U is computable and by definition assigns to each query in $\hat{\mathcal{C}}$ a query in $\mathcal{C}$.

We have that $\hat{\mathbf{A}}$ is a substructure of $\mathbf{A}$ and there is a homomorphism from $\mathbf{A}$ to $\hat{\mathbf{A}}$, because there is a homomorphism from $\mathsf{aug}(\mathbf{A}, S)$ to $\mathsf{aug}(\hat{\mathbf{A}}, S)$. Hence, $\mathbf{A}$ and $\hat{\mathbf{A}}$ are homomorphically equivalent and by Theorem 14 we have that $(\mathbf{A}, S)$ and $(\hat{\mathbf{A}}, S)$ are equivalent. Thus setting $r((\mathbf{A}, S), (\hat{\mathbf{A}}, S), \mathbf{B})) := \mathbf{B}$ yields the desired parsimonious slice-reduction. ◀

Let $\hat{\mathcal{C}}^* := \{(\hat{\mathbf{A}}^*, S) \mid (\hat{\mathbf{A}}, S) \in \hat{\mathcal{C}}\}$. Note that, by Lemma 30, there is a counting slice reduction from $p\text{-}\#\mathrm{CQ}[\hat{\mathcal{C}}^*]$ to $p\text{-}\#\mathrm{CQ}[\hat{\mathcal{C}}]$.

Let now $\mathcal{G}$ be the class of S-hypergraphs associated to the queries in $\hat{\mathcal{C}}$.

▶ **Claim 5.** *There is a parsimonious slice reduction from $\#\mathrm{CQ}_{hyp}[\mathcal{G}]$ to $\#\mathrm{CQ}[\hat{\mathcal{C}}^*]$.*

Proof. The relation U relates every S-hypergraph $(\mathcal{H}, S)$ in $\mathcal{G}$ to all queries $(\hat{\mathbf{A}}^*, S)$ with the hypergraph $(\mathcal{H}, S)$. Certainly, U is computable and by definition of $\mathcal{G}$ it assigns to S-hypergraph in $\mathcal{G}$ a query in $\hat{\mathcal{C}}^*$.

It remains to describe the function r. So let $((\mathbf{A}, S), \mathbf{B})$ be a p-#CQ-instance such that $(\mathbf{A}, S)$ has the S-hypergraph $(\mathcal{H}, S) \in \mathcal{G}$. We assume w.l.o.g. that every tuple appears only in one relation of $\mathbf{A}$. If this is not the case, say a tuple t appears in two relations $R_1^{\mathbf{A}}$ and $R_2^{\mathbf{A}}$, then we build a new instance as follows: Delete t from $R_1^{\mathbf{A}}$ and $R_2^{\mathbf{A}}$, add a new relation $R_t^{\mathbf{A}}$ to $\mathbf{A}$ containing only t. Finally, set $R^B = R_1^B \cap R_2^{\mathbf{B}}$. This operation does not change the associated S-hypergraph, so this new instance still has the S-hypergraph $(\mathcal{H}, S)$. Moreover it is easy to see that it has the same set of solutions.

Let the vocabulary of $(\hat{\mathbf{A}}, S)$ be τ. We construct a structure $r((\mathbf{A}, S), (\hat{\mathbf{A}}^*, s), \mathbf{B}) =: \hat{\mathbf{B}}$ over the same relation symbols as $\hat{\mathbf{A}}^*$, i.e., over the vocabulary $\tau \cup \{R_a \mid a \in \hat{A}\}$. The structure $\hat{\mathbf{B}}$ has the domain $\hat{B} := A \times B$ where A is the domain of $\mathbf{A}$ and B is the domain of $\mathbf{B}$. For $\hat{R} \in \tau$ we set

$$\hat{R}^{\hat{\mathbf{B}}} := \{((a_1, b_1), \ldots, (a_k, b_k)) \mid (a_1, \ldots, a_k) \in \hat{R}^{\hat{\mathbf{A}}}, (a_1, \ldots, a_k) \in R^{\mathbf{A}}, (b_1, \ldots, b_k) \in R^{\mathbf{B}}\}.$$

Furthermore, for the relations symbols $\hat{R}_a$ that are added in the construction of $\hat{\mathbf{A}}^*$ from $\hat{\mathbf{A}}$ we set $\hat{R}_a^{\hat{\mathbf{B}}} := \{(a, b) \mid b \in B\}$, where B is the domain of $\mathbf{B}$.

It is easy to see that from a satisfying assignment $h : \mathbf{A} \to \mathbf{B}$ we get a homomorphism $h' : \hat{\mathbf{A}}^* \to \hat{\mathbf{B}}$ by setting $h'(a) := (a, h(a))$. Furthermore, this construction is obviously

bijective. Thus we get $|\hom(\mathbf{A}, \mathbf{B}, S)| = |\hom(\hat{\mathbf{A}}^*, \hat{\mathbf{B}}, S)|$. Since $\hat{\mathbf{B}}$ can be constructed in polynomial time in $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$, this is a parsimonious slice reduction. ◀

▶ **Corollary 31.** *Let $\mathcal{C}$ be a class of conjunctive queries of bounded arity and let $\mathcal{G}$ be the class of S-hypergraphs of the cores of $\mathcal{C}$. Then there is a counting slice reduction from $\#\mathrm{CQ}_{hyp}[\mathcal{G}]$ to $\#\mathrm{CQ}[\mathcal{C}]$.*

7.3 Strict star size

In this section we introduce a notion strict S-star size to simplify some of the arguments in the next section. We define the strict S-star size of a hypergraph to be the maximum number of vertices in S that are contained in one S-component of $\mathcal{H}$.

▶ **Lemma 32.** *Let $\mathcal{G}$ be a class of S-hypergraphs of bounded arity. If the strict S-star size of the S-hypergraphs in $\mathcal{G}$ is unbounded, then there is a counting slice reduction from* Clique[$\mathbb{N}$] *to $\#\mathrm{CQ}_{hyp}[\mathcal{G}]$.*

7.4 The main hardness results

In this section we use the results of the last sections to prove the hardness results of Theorem 22.

The proof of Theorem 11 in [8] directly yields the following result.

▶ **Lemma 33.** *Let $\mathcal{G}$ be a class of S-hypergraphs of bounded arity. If the treewidth of $\mathcal{G}$ is unbounded, then there is a counting slice reduction from to* Clique[$\mathbb{N}$] *to $\#\mathrm{CQ}_{hyp}[\mathcal{G}]$.*

Combining Lemma 33 and Corollary 31 yields the following corollary.

▶ **Corollary 34.** *Let $\mathcal{C}$ be a class of queries such that the treewidth of the cores of the queries in $\mathcal{C}$ is unbounded. Then there is a counting slice reduction from* Clique[$\mathbb{N}$] *to $\#\mathrm{CQ}[\mathcal{C}]$.*

▶ **Lemma 35.** *Let $\mathcal{G}$ be a class of hypergraphs such that $\mathsf{contract}(\mathcal{G})$ is of unbounded treewidth. Then there is a counting slice reduction from $\#$Clique[$\mathbb{N}$] to $\#\mathrm{CQ}_{hyp}[\mathcal{G}]$.*

Proof. Assume first that $\mathcal{G}$ is of unbounded strict S-star size. Then $\#\mathrm{CQ}_{hyp}[\mathcal{G}]$ is $\#\mathsf{W}[1]$-hard by Lemma 32. So we assume in the remainder of the proof that there is a constant c such that for every $(\mathcal{H}, S)$ in $\mathcal{G}$ every S-component of $(\mathcal{H}, S)$ contains only c vertices from S.

From Theorem 10 it follows that there is a counting slice reduction from $\#$Clique[$\mathbb{N}$] to $\#\mathrm{CQ}_{hyp}[\mathsf{contract}(\mathcal{G})]$. Therefore, it suffices to show parsimonious slice reduction (U, r) from $\#\mathrm{CQ}_{hyp}[\mathsf{contract}(\mathcal{G})]$ to $\#\mathrm{CQ}_{hyp}[\mathcal{G}]$ to show the lemma.

The relation U is defined as $U := \{(\mathsf{contract}(\mathcal{H}, S), (\mathcal{H}, S)) \mid (\mathcal{H}, S) \in \mathcal{G}\}$. By definition, this satisfies the covering condition.

For the definition of r, consider an instance $((\mathbf{A}, S), \mathbf{B})$ of $\#\mathrm{CQ}_{hyp}[\mathsf{contract}(\mathcal{G})]$ and let $\mathcal{H}$ be the hypergraph of $\mathbf{A}$. Moreover, let $(\mathcal{H}', S) \in \mathcal{G}$ be an S-hypergraph such that $(\mathcal{H}, S) = \mathsf{contract}(\mathcal{H}', S)$. W.l.o.g. assume that for every edge e of $\mathcal{H}$, the structure $\mathbf{A}$ contains one relation R_e containing only a single tuple $\vec{e}$ where $\vec{e}$ contains the elements of e in an arbitrary order. We construct an instance $r((\mathcal{H}, S), (\mathcal{H}', S), ((\mathbf{A}, S), \mathbf{B})) = ((\mathbf{A}', S), \mathbf{B}')$. Similarly to $\mathbf{A}$, the structure $\mathbf{A}'$ has for every edge e in $\mathcal{H}'$ a relation R_e that contains only a single tuple $\vec{e}$ with the properties as before. For every S-component C of $\mathcal{H}$ we do the following: Let D_C be the tuples encoding the homomorphisms h from $\mathbf{A}[V(C) \cap S]$ to $\mathbf{B}$. For every $v \in V(C) \setminus S$ we let D_C be the domain of v. Whenever two elements $u, v \in V(C) \setminus S$ appear in an edge, we set $R^{\mathbf{B}'}$ in such a way that for all tuples in $R_e^{\mathbf{B}}$ the assignments to u

and v coincide. Moreover, whenever $u \in V(C) \cup S$ and $v \in V(C) \setminus S$ we allow only tuples in which the assignment to u coincides with the assignment to u that is encoded in the assignment to v. For all edges e of $\mathcal{H}'$ with $e \setminus S \neq \emptyset$, we let $R_e^{\mathbf{B}'}$ contain all tuples that satisfy the two conditions above. Finally, for all edges e with $e \in S$ we set $R^{\mathbf{A}'} := R^{\mathbf{A}}$.

It is easy to verify that $\hom(\mathbf{A}, \mathbf{B}, S) = \hom(\mathbf{A}', \mathbf{B}', S)$. Thus it only remains to show that the construction can be done in polynomial time. Note first that the number of variables from S in any S-component of $\mathcal{H}'$ is bounded by c. Thus we can compute all domains D_C in time $\|\mathbf{B}\|^{O(c)}$. The rest of the construction can then be easily done in polynomial time. ◀

▶ **Corollary 36.** *Let $\mathcal{C}$ be a class of conjunctive queries such that* $\mathsf{contract}(\mathcal{C})$ *is of unbounded treewidth. Then there is a counting slice reduction from* #Clique[$\mathbb{N}$] *to* #CQ[$\mathcal{C}$].

Proof. This follows by combination of Lemma 35 and Corollary 31. ◀

7.5 Putting things together

We now finally show Theorem 22 by putting together the results of the last sections.

Proof of Theorem 22.
1. follows directly from Corollary 26 with the observation that bounded treewidth of $\mathsf{contract}(\mathcal{C})$ implies bounded S-star size.
2. is Corollary 34 and Lemma 27 using the fact that CQ[ALL] counting slice reduces to Clique[$\mathbb{N}$]; this follows from [9, Section 6.1].
3. is Corollary 36. ◀

8 Conclusion

In this paper we have proved a complete classification for the counting complexity of conjunctive queries, continuing a line of work that spans several previous papers [6, 8, 11]. While this solves the bounded arity case completely, the most apparent open question is what happens for the unbounded arity case. This case is rather well understood for the decision version CQ of the problem [14], but for counting not much is known. In particular, it is not known if the results of [14] can even be adapted to the quantifier free setting.

Another interesting problem would be to go from conjunctive queries to more expressive query languages. This has been done with some success for decision problems (see e.g. [3] and the references therein), but the situation for counting is much less clear. It is known that counting and decision differ a lot at least in some settings, e.g. even very simple unions of conjunctive queries yield hard counting problems [17, 15] while CQ for these queries is very easy. Can we get a better understanding of counting complexity in this setting and how it differs from decision?

To prove our results, we have extended the case complexity framework to counting complexity. We are very optimistic that this will be helpful when studying the research areas discussed above. Moreover, due to its generic nature, we feel that this framework should also be of use outside of the query answering context and allow transparent proofs and presentations in other areas of parameterized complexity.

Acknowledgements. Chen was supported by the Spanish project TIN2013-46181-C2-2-R, by the Basque project GIU12/26, and by the Basque grant UFI11/45.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2 A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC 1977*, pages 77–90. ACM, 1977.
3 Hubie Chen. The tractability frontier of graph-like first-order query sets. *CoRR*, abs/1407.3429v1, 2014. Conference version appeared in the proceedings of LICS '14.
4 Hubie Chen and Martin Grohe. Constraint satisfaction with succinctly specified relations. *J. Comput. Syst. Sci.*, 76(8):847–860, 2010.
5 Hubie Chen and Moritz Müller. One hierarchy spawns another: Graph deconstructions and the complexity classification of conjunctive queries. In *LICS*, 2014.
6 V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.
7 V. Dalmau, P.G. Kolaitis, and M.Y. Vardi. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In *International Conference on Principles and Practice of Constraint Programming 2002*, pages 310–326, 2002.
8 A. Durand and S. Mengel. Structural tractability of counting of solutions to conjunctive queries. *Theory of Computing Systems*, pages 1–48, 2014. accepted, to appear, final version available at http://dx.doi.org/10.1007/s00224-014-9543-y.
9 J. Flum and M. Grohe. Parameterized Complexity Theory. *Texts in Theoretical Computer Science. An EATCS Series*, 2006.
10 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.
11 Gianluigi Greco and Francesco Scarcello. Counting solutions to conjunctive queries: structural and hybrid tractability. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14*, pages 132–143, 2014.
12 M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), 2007.
13 P. Kolaitis and M. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.
14 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42, 2013.
15 S. Mengel. *Conjunctive Queries, Arithmetic Circuits and Counting Complexity*. PhD thesis, University of Paderborn, 2013.
16 C. Papadimitriou and M. Yannakakis. On the Complexity of Database Queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.
17 R. Pichler and S. Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 2013.
18 Nicole Schweikardt, Thomas Schwentick, and Luc Segoufin. Database theory: Query languages. In Mikhail J. Atallah and Marina Blanton, editors, *Algorithms and Theory of Computation Handbook*, volume 2: Special Topics and Techniques, chapter 19. CRC Press, second edition, Nov 2009.

Learning Tree Patterns from Example Graphs

Sara Cohen and Yaacov Y. Weiss

Rachel and Selim Benin School of Computer Science and Engineering
Hebrew University of Jerusalem
Jerusalem, Israel
{sara,yyweiss}@cs.huji.ac.il

Abstract

This paper investigates the problem of learning tree patterns that return nodes with a given set of labels, from example graphs provided by the user. Example graphs are annotated by the user as being either *positive* or *negative*. The goal is then to determine whether there exists a tree pattern returning tuples of nodes with the given labels in each of the positive examples, but in none of the negative examples, and, furthermore, to find one such pattern if it exists. These are called the *satisfiability* and *learning* problems, respectively.

This paper thoroughly investigates the satisfiability and learning problems in a variety of settings. In particular, we consider example sets that (1) may contain only positive examples, or both positive and negative examples, (2) may contain directed or undirected graphs, and (3) may have multiple occurrences of labels or be uniquely labeled (to some degree). In addition, we consider tree patterns of different types that can allow, or prohibit, wildcard labeled nodes and descendant edges. We also consider two different semantics for mapping tree patterns to graphs. The complexity of satisfiability is determined for the different combinations of settings. For cases in which satisfiability is polynomial, it is also shown that learning is polynomial. (This is non-trivial as satisfying patterns may be exponential in size.) Finally, the *minimal learning* problem, i.e., that of finding a minimal-sized satisfying pattern, is studied for cases in which satisfiability is polynomial.

1998 ACM Subject Classification H.2.3 Query languages, I.2.6 Learning

Keywords and phrases tree patterns, learning, examples

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.127

1 Introduction

Node-labeled graphs (or simply *graphs*, for short) are a popular data model and are useful in describing information about entities and their relationships. Traditional query languages for graphs are often based on *pattern matching*, i.e., the query is a pattern with a tuple of annotated nodes, called *output nodes*. Query results are simply tuples of nodes that are in the image of the output nodes with respect to some mapping from the query to the graph. The precise properties required of such a mapping depend on the setting (they may be required to be injective, they may allow edges to be mapped to paths, etc.).

Formulating a query over a set of graphs that returns specific tuples of nodes can be a difficult task. The graphs may be intricate, amalgamating many different relationships among nodes into a single structure. Finding a precise query that returns exactly the tuples of nodes of interest to the user may require extensive investigation into the precise structure of the graphs, and thus, may be infeasible for the user.

In this paper we study the problem of learning tree patterns from a given set of graphs with annotated tuples of interest. If tree patterns can be learned, then query formulation for

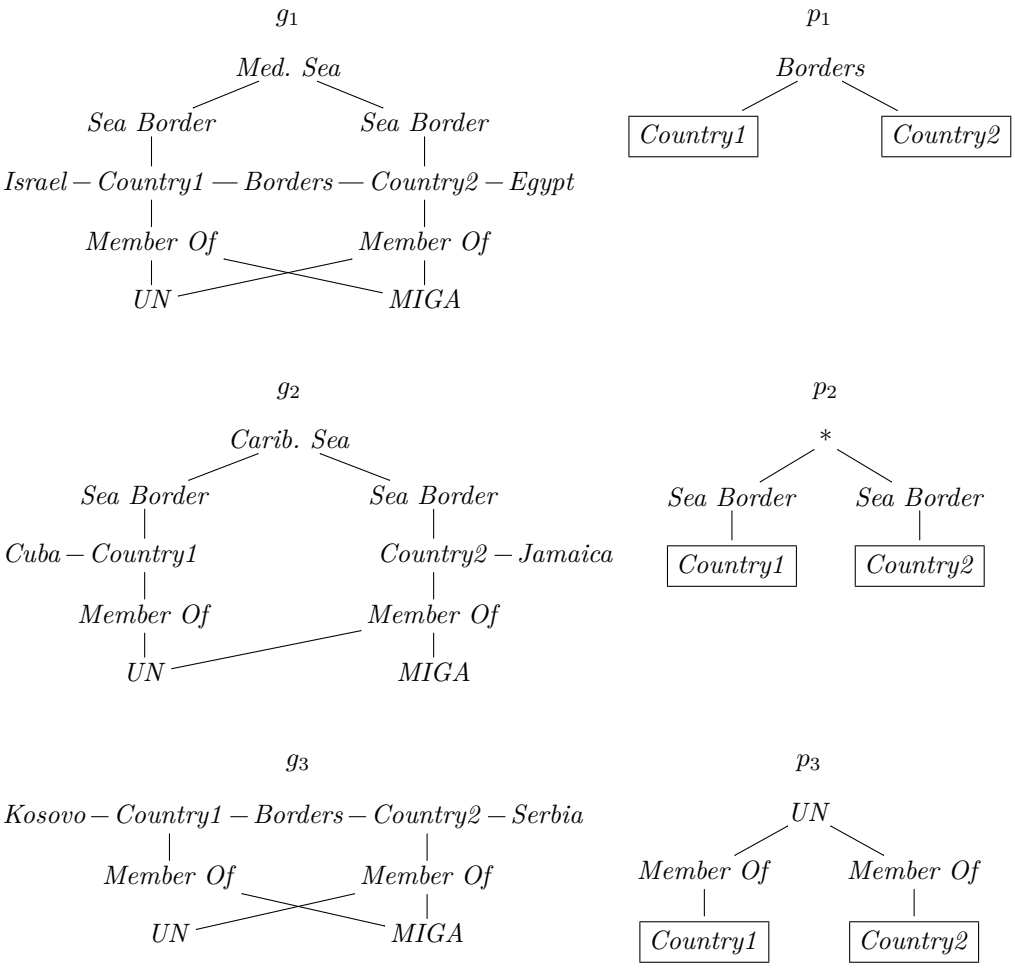


Figure 1 Motivating example with data from the CIA fact-book.

the user can be simplified, by having the user simply provide examples, while the system infers a satisfying pattern.

The ability to learn satisfying patterns is also of interest in other scenarios. For example, it allows certain and possible answers to be identified and returned [7]. Intuitively, using the terminology of [7], a certain (resp. possible) answer is one that is returned by all (resp. at least one) patterns satisfying the given examples. It has been shown in [7] that the ability to determine satisfiability of a set of examples immediately implies the ability to determine whether an answer is certain or possible. Thus, even without learning the user's precise query, if satisfiability can be determined, then the system can provide the user with answers that will (certainly or possibly) be of interest.

▶ **Example 1.** Consider graphs g_1, g_2, g_3 in Figure 1. These graphs depict a small portion of the information from the CIA fact-book, a labeled graph data source containing information about countries around the world, and their relationships. Thus, for example, g_1 describes the relationships between Israel and Egypt. These countries border each other, and both border the Mediterranean Sea. In addition, both Israel and Egypt are members of the UN and of MIGA. Similarly, g_2 contains information about the relationships of Cuba and

Jamaica. Given both g_1 and g_2 as positive examples, one possible pattern that can be inferred as holding between Country1 and Country2 is p_2, i.e., that they both border on some common sea. Another possible pattern that can be inferred as holding between Country1 and Country2 is that derived by replacing "UN" in p_3 (of Figure 1) by a wildcard. We call this pattern p_4.

The goals of this paper are to *(1)* determine whether there exists a satisfying pattern for a set of examples (e.g., "yes" for g_1, g_2), and to *(2)* develop methods to automatically find a (minimal) satisfying pattern (e.g., p_2, p_4). As explained above, a by-product of these results is the ability to determine whether a new (unlabeled) example is certainly, or possibly, of interest to the user (e.g., g_3 is possibly of interest, as the user may desire countries that are members of the same organization, and this holds for each of g_1, g_2, g_3).

The setting considered in this paper is as follows. We are given a set of graphs G, containing both *positive examples* g^+ and *negative examples* g^-. In addition, we are given a tuple of *distinguished labels* $l_1, \ldots, l_n$. Depending on the setting, these labels may uniquely identify a tuple of nodes in each graph of G, or may correspond to many different tuples of nodes. In the former case, we say that G is *output identifying*, and in the latter, G is *arbitrary*. Additional features of interest of the dataset $\Delta = (G, (l_1, \ldots, l_n))$ include *(1)* whether the graphs are *uniquely labeled*, in which case no labels appear more than once in a graph, *(2)* whether the graphs are directed or undirected, and *(3)* whether there are only positive examples, or both positive and negative examples.

We consider tree patterns that may contain (or prohibit) nodes with wildcards and/or descendant edges. In addition, a tree pattern contains a designated tuple of *output nodes*. Different semantics are defined for mapping tree patterns to graphs, depending on whether we require such mappings to be injective or not. Intuitively, a given tree pattern satisfies a positive (resp. negative) example, if there exists (resp. does not exist) a mapping from the tree pattern to the graph that maps the output nodes to nodes with the distinguished labels.

Three problems of interest are defined and studied. First, the *satisfiability problem* is to determine whether there exists a pattern satisfying a given dataset Δ. Second, the *learning problem* is to find a pattern satisfying Δ, if one exists. Note that even if satisfiability is in polynomial time, learning may still be difficult as satisfying patterns may be exponential in size. Third, the *minimal learning problem* is to find a pattern satisfying Δ, that is of minimal size. The complexity of these problems varies greatly depending on the given setting, i.e., characteristics of the dataset, the type of patterns allowed, and the type of mappings required from the patterns to the dataset. In some cases these problems are quite easy (i.e., in polynomial time), while in other cases even satisfiability is PSPACE-complete.

The main contributions of this paper are a thorough study of the three problems of interest, in different combinations of settings. Our results are quite extensive and are summarized in Table 1 for datasets with only positive examples, in Table 2 for datasets with both positive and negative examples, and in Table 3 for the minimal learning problem. Thus, our work clearly delineates in which cases learning is efficient, and thereby, provides the groundwork for a system which induces tree patterns from user examples.

2 Related Work

While the problem at hand has not been studied in the past, there is significant previous related work, falling into three general categories: learning queries from examples, graph mining and explaining query results. We discuss relevant work in each of these areas.

Learning Queries from Examples. There is a wealth of work on using machine learning techniques to learn XPath queries or tree patterns from positive and negative examples. Most often, these works are in the context of wrapper induction.

The goal of [5, 2, 4, 1, 17, 22, 21, 23] is to learn a tree-like query, usually using equivalence or membership questions. Intuitively, equivalence questions provide the user with a query (or examples of results of a query) and ask the user if the computer-provided query is equivalent to that of the user. Membership questions provide the user with an example, and ask the user whether this example should be marked positively. Many negative results have been shown, most notably, that tree automata cannot be learned in polynomial time with equivalence questions [2], as well as a similar negative result for learning even simple fragments of XPath [5]. Several of these works [17, 22] have been experimentally tested against wrapper induction benchmarks, and have been shown to work extremely well. Among these works, only [21] considers learning trees from graphs (but their setting differs significantly from that considered here, e.g., graphs contain edge variables that can be replaced by additional graphs, and the focus is on equivalence or subset queries). The above results are completely different from (and complementary to) those of ours. Most significantly, we do not ask the user questions of any type. The goal in this paper is quite simple: determine whether there exists any satisfying pattern, and find one if possible.

Somewhat slightly related to our work is the definability problem for graph query languages, studied in [3]. The goal in [3] is to determine whether a given relation over nodes in a graph can be defined by a query in a given language. The language considered differs significantly from ours, as they allow regular expressions and non-tree patterns. We note that they also have only positive examples (while all other examples are implicitly negative, and thus, there are no "non-labeled" examples), and they do not consider the problem of learning minimal sized satisfying patterns.

In [7] the problem of learning tree patterns from tree examples was studied. It was also shown that the ability to efficiently determine satisfiability is sufficient in order to efficiently determine whether a new unlabeled example can possibly be positive (i.e., there exists a satisfying pattern returning this answer), or is certainly positive (i.e., all satisfying patterns return this answer). While [7] showed this result in the context of tree data, it carries over to our framework.

Our paper differs very significantly from [7] – in its setting, scope and techniques. In [7], only tree data was considered (i.e., all examples are trees), trees are always directed and arbitrarily labeled, the width of the labels is exactly two (and always includes the root of each example) and the number of examples is constant. In the current paper, we consider a large number of different scenarios, all of which involve graph examples. Graph examples are a much richer formalism than tree examples. This is true even though our patterns are trees, as patterns must now choose between exponentially many alternative paths in a graph, and can traverse a cycle arbitrarily many times. The current paper presents comprehensive results on the many different possible settings (whereas [7] mostly focused on a single polynomial case). Thus, the current paper is significantly larger in scope than [7]. We note also that the techniques used in [7] are not applicable for the problems in the current paper as they were heavily based on the fact that the data in all examples are trees (and on the additional restrictions considered there), and do not carry over to a graph setting.

Graph Mining. Given a set of n graphs, the graph mining problem is to find subgraphs that are common to at least k of these graphs, where k is a user provided parameter. One commonly considered flavor of this problem is when the frequent subgraphs are trees. See [14]

for a survey on frequent subgraph mining and [15] for recent theoretical results on the complexity of frequent subgraph mining.

The results of this paper are closely related to the frequent subtree mining problem, for the special case where $k = n$, i.e., when the subtree must appear in all graphs. In particular, when there are only positive examples, and patterns cannot contain descendant edges, then learning tree patterns is quite similar in spirit to subtree mining. However, in our setting, the user also provides a tuple of distinguished labels that must be "covered" by the satisfying patterns. This significantly differs from subtree mining, in which no constraints require that specific subportions of the graph appear in any frequent subtree. It also makes determining whether there exists a satisfying pattern significantly harder. For example, it is always easy to find a single frequent subtree [15]; however, finding a satisfying pattern is often intractable.

Explaining Query Answers. Recently, there has been considerable interest in explaining query results to a user, i.e., in justifying why certain tuples do, or do not, appear in a query result. Most of this work, e.g., [6, 13, 19, 11], assumes that the query is given. Answers take the form of provenance of tuples, or explanations as to which query clause caused tuples to be eliminated. This is very different from our framework in which only the answers are given, and not the queries.

On the other hand [26, 25, 8] explain query answers, by synthesizing a query from the result. In [26], the focus is on generating a query that almost returns a given output (with a precise definition for "almost"). In [25] missing tuples in a query result are explained by generating a query that returns the query result, as well as the missing tuples. Finally, in [8] the problem of synthesizing view definitions from data is studied. These works consider relational databases and conjunctive queries, and the results are not immediately applicable to other settings. However, the results in this paper can immediately be applied to the problem of explaining query answers, for tree patterns over graph datasets. In particular, we show in this paper how to synthesize (i.e., learn) a tree pattern from a set of examples.

3 Definitions

In this section, we present basic terminology used throughout the paper and formally define the problems if interest.

Graphs and Examples. In this paper we consider labeled graphs that may be directed or undirected. We use Σ to denote an infinite set of labels. Given a graph g, we use V_g to denote the nodes of g, E_g to denote the edges of g and $l_g : V_g \to \Sigma$ to denote the *labeling function* that associates each node in g with a label from Σ.

We say that l is a *unique label* in g if there is precisely one node v for which $l_g(v) = l$. We say that a graph g is *uniquely labeled* if, for all $u \neq v \in V_g$ it holds that $l_g(u) \neq l_g(v)$.

A graph g, annotated with the superscript "+" or "−" is called an *example*. We say that g^+ is a *positive example* and g^- is a *negative example*. We use Δ to denote a pair $(G, \bar{l})$ where G is a set of examples, and $\bar{l} = (l_1, \ldots, l_n)$ is a tuple of distinct labels. We call Δ a *dataset* and $\bar{l}$ the *distinguished labels*.

We distinguish several special types of datasets. In particular, Δ is *positive* if G contains only positive examples, Δ is *uniquely labeled* if all graphs in G are uniquely labeled, and Δ is *output identifying* if, for all $i \leq n$, we have that l_i is a unique label in all graphs in G. Note that if Δ is uniquely labeled, then it is also output identifying, but the opposite does not hold in general. We require Δ to contain only directed or only undirected graphs (and not a

mix of the two). In the former case, we say that Δ is *directed* and in the latter, we say that Δ is *undirected*.

▶ Remark. Recall that the goal of this paper is to learn patterns connecting a series of labels in a graph. The special case of output identifying datasets is of particular interest, as it is equivalent to the following problem: We are given a set of graphs, each of which has a tuple of distinguished nodes. The goal is to find a pattern that returns the given tuples of nodes from the positive examples, and does not return the tuples of nodes from the negative examples. In other words, learning patterns from output-identifying datasets is the problem of learning patterns connecting a given series of nodes from each example graph.

▶ **Example 2.** Consider the graphs g_1, g_2, g_3 of Figure 1. The dataset Δ, defined as $(\{g_1^+, g_2^+, g_3^-\}, (\text{Country1}, \text{Country2}))$ contains undirected graphs, and both positive (g_1^+, g_2^+) and negative (g_3^-) examples. It is output identifying, as there is a single node in each graph with label *Country1* and a single node in each graph with label *Country2*. Note that the graphs are not uniquely labeled. The dataset $\Delta' = (\{g_1^+, g_2^+, g_3^-\}, (\text{Sea Border}, \text{Member Of}))$, which looks for patterns connecting the labels *Sea Border* and *Member Of* is not output identifying.

Tree Patterns. *Tree patterns* (or simply *patterns*, for short) are used to extract tuples of nodes from graphs. To be precise, a *tree pattern* is a pair $p = (t, \bar{o})$, where t is a labeled tree, and $\bar{o}$ is a tuple of nodes from t, called the *output nodes*, such that

- The labels of t are drawn from the set $\Sigma \cup \{*\}$ where $*$ is a special label, called the *wildcard*.
- The set of edges E_t of t is the union of disjoint sets $E_t^{/}$ and $E_t^{//}$, representing *child* and *descendant* edges, respectively.
- All leaf nodes in t are also output nodes (but not necessarily vice-versa).

As before, we use V_t and l_t to denote the nodes and labeling function of t, respectively. We will consider both undirected trees, and directed, rooted trees. When t is a directed rooted tree, we use $r_t \in V_t$ to denote the root of t.

▶ Remark. In this paper we focus on learning tree patterns from example graphs. Tree patterns in graphs have been of interest in the past, e.g., in the context of keyword proximity search, as they represent tight relationships between nodes [9, 16, 12]. While we do not study the problem of learning graph patterns directly, we note that this is somewhat less interesting than tree patterns. In particular, if there are only positive examples, then whenever there is a connected graph pattern satisfying the examples, there is also a connected tree pattern (derived by removing edges). When negative examples are allowed, then the setting in which there is a single positive and a single negative example already reduces to the graph homomorphism problem, and thus, will not be tractable [10]. Thus, the additional expressive power provided by graphs is either unnecessary (in the positive case) or quickly yields both NP and Co-NP hardness (when negative examples are allowed).

Embeddings. An *embedding* from a tree pattern $p = (t, \bar{o})$ to a graph g, is a function $\mu : V_t \to V_g$ such that

- μ maps (directed) edges in $E_t^{/}$ to (directed) edges in E_g, i.e., for all $(u, v) \in E_t^{/}$, $(\mu(u), \mu(v)) \in E_g$;
- μ maps (directed) edges in $E_t^{//}$ to (directed) paths in E_g, i.e., for all $(u, v) \in E_t^{//}$, there is a (directed) path from $\mu(u)$ to $\mu(v)$ in E_g of length at least 1;

- μ maps nodes v in V_t with a non-wildcard label to nodes in V_g with the same label, i.e., for all $v \in V_t$, either $l_t(v) = *$ or $l_t(v) = l_g(\mu(v))$.

Next we define when a tree pattern p satisfies a dataset Δ, using the notion of an embedding. We note that in the following definition, and throughout this paper, we assume that either *(1)* Δ contains undirected graphs and p is an undirected tree or *(2)* Δ contains directed graphs and p is a directed rooted tree. Moreover, we assume that the tuple of distinguished labels in Δ has the same cardinality as the output nodes tuple in p.

Given a tree pattern $p = (t, \bar{o})$ and a dataset $\Delta = (G, \bar{l})$, we will say that p *satisfies* Δ if

- for all $g^+ \in G$, there exists an embedding μ from p to g such that $l_g(\mu(\bar{o})) = \bar{l}$;
- for all $g^- \in G$, there does not exist an embedding μ from p to g for which $l_g(\mu(\bar{o})) = \bar{l}$.

Similarly, we say that p *injectively satisfies* Δ, if for all positive examples g^+, there exists an injective embedding that maps $\bar{o}$ to nodes with the labels $\bar{l}$ in g^+, and for all negative examples g^-, there does not exist an injective embedding that maps $\bar{o}$ to nodes with the labels $\bar{l}$ in g^-.

▶ **Example 3.** Figure 1 contains three tree patterns. The output nodes in these patterns are denoted with a rectangular box. Consider the datasets

$$\Delta_1 = (\{g_1^+, g_2^+, g_3^-\}, (\text{Country1}, \text{Country2}))$$
$$\Delta_2 = (\{g_1^+, g_2^-, g_3^+\}, (\text{Country1}, \text{Country2}))$$
$$\Delta_3 = (\{g_1^+, g_2^+\}, (\text{Country1}, \text{Country2}))\,.$$

The patterns p_2 and p_3 satisfy the datasets Δ_1 and Δ_3, but not Δ_2. Note that every pattern satisfying Δ_1 will satisfy Δ_3, but not vice-versa. Pattern p_1 satisfies Δ_2 (but not any of the others). We note that the pattern derived by replacing "UN" in p_3 with "$*$", satisfies Δ_3, as well as the dataset in which all three examples are positive.

Problems of Interest. Given a dataset, we will be interested in determining whether there exists a satisfying pattern (or an injectively satisfying pattern), as well as in finding such a pattern if one exists. Thus, our two problems of interest can be defined as follows.

▶ **Problem 1** ((Injective) Satisfiability)**.** Given a dataset Δ, determine whether there exists a pattern p that (injectively) satisfies Δ.

▶ **Problem 2** ((Injective) Learning)**.** Given a dataset Δ, find a pattern p that (injectively) satisfies Δ.

The complexity of these problems depends highly upon the types of patterns, embeddings and datasets allowed. Thus, we will study these problems for a variety of settings, considering: injective and arbitrary embeddings, patterns allowing/prohibiting wildcards and descendant edges, directed and undirected datasets, and uniquely labeled, output distinguishing and arbitrary datasets. We will also consider bounding various aspects of the problem (e.g., the number of examples, the number of distinguished labels or the pattern size), and see how this affects the complexity.

We note that the learning problem is at least as hard as the satisfiability problem. Indeed, in some cases, satisfiability may be in polynomial time, but no polynomial size satisfying pattern may exist. For the most part in this paper, we focus on satisfiability, as this problem is usually already hard (i.e., not in polynomial time, unless P = NP). However, for cases in which satisfiability is polynomial, we also consider the learning problem, and show how to find a satisfying pattern (or compact representation of a satisfying pattern) in polynomial time. Later, in Section 6, we consider a third problem of interest – that of finding a minimal-sized satisfying pattern.

Table 1 Complexity of satisfiability for positive datasets.

	Pattern Features	Embedding Type	Graph Type	Data Set	Additional Conditions	Complexity
1.1	{//}, {∗, //}	–	–	–	–	PTIME (Thm. 5)
1.2	–	–	–	–	BoundP[1]	PTIME (Thm. 5)
1.3	∅	–	–	unq	–	PTIME (Thm. 5)
1.4	–	1:m	–	–	BoundE[2]	PTIME (Thm. 7)
1.5	{*}	1:m	udt	oident, unq	–	PTIME (Thm. 7)
1.6	{*}	1:m	udt	any	ConD[3]	PTIME (Thm. 7)
1.7	{*}	1:m	udt	any	BoundLD[4]	PTIME (Thm. 7)
1.8	{*}	1:1	–	–	–	NPC (Thm. 8)
1.9	∅	1:1	–	oident, any	–	NPC (Thm. 8)
1.10	{*}	1:m	drt	–	–	NPC (Thm. 9)
1.11	{*}	1:m	udt	any	–	NPC (Thm. 9)
1.12	∅	1:m	udt	oident, any	–	NPH (Thm. 10)
1.13	∅	1:m	drt	oident, any	–	PSPACE (Thm. 10)

[1] *bounded number of nodes in patterns*
[2] *bounded number of examples*
[3] *connected dataset*
[4] *bounded number of distinct labels in dataset*

4 Positive Datasets

In this section we consider the satisfiability and learning problems for positive datasets. Table 1 summarizes the complexity of satisfiability for a variety of settings. Note that:

- The first column is simply a running number, that will be convenient for reference later in Table 3.
- The second column specifies features allowed in the pattern. All patterns allow labeled nodes and child edges. We note the set of additional features (wildcards and/or descendant edges) allowed in each row. For example, the first row considers the complexity of satisfiability when patterns can contain descendant edges (and possibly wildcards) and the third row considers the complexity of satisfiability when no special features (neither wildcards nor descendant edges) are allowed.
- The third column specifies the type of satisfaction (i.e., embedding) allowed. We use 1:1 to denote injective embeddings and 1:m to denote arbitrary embeddings.
- The fourth column indicates whether the graphs (in the dataset and pattern) are undirected (udt) or directed (drt).
- Finally, the fifth column indicates whether the dataset is uniquely labeled (unq), is output identifying (oident) or can be arbitrary (any). Note that every dataset that is uniquely labeled is also output identifying.

For table cells in which no value is specified (i.e., "–" appears), the complexity result holds regardless of the value of the cell. Due to lack of space, all proofs are deferred to the appendix. However, next to each theorem, we state the intuition behind the proof.

In the remainder of this section, we discuss the complexity of satisfiability and learning, with an emphasis on when and why the problem becomes tractable or intractable. The following example demonstrates one of the difficulties, namely that it is sometimes the case that all satisfying patterns will be exponential in the size of the input.

▶ **Example 4.** Consider the dataset in Figure 2, defined as $\Delta = (\{g_1^+, g_2^+, g_3^+\}, (S, F))$. It is not difficult to see that the smallest tree pattern which satisfies all examples has

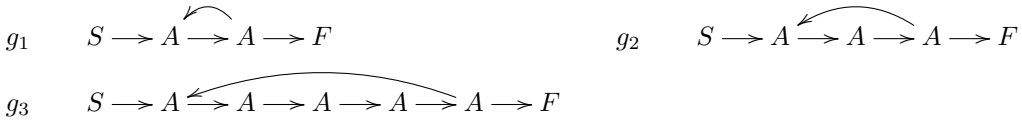


Figure 2 Dataset $\Delta = (\{g_1^+, g_2^+, g_3^+\}, (S, F))$, for which satisfying patterns are large.

$1 + 2 * 3 * 5 + 1 = 32$ nodes – it is simply a path starting with a node labeled S, and then containing 30 consecutive nodes labeled A, and finally ending with a node labeled F (with the first and last nodes in the path being the output nodes).

Example 4 demonstrates that we cannot always expect the problem of learning from positive examples to be polynomial, as the pattern that must be learned may itself be *at least* exponential in size. The precise complexity of satisfiability and learning, given various restrictions of the settings, is discussed next.

Polynomial Cases. We start by considering cases where both satfiability/injective satisfiability and learning/injective learning, are tractable. Theorem 5 presents three such cases. Intuitively, they all follow from the fact that the given restrictions imply that (1) there are polynomially many (small) patterns that must be considered, in order to find a satisfying pattern, should one exists *and* (2) it is possible to check in polynomial time whether such a pattern satisfies the dataset.

▶ **Theorem 5.** *The (injective) satisfiability and (injective) learning problems are in polynomial time for positive datasets, if one of the following conditions holds*

1. *patterns may contain descendant edges;*
2. *patterns are bounded in size;*
3. *patterns can contain neither wildcards nor descendant edges,* and *datasets are uniquely labeled.*

Intuitively, (1) holds since when patterns can contain descendant edges, (injective) satisfiability and (injective) learning reduce to the problem of graph reachability. Case (2) is immediate, since it implies that there are a bounded number of patterns, each of bounded size, that must be considered. We note that this is useful in practice, as the user may sometimes only be interested in viewing small patterns, and may disregard large patterns as appearing by chance.

To show Case (3), we need the following definition. Let $g_1, \ldots, g_m$ be graphs. We denote by $g_1 \otimes \cdots \otimes g_m$ the *multiplication graph* g defined as follows:

- $V_g \subseteq V_{g_1} \times \cdots \times V_{g_m}$ is defined as the set $\{(v_1, \ldots, v_m) \mid l_{g_1}(v_1) = \cdots = l_{g_m}(v_m)\}$;
- $l_g((v_1, \ldots, v_m)) = l_{g_1}(v_1)$;
- $((v_1, \ldots, v_m), (u_1, \ldots, u_m)) \in E_g$ if and only if $(v_i, u_i) \in E_{g_i}$ for all $i \leq m$;

The following proposition is easy to show.

▶ **Proposition 6.** *Let $\Delta = (\{g_1^+, \ldots, g_m^+\}, (l_1, \ldots, l_n))$ be a dataset. Let $g = g_1 \otimes \cdots \otimes g_m$. There exists a pattern p containing no descendant edges and no wildcards that satisfies Δ if and only if there are sets $V \subseteq V_g$ and $E \subseteq E_g$, and a tuple of nodes $(o_1, \ldots, o_n)$ from V such that all three of the following conditions hold:* (1) *(V, E) is a tree,* (2) *$l_g(o_i) = l_i$ for all $i \leq n$ and* (3) *every leaf node in (V, E) is in $(o_1, \ldots, o_n)$.*

We note that Proposition 6 deals with satisfaction, and does not specifically consider injective satisfaction. Notwithstanding, Case (3) of Theorem 5 holds for both satisfiability

and injective satisfiability. Intuitively, the proof follows from the fact that the multiplication graph is polynomial in size when the dataset is uniquely labeled, and from the fact that the existence of an embedding implies the existence of an injective embedding in this case.[1]

We now consider cases in which satisfiability and learning is tractable, but the injective versions are not. (We show intractability of the injection versions of these problems in the next section.)

▶ **Theorem 7.** *The satisfiability problem for positive datasets is in polynomial time, if one of the following conditions holds:*

1. *the number of examples is bounded by a constant;*
2. *patterns can contain wildcards, the dataset is undirected and also*
   - **(a)** either *is output identifying,*
   - **(b)** or *is connected (i.e., all examples are connected graphs),*
   - **(c)** or *contains a bounded number of distinguished labels.*

Moreover, a satisfying pattern can be found and compactly represented in polynomial time and space.

To prove Case (1) of Theorem 7, we show that if Δ is a satisfiable positive dataset containing m examples, each of which contains a graph with at most k nodes, then there exists a pattern satisfying Δ of size at most k^m. Thus, if the number of examples is bounded, then there exists a satisfying pattern of size polynomial in the input. Note that such patterns can be enumerated and their satisfaction verified. (Injective satisfaction cannot, however, be efficiently verified, and thus, we will see later that two examples are sufficient for intractability of injective satisfiability.)

We now consider Case (2) of Theorem 7. This is proven by showing that if there exists a satisfying pattern, then there is one of a special format, called an *output-rooted pattern*. Such patterns $(t, (o_1, \ldots, o_k))$ have only wildcard labeled nodes, and consist of a path from o_1 to each o_i for $1 < i \leq k$. This format allows compact representation (as we only have to store the path lengths, and not the actual paths). Moreover, checking for the existence of such a satisfying output-rooted pattern can be reduced to the problem of determining the existence of paths of odd or even lengths between pairs of nodes, which is solved by 2-coloring the graph. This holds as whenever there is a path of odd (resp. even) distance between two nodes, it can be extended to arbitrarily large paths of odd (resp. even) distance, by repeatedly traversing back and forth over some edge in the path.

Intractable Cases. We now consider cases in which satisfiability, or injective satisfiability, is intractable (unless P = NP). We start by considering injective satisfiability for positive datasets, and show that in every case not covered by the previous section, the problem is NP-complete. We note that membership in NP is immediate, since only injective embeddings are allowed, and hence, one must only consider tree patterns which are at most the size of the smallest graph in any example. NP-hardness is shown by a reduction to the Hamiltonian path problem.

▶ **Theorem 8.** *Injective satisfiability is NP-complete for positive datasets if patterns may contain wildcards, or if datasets are not uniquely labeled. This holds even if there are only two examples in the dataset, and even if there are only two distinguished labels in the dataset.*

[1] Note that the tuple of distinguished labels of a dataset cannot contain repeated labels, by definition.

Next, we consider the satisfiability problem. Theorem 7 showed cases in which the presence of wildcards makes satisfiability polynomial. In Theorem 9 we show that in all remaining cases (either directed datasets, or undirected datasets of arbitrary type), if wildcards are allowed, satisfiability becomes NP-complete.

▶ **Theorem 9.** *When patterns can contain wildcards, satisfiability is NP-complete if one of the following conditions holds:*

1. *the dataset is directed;*
2. *the dataset is undirected, is not output identifying, has an unbounded number of distinguished labels* and *examples may be unconnected graphs.*

For Case (1) of Theorem 9, both NP-hardness and membership in NP are not easy to show. For NP-hardness, our proof is by a reduction from 3-SAT. However, since patterns can contain wildcards, the reduction is intricate, as labels on nodes can no longer be used to differentiate between truth assignments. Instead, in our proof we create examples containing cycles of different prime lengths, and employ the Chinese Remainder Theorem to show that the dataset constructed is satisfiable if and only if the given 3-SAT formula is satisfiable.

The principle difficulty in showing membership in NP is that satisfying patterns may be exponential in size. Thus, we must show that if there is a satisfying pattern, we can find one representable in polynomial size, and also verify the correctness of such a pattern in polynomial time. The crux of the proof lies in two claims that we show. First, if there exists a satisfying pattern, then there is one that has a very simple form (a tree which branches only at the root), with paths that are not more than exponentially long, and hence, can be compactly written. Second, it is possible to determine in polynomial time whether there is a (not necessarily simple) path of a given length l from a given node u to another node v.

For Case (2) of Theorem 9, Membership in NP follows from the fact that if a satisfying pattern exists, then it can be polynomially represented using an output-rooted pattern. NP-hardness is shown by a reduction from 3-SAT.

The remaining case to be considered is when both wildcards and descendent edges are prohibited, and the dataset is not uniquely labeled. For undirected graphs, the precise complexity of satisfiability is unknown; however satisfiability is at least NP-hard. For directed graphs, satisfiability is PSPACE complete.

▶ **Theorem 10.** *When patterns cannot contain wildcards, satisfiability of positive datasets is NP-hard for graphs that are undirected and PSPACE-complete for graphs that are directed. This holds even if there are only two distinguished labels in the dataset, and even if the dataset is output identifying.*

NP-hardness for undirected graphs is shown by a reduction from 3-SAT. PSPACE-completeness for directed graphs is shown by a reduction from the problem of determining emptiness of the intersection of deterministic finite automata [18].

5 Datasets with Positive and Negative Examples

In this section, we explore (injective) satisfiability and (injective) learning for datasets that may contain both positive and negative examples. Unsurprisingly, these problems are usually significantly harder when negative examples are allowed. We present comprehensive results on the complexity of satisfiability and learning. For the injective versions of these problems we also present complexity results; however, there are some remaining open problems. Note that, as before, we primarily consider the (injective) satisfiability problem, but in cases where it is shown to be polynomial, we also consider (injective) learning. See Table 2 for a summary of the satisfiability results of this section.

Table 2 Complexity of satisfiability for arbitrary datasets (which may contain both positive and negative examples).

	Pattern Features	Embedding Type	Graph Type	Data Set	Additional Conditions	Complexity
2.1	–	–	–	–	BoundP[1]	PTIME (Thm. 11)
2.2	$\emptyset$, {//}	1:m	udt	unq	–	PTIME (Thm. 12)
2.3	{∗}, {∗, //}	1:m	udt	unq	BoundLD[2]	PTIME (Thm. 12)
2.4	$\emptyset$, {//}	1:1	–	unq	–	NPC (Thm. 13)
2.5	–	1:m	drt	unq	–	NPC (Thm. 14)
2.6	{∗}, {∗, //}	1:m	udt	unq	–	NPC (Thm. 14)
2.7	–	1:m	–	oident, any	–	PSPACE (Thm. 15)
2.8	–	1:1	–	–	–	NPH, Co-NPH, in Σ_2^P (Thm. 16)

[1] *bounded number of nodes in patterns* [2] *bounded number of distinct labels in dataset*

Polynomial Cases. When datasets can contain negative examples, there are only few cases in which satisfiability is tractable. In particular, for the special case where the patterns are of bounded size, the problem remains polynomial. As before, this holds as there are a polynomial number of different bounded patterns that must be considered, and all embeddings can be polynomially enumerated and verified.

▶ **Theorem 11.** *(Injective) Satisfiability and (injective) learning of datasets is in polynomial time, regardless of the type of dataset and pattern, if the tree patterns are bounded in size.*

Theorem 12 presents two additional cases in which satisfiability and learning are in polynomial time. The first case, in which patterns cannot contain wildcards, is easily verified using the multiplication graph of the positive examples. The second case, which allows for wildcards, is shown by proving that the number of different patterns that must be considered is polynomial in the size of the input. Once again, this requires the ability to bound lengths of pattern paths over undirected graphs.

▶ **Theorem 12.** *Satisfiability and learning of uniquely-labeled undirected datasets is in polynomial time, if one of the following conditions hold:*
1. *patterns cannot contain wildcards;*
2. *patterns can contain wildcards, but only have a bounded number of distinguished labels.*

Intractable Cases. Theorem 12 showed cases in which the fact that the dataset is uniquely labeled yields tractability. Unfortunately, the following two theorems shows that this is not always the case. First, we show that Case (1) of Theorem 12 no longer holds if *injective* satisfiability is desired.

▶ **Theorem 13.** *The injective satisfiability problem is NP-complete if the dataset is uniquely labeled and the patterns cannot contain wildcards.*

For the non-injective case, the following theorem shows that every case of uniquely labeled datasets, other than those considered in Theorem 12, is NP-complete.

▶ **Theorem 14.** *Satisfiability of uniquely-labeled datasets is NP-complete, if one of the following conditions holds*
1. *the dataset is directed;*
2. *the patterns can contain wildcards, and there is an unbounded number of distinguished labels.*

For Case (1), NP-hardness is shown by a reduction from 3-SAT. Membership in NP is shown by proving that if a satisfying pattern exists, then there is one that has a representation that is polynomial in the size of the input. We note that proving the latter (i.e., membership) is quite intricate, and is based extensively on the fact that the datasets are uniquely-labeled. Proving this result requires, among other claims, the ability to bound the length of paths in a satisfying pattern. Thus, for example, we can show that in a directed path with k nodes, whenever there is a path from a node u to a node v of length $c > 2k^2 \cdot k!$, there is also a path from node u to node v of length $c - k!$.

For Case (2), membership is shown using the techniques from Theorem 12, while hardness is shown by a reduction from 3-SAT.

The previous theorems fully cover the cases in which the dataset is uniquely labeled. For datasets that are not uniquely labeled, satisfiability is PSPACE complete, regardless of whether the dataset is directed or undirected, and regardless of the allowed features in the patterns. In fact, whereas bounding the number of examples was sufficient to achieve polynomial time for determining satisfiability of positive datasets, this is no longer the case when negative examples are allowed. Satisfiability is PSPACE complete, for datasets that are output identifying or arbitrary, even when there is only a single positive example and a constant number of negative examples.

▶ **Theorem 15.** *Satisfiability of datasets that are output identifying or arbitrary is PSPACE-complete. This holds even if there are only two distinguished labels, a single positive example and a constant number of negative examples.*

We show the above result by a reduction to the problem of equivalence of NFAs [24]. This is achieved by creating examples in which satisfying patterns corresponds to words in an NFA. We note that even when wildcards and descendant edges are technically allowed within patterns, these features can be effectively ruled out using appropriately defined negative examples. We note also that it is possible to use an undirected graph to simulate an automaton (which is, in essence, a directed graph).

We now consider one remaining case of injective satisfiability. For arbitrary graphs, injective satisfiability is NP-hard and Co-NP-hard, and is in Σ_2^P. NP-hardness follows from Theorem 8, Co-NP-hardness can be shown by a reduction to the Hamiltonian path problem, and containment in Σ_2^P is immediate.

▶ **Theorem 16.** *The injective satisfiability problem is NP-hard, Co-NP-hard and in Σ_2^P if the dataset can contain both positive and negative examples, and either the dataset is not uniquely labeled, or patterns can contain wildcards.*

6 Learning Minimal Tree Patterns

The previous sections dealt with the (injective) satisfiability and (injective) learning problems, i.e., determining whether there exists a satisfying pattern for a dataset, and finding an arbitrary such pattern. In this section we focus on finding a *minimal* satisfying pattern, i.e., a satisfying pattern of minimal size.

Formally, given a dataset Δ, we say that a pattern $p = (t, \bar{o})$, satisfying Δ, is *minimal* if there does not exist a pattern $p' = (t', \bar{o}')$ such that *(1)* p' satisfies Δ and *(2)* t' has less nodes than t. This section studies the following problem:

▶ **Problem 3** (Minimal (Injective) Learning)**.** Given a dataset Δ, find a minimal pattern p that (injectively) satisfies Δ.

Table 3 Complexity of minimal learning problem.

Case	Num of Dist. Labels	Pos/ Neg	Pattern Features	Embedding Type	Graph Type	Data Set	Additional Conditions	Complexity
1.1	–	+	{//}, {∗, //}	–	–	–	–	PTIME (Thm. 17)
1.2, 2.1	–	–	–	–	–	–	BoundP[1]	PTIME (Thm. 17)
1.3–1.7	var	+	∅, {∗}	–	–	–	–	NPC (Thm. 19)
1.3	const	+	∅	–	–	unq	–	PTIME (Thm. 20)
1.4	const	+	–	1:m	–	–	BoundE[2]	PTIME (Thm. 20)
1.5–1.7	const	+	{∗}	1:m	udt	any	–	PTIME (Thm. 18)
2.2, 2.3	–	±	–	1:m	udt	unq	–	NPC (Thm. 21)

[1] *bounded number of nodes in patterns* [2] *bounded number of examples*

Obviously, the problem of finding a minimal satisfying pattern is no easier than determining satisfiability or finding an arbitrary satisfying pattern. Under most conditions, both of the latter problems are intractable. However, in the previous sections we identified several cases in which satisfiability and learning are polynomial-time problems. For these cases, we study the complexity of the problem of finding a minimal satisfying pattern. Finding such patterns is clearly a problem of practical interest.

The complexity results are summarized in Table 3. *We emphasize that the only cases considered here are those for which satisfiability and learning have previously been shown to be in polynomial time. Thus, cases not considered in this table are certainly intractable.* All columns other that the first three, appeared in previous tables. In addition:

- The first column indicates the number in Table 1 or 2 of the case considered.
- The second column indicates whether the number of distinguished labels in the dataset is assumed to be constant (const) or whether this number is a variable of the input (var). We note that it is natural to assume that the number of distinguished labels is constant, as it is likely to a small number (whereas the examples may be large). As indicated in the table, often the assumption that the number of distinguished labels is constant is sufficient to achieve polynomial time.
- The third column indicates whether the dataset contains only positive examples (+) or both positive and negative (±). In previous sections this column was not needed as these two different cases were presented in different tables.[2]

In the remainder of this section, we prove the complexity results summarized in Table 3. We divide up the discussion by the type of proof technique used to derive the results.

Polynomial Number of Candidates. In some cases, given Δ, it is possible to define a polynomial size set of patterns P, for which it is guaranteed that if Δ is satisfiable, then P must contain a minimal-size satisfying pattern. We will say that P is the set of minimal pattern *candidates*. If P is polynomial in size, and can be efficiently found, then minimal learning is clearly in polynomial time. The next theorem deals with two cases in which such a polynomial size set of candidates can be efficiently found, and thus, minimal satisfying and injectively satisfying patterns can be found in polynomial time. Both of these cases are quite straightforward – descendent edges significantly simplify the candidates that must be considered (in the first case), and the second case explicitly bounds the size of the candidates considered.

[2] As before, columns with "–" allow for any value, without affecting problem complexity.

▶ **Theorem 17.** *Let Δ be a satisfiable dataset. Then, it is possible to find a minimal (injectively) satisfying pattern in polynomial time if any of the following conditions hold:*
1. *Δ is positive, and patterns can contain descendant edges;*
2. *there is a constant k, such that only patterns containing at most k nodes are to be returned;*

The following theorem presents an additional case where the set of candidates for satisfaction is polynomial. Note that bounding the number of distinguished labels infers a bound on the number of leaf nodes in the patterns, which can be used to significantly reduce the number of patterns considered.

▶ **Theorem 18.** *Let Δ be a satisfiable dataset with a constant number of distinguished labels. Then, it is possible to find a minimal satisfying pattern in polynomial time if Δ is positive and undirected, and patterns can contain wildcards.*

Reductions to the Steiner Tree Problem. We consider cases in which the problem of finding a minimal satisfying pattern is similar to the problem of finding a Steiner tree in a graph. The Steiner tree decision problem is: given a graph g, a set of nodes V, and a number k, find a subtree of g of size at most k, containing V. It is well known that the Steiner tree problem is NP-complete. However, it is solvable in polynomial time if the number of nodes in V is constant, both if the graph is undirected [9], and if the graph is directed [20] (and a rooted Steiner tree is desired).

▶ **Theorem 19.** *Let Δ be a positive dataset with an unbounded number of distinguished labels and let k be a positive integer. Then, the problem of determining whether there exists a pattern p that (injectively) satisfies Δ and contains at most k nodes is NP-complete if no descendant edges are allowed in the pattern.*

Membership is easy, as given a pattern with k nodes, we can guess and verify embeddings in polynomial time. Hardness is shown by a reduction from the Steiner tree problem.

When considering datasets with a constant number of labels, the minimal learning problem sometimes becomes tractable. This is the case, in particular, in Theorem 20. The proof leverages the aforementioned polynomial case for Steiner trees. In particular, in both cases below, the multiplication graph of the examples is guaranteed to be polynomial in size, and a Steiner tree in this graph can be used to generate a minimal satisfying pattern.

▶ **Theorem 20.** *Let Δ be a positive dataset with a constant number of distinguished labels. Then, it is possible to find a minimal satisfying pattern in polynomial time if one of the following conditions hold:*
1. *Δ is uniquely labeled, and no wildcards are allowed in the patterns; or*
2. *Δ contains a constant number of examples.*

Datasets With Negative Examples. The final case that must be considered is one in which the dataset can contain both positive and negative examples. Bounding the number of distinguished labels no longer leads to tractability in this case.

▶ **Theorem 21.** *Let Δ be a dataset with both positive and negative examples, and with two distinguished labels. Let k be a positive integer. Then, the problem of determining whether there exists a pattern p that satisfies Δ and contains at most k nodes is NP-complete if Δ is uniquely labeled and undirected.*

Once again, membership in NP is immediate. We show NP-hardness by using the same reduction from 3-SAT as in the proof of Theorem 13.

7 Conclusion

In this paper we extensively studied three problems – satisfiability, learning and minimal learning, for datasets containing positive and negative example graphs, and a tuple of distinguished labels. We have shown that the complexity of these problems is highly dependent on the precise setting, i.e., that the presence or absence of specific features can make these problems significantly easier or harder.

Many interesting problems have been left open for future work. First, from a practical standpoint, we intend to investigate how these results can be integrated into a system, in order to allow users to provide examples, instead of explicit queries. A second problem of interest is to develop an algorithm that can find top-k satisfying patterns (preferably of diverse structure), for some appropriately defined ranking problem,. Third, we intend to study the problem of maximally satisfying the given examples, when there is no pattern that can satisfy all examples. Finally, we intend to study classic machine learning questions arising from this setting, such as investigating in which cases any specific hypothesis can be guaranteed to be learned, given that the system can provide examples to the user, and specifically ask whether these examples are positive. In machine learning terminology, this is the problem of learning patterns from (equivalence/membership/subset etc.) queries.

Acknowledgments. The authors were supported by the Israel Science Foundation (Grant 1467/13) and the Ministry of Science and Technology (Grant 3-9617).

References

1 Thomas Amoth, Paul Cull, and Prasad Tadepalli. On exact learning of unordered tree patterns. *Machine Learning*, 44:211–243, 2001.

2 Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, July 1990.

3 Timos Antonopoulos, Frank Neven, and Frédéric Servais. Definability problems for graph query languages. In *Proceedings of the 16th International Conference on Database Theory*, pages 141–152, New York, NY, USA, 2013. ACM.

4 Hiroki Arimura, Hiroki Ishizaka, and Takeshi Shinohara. Learning unions of tree patterns using queries. *Theor. Comput. Sci.*, 185(1):47–62, 1997.

5 Julien Carme, Michal Ceresna, and Max Goebel. Query-based learning of XPath expressions. In *ICGI*, 2006.

6 Adriane Chapman and H. V. Jagadish. Why not? In *SIGMOD*. ACM, 2009.

7 Sara Cohen and Yaacov Y. Weiss. Certain and possible XPath answers. In *ICDT*, 2013.

8 Anish Das Sarma, Aditya Parameswaran, Hector Garcia-Molina, and Jennifer Widom. Synthesizing view definitions from data. In *ICDT*, 2010.

9 S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

10 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

11 Melanie Herschel, Mauricio A. Hernández, and Wang-Chiew Tan. Artemis: a system for analyzing missing answers. *Proc. VLDB Endow.*, 2:1550–1553, August 2009.

12 Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.

13 Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

14 Chuntao Jiang, Frans Coenen, and Michele Zito. A survey of frequent subgraph mining algorithms. *Knowledge Eng. Review*, 28(1):75–105, 2013.

15 Benny Kimelfeld and Phokion G. Kolaitis. The complexity of mining maximal frequent subgraphs. In *PODS*, 2013.

16 Benny Kimelfeld and Yehoshua Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, 2006.

17 Raymond Kosala, Maurice Bruynooghe, Jan Van Den Bussche, and Hendrik Blocked. Information extraction from web documents based on local unranked tree automaton inference. In *IJCAI*, 2003.

18 D. Kozen. Lower bounds for natural proof systems. In *FOCS*, 1977.

19 Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. WHY SO? or WHY NO? Functional Causality for Explaining Query Answers. In *Management of Uncertain Data*, 2010.

20 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. *J. Comb. Optim.*, 24(2):131–146, 2012.

21 Rika Okada, Satoshi Matsumoto, Tomoyuki Uchida, Yusuke Suzuki, and Takayoshi Shoudai. Exact learning of finite unions of graph patterns from queries. In *Algorithmic Learning Theory*, LNCS, pages 298–312. Springer Berlin Heidelberg, 2007.

22 Stefan Raeymaekers, Maurice Bruynooghe, and Jan Bussche. Learning (k,l)-contextual tree languages for information extraction from web pages. *Machine Learning*, 71(2-3):155–183, June 2008.

23 Slawek Staworko and Piotr Wieczorek. Learning twig and path queries. In *ICDT*, 2012.

24 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *STOC*, 1973.

25 Quoc Trung Tran and Chee-Yong Chan. How to conquer why-not questions. In *SIGMOD*, 2010.

26 Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. Query by output. In *SIGMOD*. ACM, 2009.

Characterizing XML Twig Queries with Examples

Sławek Staworko[1] and Piotr Wieczorek[2]

1 **University of Lille 3, INRIA LINKS, CNRS**
Lille, France
slawomir.staworko@inria.fr

2 **University of Wrocław, Institute of Computer Science**
Wrocław, Poland
piotrek@cs.uni.wroc.pl

Abstract

Typically, a (Boolean) query is a finite formula that defines a possibly infinite set of database instances that satisfy it (*positive* examples), and implicitly, the set of instances that do not satisfy the query (*negative* examples). We investigate the following natural question: for a given class of queries, is it possible to *characterize* every query with a finite set of positive and negative examples that no other query is consistent with.

We study this question for *twig* queries and XML databases. We show that while twig queries are characterizable, they generally require exponential sets of examples. Consequently, we focus on a practical subclass of *anchored* twig queries and show that not only are they characterizable but also with polynomially-sized sets of examples. This result is obtained with the use of *generalization operations* on twig queries, whose application to an anchored twig query yields a properly contained and minimally different query. Our results illustrate further interesting and strong connections between the structure and the semantics of anchored twig queries that the class of arbitrary twig queries does not enjoy. Finally, we show that the class of unions of twig queries is not characterizable.

1998 ACM Subject Classification H.2.3 Query languages, H.2.1 Normal forms

Keywords and phrases Query characterization, Query examples, Query fitting, Twig queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.144

1 Introduction

One of the central, if not defining, instruments in computer science is using a formula, a finite syntactic object, to define a (possibly infinite) set of its models. A typical example are regular expressions that define languages of words. Database queries also fall into this category, which is best illustrated with Boolean queries: a query q defines the set of instances satisfying q, *positive examples*, and implicitly the set of instances that do not satisfy q, *negative examples*. In this paper, we study the question of (finite) *characterizability*: Can every query be characterized with a finite set of examples? More precisely, given a class of queries $\mathcal{Q}$ is it possible for every $q \in \mathcal{Q}$ to find a set of examples such that q is the only query (modulo equivalence) in $\mathcal{Q}$ *consistent* with it i.e., a query satisfying all positive examples and none of negative examples. And if it is possible, can we say anything about the number and the sizes of the necessary examples?

The question of characterizability arises naturally in the context of learning/teaching [9, 6] which deals with the problem of constructing a formula (query) consistent with a given set of examples. However, research on characterizability has a number of potential applications of independent interest because it yields way to generate a set of examples consistent with a

given query. Such examples can be used, for instance, for elementary query engine debugging, and query visualization and explanation. The exhaustive nature of the examples provided by characterizability i.e., there is exactly one query satisfying the examples, may also be useful in final, verification, stages of reverse engineering of database queries.

In this paper, we investigate the problem of characterizability for XML databases and twig queries [2, 14]. XML documents can be seen as labeled unranked trees and twig queries are tree-shaped patterns that additionally use a wildcard label $\star$ (matching any label) and descendant edges (matching a path of positive length). Twig queries are the core of virtually any XML query language and have been extensively studied in the literature [4]. In particular, learning twig queries from examples has been previously investigated [19, 5] and the current paper can be seen as a continuation of this line of work and an attempt at deepening our understanding of the relationship between a query and its examples.

In essence, our results show that characterizability is a measure of richness of the query class, which is closely related to its expressive power. We show that unions of twig queries are not characterizable because virtually any set of examples has infinitely many consistent queries i.e., the class of unions of twig queries is very rich. On the other hand, the class of twig queries is less rich, given a tree the number of twig queries satisfied in it is finite, which allows to show that twig queries are characterizable.

While twig queries are characterizable with finite sets of examples, we show that the number of necessary examples may be exponential. The main contribution of this paper is showing that (polynomially) small sets of examples are sufficient to characterize *anchored twig queries* [19]. In essence, this subclass of twig queries forbids descendant edges incident to a $\star$ node, which merely prevents from imposing conditions on ancestry with a lower bound on depth: "a node x is a ancestor of node y and the path from x to y is of length $\geq k$", where $k > 1$. While the expressive power does not seem to be significantly restricted from the practical point of view, this class of twig queries exhibits a very close relationship between the structure and the semantics: containment of two twig queries is equivalent to the existence of an embedding between the queries, an equivalence that does not hold for arbitrary twig queries [13]. This relationship between the structure of the query and its semantics goes even further [19]: the use of two positive examples of an anchored query q allows to identify all queries that contain q. We continue to explore this relationship and deepen its understanding by characterizing the structure of the semi-lattice of anchored twig queries: for a given anchored twig query q we are interested in the number and the sizes of the most specific anchored twig queries properly containing q, and interestingly, we show that their number and their sizes are polynomially small (w.r.t. the size of q). This result is essential in showing that anchored twig queries have polynomially-sized characterizing sets of examples.

To understand the existence of an embedding, and hence the containment, between two anchored twig queries, we study three generalization operations applied to twig queries (cf. Fig. 5): 1) changing a label to $\star$, 2) changing the type of an edge from child to descendant, and 3) removing a node. While natural and elementary, these operations capture precisely the subclass of *injective* (*ancestor-preserving*) embeddings between queries: query p can be obtained from a query q by applying a sequence of generalization operations if and only if there is an injective embedding of q into p. Consequently, when applied, in a diligent manner, to anchored twig queries they allow to characterize the semi-lattice of anchored twig queries under injective semantics. We also show that anchored twig queries have unique canonical forms which can be efficiently obtained by iteratively applying the operations and the unique canonical form is in fact the (size-)minimal equivalent query. This further

illustrates the desirable properties of anchored twig queries as minimization for arbitrary twig queries is known to be intractable [2, 13]. We point out, however, that classes of twig queries properly containing anchored twig queries are known to have tractable minimization based on operations reducing the query [11].

To extend the characterization results from injective to standard semantics, we identify mapping overlap as the essential difference between the corresponding two type of embeddings: unlike the injective embedding, the standard embedding of a query q into a query p may map different fragments of q into the same fragment of p creating an overlap of the images of the different fragments of q. To address this difference we explore using a duplication operation that creates separate copies of the fragment of p thus eliminating the overlap. However, this operation introduces redundancies and may increase arbitrarily the size of the query. Consequently, it is applied together with different generalization operations to avoid introducing redundancies and to avoid undesirable growth of the query we devise a recursive pattern of applying duplication operations that allows to polynomially bound the number of introduced copies. The number of the generalizations is linear in the size of the original query, and thanks to applying the duplication operation in a controlled manner, the size of each generalization is at most quadratic. The generalizations are then used to construct a set of characterizing examples consisting of a polynomial number of examples each of polynomially bounded size.

The main contributions of the paper are:

- We formulate the problem of characterizability of a Boolean class of XML queries with examples and study it for classes of twig queries.
- We show that unions of twig queries are not characterizable while twig queries alone are but the number of examples necessary to characterize a twig query may be exponential in the size of the query.
- We investigate characterizability of a rich subclass of anchored twig queries, and propose a set of natural generalization operations that allows to characterize with a polynomially-sized set of examples any anchored twig query under the injective (ancestor-preserving) semantics.
- We propose a duplication operation whose diligent use allows to extend the characterizability to anchored twig queries to the standard semantics.

Related work. Our work is closely related to the scenario of *teaching* [10, 9, 16] where the set of examples to be presented is selected by a helpful teacher. Goldman and Kearns [9] define a sequence of positive and negative examples to be a *teaching sequence* for a given concept c if it uniquely specifies c in the concept class C. Hence, this is essentially the same idea as in our case. They study the properties of a *teaching dimension* of a concept class that is a minimum number of examples a teacher has to reveal in order to uniquely identify any concept in the class. They consider, however, different concept classes then ours, namely orthogonal rectangles and boolean formulas. Also, teaching sequences for other classes of boolean formulas has been studied in [18, 3].

Recently, learning and verification of *qhorn* queries was studied in [1]. *qhorn* is a special class of Boolean quantified queries whose underlying form is conjunctions of quantified Horn expressions. In order to verify that a given query is equivalent to the one intended by a user, a unique *verification set* of polynomial size is constructed. The verification set consists of examples uniquely determining the given query. The verification algorithm classifies some questions in the set as answers (positive examples) and others as non-answers (negative examples). The query is incorrect if the user disagrees with any of the query's classification of questions in the verification set.

A number of notions of characterizability has been studied in the context of grammatical inference [6]. Their work is related to Gold's classical model [7, 8]. In the first approach characteristic samples are constructed for a given algorithm. The algorithm must return a concept consistent with the given sample and for any sample extending the characteristic sample for the concept c, it has to return c. In another variant characterizability is parametrized with a set I of algorithms, i.e., for each concept c, its characteristic set must allow any algorithm from I to identify c. Finally, they introduce the following notion: a concept class C is *polynomially characterizable* iff for each concept c there exists a characteristic sample S of polynomial size such that if another non-equivalent concept c' is compatible with S then c is not compatible with the characteristic sample for c'. In our case q is the only query that is consistent with the characteristic sample for q.

In the learning scenario the main objective is to find a learning algorithm that produces a formula (query) consistent with a given set of examples [7, 8, 6, 19, 20]. One typically uses a weaker notion of characteristic sample w.r.t. a given learning algorithm. This allows learner bias, *collusion*, e.g., using a fixed order on the alphabet in language inference. Characterizability is stronger because it implies the existence of a characterizing sample independent of the learning algorithm (no bias).

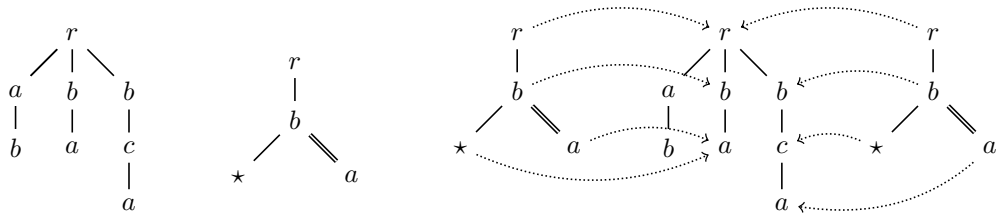
Organization. In Section 2 we recall basic notions on XML and twig queries. In Section 3 we formalize the problem of characterizing queries with examples and show that unions of twig queries are not characterizable. In Section 4 we show that twig queries are characterizable but may require exponentially many examples. In Section 5 we recall anchored twig queries and their fundamental properties. In Section 6 we present basic generalization operations, show their connection with injective embeddings, and show how to use them to characterize anchored twig queries under the injective semantics with polynomially small examples. In Section 7 we show that the generalization operations can be used to minimize anchored twig query and then show how to extend the approach used in Section 6 to construct polynomially small sets of examples characterizing anchored twig queries in the standard semantics. In Section 8 we summarize our results and outline future directions of study.

Acknowledgments. This paper is partially supported by the Polish National Science Centre grant DEC-2013/09/B/ST6/01535.

2 Basic notions

In this section we recall basic notions used to model XML documents and twig queries. Throughout this paper we assume an infinite set of node labels Σ which allows us to model documents with textual values. Also, we fix one special label $r \in \Sigma$ that we use on the root nodes of all trees and queries.

Trees. We model XML documents with unranked trees whose nodes are labeled with elements of Σ. Formally, a *tree* t is a tuple $(N_t, root_t, parent_t, lab_t)$, where N_t is a nonempty finite set of nodes, $root_t \in N_t$ is the root node, $parent_t : N_t \setminus \{root_t\} \to N_t$ is a child-to-parent function, and $lab_t : N_t \to \Sigma$ is a labeling function. By *Tree* we denote the set of all trees. An example of a tree is presented in Fig. 1a. We define a number of additional notions. A *leaf* is any node that has no children. A *path* in t from n to m (of length k) is a sequence of nodes $n = n_1, \ldots, n_k = m$ such that $parent_t(n_i) = n_{i-1}$ for $1 < i \leq k$. Then we also say that n is an *ancestor* of m and m is a *descendant* of n. Note that those two terms are reflexive: every



(a) Tree t_0. **(b)** Twig query q_0. **(c)** Embeddings of q_0 in t_0.

Figure 1 Tree, twig query, and embeddings.

node is its own ancestor and descendant. We add the adjective *proper* to indicate that n and m are different nodes. The *depth* of a node is the length of the path from the root to the node. The *height* of a tree t, denoted $height(t)$, is the depth of its deepest leaf. The *size* of t, denoted $size(t)$, is the number of its nodes.

Queries. In general, a class of Boolean queries is a set $\mathcal{Q}$ with an implicitly given function $L : \mathcal{Q} \to 2^{Tree}$ that maps every query $q \in \mathcal{Q}$ to the set $L(q) \subseteq Tree$ of trees that satisfy q. The base class of queries, that we study in this paper, are (Boolean) twig queries, known also as *tree patterns* [2]. Basically, a twig query is an unranked tree that may additionally use a distinguished wildcard symbol $\star$ as a label and has two types of edges, child and proper descendant, corresponding to the standard XPath axes. Fig. 1b contains example of a twig queries: child edges are drawn with a single line and descendant edges with a double line.

Formally, a *twig query* q is a tuple $(N_q, root_q, parent_q, lab_q, edge_q)$, where N_q is a nonempty finite set of nodes, $root_q$ is the *root* node, $parent_q : N_q \setminus \{root_q\} \to N_q$ is a child-to-parent function, $lab_p : N_q \to \Sigma \cup \{\star\}$ is a labeling function, and $edge_q : N_q \setminus \{root_q\} \to \{child, desc\}$ is the function that indicates the type of the incoming edge of a non-root node. By *Twig* we denote the set of all twig queries. We adapt the standard notions defined for trees (leaf, path, etc.) to twig queries by ignoring the $edge_q$ component of the query.

Embeddings. We define the semantics of twig queries using the notion of an embedding which essentially maps nodes of the twig query to the nodes of the tree in a manner consistent with the semantics of the edges and the node labels. In the sequel, for two $x, y \in \Sigma \cup \{\star\}$ we say that x *matches* y if $y \neq \star$ implies $x = y$. Note that this relation is not symmetric: the label a matches $\star$ but $\star$ does not match a. Formally, an *embedding* of a twig query q in a tree t is a function $\lambda : N_q \to N_t$ such that:

1. $\lambda(root_q) = root_t$,
2. $lab_t(\lambda(n))$ matches $lab_q(n)$ for every node n of q,
3. $\lambda(n)$ is a proper descendant of $\lambda(parent_q(n))$ for every $n \in N_q \setminus \{root_q\}$,
4. $\lambda(n)$ is a child of $\lambda(parent_q(n))$ for every $n \in N_t \setminus \{root_q\}$ such that $edge_q(n) = child$.

Then, we say that t is *satisfies* q. Fig. 6 presents all embeddings of the query q_0 in tree t_0. The *language* of a query $q \in Twig$ is the set of all trees satisfying q

$$L(q) = \{t \in Tree \mid t \text{ satisfies } q\}.$$

The notion of an embedding extends in a natural fashion to a pair of queries $q, p \in Twig$: an *embedding* of q in p is a function $\lambda : N_q \to N_p$ that satisfies the conditions 1, 2, and 3 above (with t being replaced by p) and the following condition (which ensures that child edges are mapped to child edges only):

4′. $\lambda(n)$ is a child of $\lambda(parent_q(n))$ and $edge_p(\lambda(n)) = child$ for every $n \in N_q \setminus \{root_q\}$ such that $edge_q(n) = child$.

Then, we write $q \preccurlyeq p$. Because a tree can be seen as a twig query, we often abuse the notation and write $t \preccurlyeq q$ to indicate that there is an embedding of q in t.

Query containment and equivalence. Given two queries q and p, q is *contained* in p, in symbols $q \subseteq p$, iff $L(q) \subseteq L(p)$. We say that q and p are *equivalent*, denoted $q \equiv p$, if $L(q) = L(p)$. It is well known that for twig queries, the existence of an embedding implies containment but the converse does not hold in general [13]. There are also significant computational differences: the containment of twig queries is coNP-complete [17, 15] whereas testing the existence of an embedding is in PTIME.

3 Characterizing queries with examples

In this section we formally define characterizability of queries and show that unions of twig queries are not characterizable.

An *example* is an element of $Tree \times \{+, -\}$, a pair consisting of a tree and an indicator of whether the example is *positive* $(+)$ or *negative* $(-)$. Every query q defines the set of its examples $L^{\pm}(q)$:

$$L^{\pm}(q) = L(q) \times \{+\} \cup (Tree \setminus L(q)) \times \{-\}.$$

Given a set of examples S, we denote by $S_+ = \{t \in Tree \mid (t, +) \in S\}$ and by $S_- = \{t \in Tree \mid (t, -) \in S\}$ the sets of respectively positive and negative examples in S. A query q is *consistent* with examples S iff $S_+ \subseteq L(q)$ and $S_- \cap L(q) = \emptyset$ (or simply $S \subseteq L^{\pm}(q)$).

▶ **Definition 3.1.** A class of queries $\mathcal{Q}$ is *characterizable* iff for every query $q \in \mathcal{Q}$, there exists a finite set of examples $Char(q)$ characterizing q i.e., such that q is the only query in $\mathcal{Q}$ consistent with $Char(q)$ (modulo query equivalence).

Characterizability alone does not ensure any bound on the cardinality of the set of characterizing examples nor any bound on their size. As we show later on, twig queries are characterizable but the number of necessary examples may be exponential, which may be undesirable for practical purposes. Therefore, we formalize a variant of characterizability that ensures a more manageable size of the characterizing set of examples.

▶ **Definition 3.2.** A class of queries $\mathcal{Q}$ is *succinctly characterizable* iff there exists a polynomial $poly(x)$ such that for every query $q \in \mathcal{Q}$ there exists a set of examples $Char(q)$ characterizing q and such that its cardinality is bounded by $poly(size(q))$ and so is the size of every of its elements.

3.1 Non-characterizability of unions of twig queries

We now consider the class $UTwig$ consists of finite subsets of twig queries $Q = \{q_1, \ldots, q_k\} \subseteq Twig$ interpreted in the natural fashion: $L(Q) = \bigcup\{L(q) \mid q \in Q\}$. The *height* of a nonempty union of twig queries Q is the maximum height of a query in Q. Note that if the height of tree t is superior to the height of a twig query q, then q is not satisfied in t. Naturally, the same necessary condition hold for unions of twig queries.

We say that a query $Q \in UTwig$ is *unsaturated* if the set of its negative examples $Tree \setminus L(Q)$ is infinite and contains trees of arbitrary height. The universal query $\{\}$ is not unsaturated because it has no negative examples. One can, however, easily see that $UTwig$

contains unsaturated queries e.g., the singleton query $Q_0 = \{q_0\}$ with q_0 from Fig. 1b is unsaturated because any tree that whose root node does not have a b child is a negative example of Q_0.

Now, take any unsaturated query $Q \in \mathit{UTwig}$ and a set of examples S consistent with Q. Let $U_- = (\mathit{Tree} \setminus L(Q)) \setminus S_-$ be the set of all negative examples of Q that are not used in S. From U_- we pick any element t whose height is greater than the height of any negative example in S. We treat t as a twig query and construct $Q' = Q \cup \{t\}$. Clearly, Q' is consistent with S because all positive examples S_+ are satisfied by $Q \subseteq Q'$ and none of the negative examples satisfy the newly added query component t because the height of t is greater than the height of any of the negative examples. Also, Q and Q' are not equivalent because t satisfies Q' but not Q.

▶ **Theorem 3.3.** *Unions of twig queries are not characterizable.*

Finally, we point out that when applied with diligence the above procedure can be iterated infinitely thus generating an infinite sequence of queries consistent with the given set of examples.

4 Characterizability of Twig queries

In this section we show that Twig queries are characterizable but may require a number of examples exponential in the size of the query.

▶ **Proposition 4.1.** *Twig queries are characterizable.*

Proof. For a tree $t \in \mathit{Tree}$ we define the *Twig*-theory it generates $\mathit{Th}(t) = \{q \in \mathit{Twig} \mid t \in L(q)\}$, the set of all twig queries that are satisfied by t. First, we show that for any tree its *Twig*-theory is finite modulo query equivalence. We observe that the height of any query $q \in \mathit{Th}(t)$ is bound by the height of t and the number of different labels in q is also bounded by the number of different labels in t. Because we consider only non-equivalent members of $\mathit{Th}(t)$, no node of q has two identical subtrees rooted at any two children of the node. The two observations above allow us to inductively prove that the number of non-equivalent different queries in $\mathit{Th}(t)$ is indeed finite.

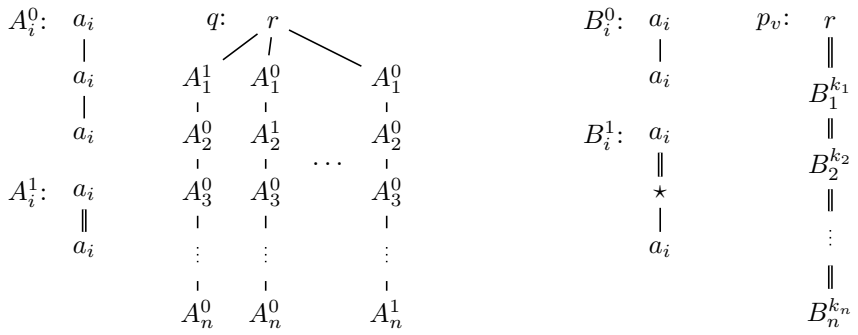
Next, for a given twig query q we outline the construction of a characterizing set of examples. For this, we construct a positive example $\mathtt{t}_0^q$ obtained from q by replacing every descendant edge by a child edge and using a fresh label a_0 (not used in q) to replace every $\star$. Note that $q \in \mathit{Th}(\mathtt{t}_0^q)$ and that for any $p \in \mathit{Th}(\mathtt{t}_0^q)$ that is not equivalent to q there exists a witness t of the non-equivalence of p and q, which we can use as a positive example, if t satisfies q but not p, or as a negative example, if t satisfies p but not q. Since $\mathit{Th}(\mathtt{t}_0^q)$ contains a finite number of queries modulo equivalence, only a finite number of examples is necessary to construct a set of examples characterizing q. ◀

Now, we show that twig queries may require exponential number of examples to characterize them.

▶ **Proposition 4.2.** *For any natural number n there exists a twig query q such that any set characterizing q contains at least 2^n examples.*

Proof. For a given natural number n we construct a query q and a set of twig queries U such that:

1. for each $p \in U$ we have $p \subseteq q$;



(a) Query q **(b)** Query p_v for $v = (k_1, \ldots, k_n)$.

Figure 2 Twig query q requiring exponentially many examples w.r.t. p_v.

2. for each $p \in U$ we have $q \not\subseteq p$; Moreover, no single positive example t can witness the fact that any two distinct $p_1, p_2 \in U$ are not equivalent to q;
3. U contains 2^n queries.

Hence any set of examples S characterizing q will have at least 2^n negative examples used to distinguish q from queries in U. We construct the query q over the set of labels $\{a_1, \ldots a_n, r\}$ as illustrated in Fig. 2a.

We also construct an auxiliary set of queries W consists of patterns p_v with v ranging over all $\{0,1\}$-vectors of length n, constructed as presented in Fig. 2b. We show that the set W satisfies the conditions 2 and 3 and later we use it to construct the set U that satisfies all conditions.

Clearly, for every vector v we have $q \not\subseteq p_v$ essentially because $A_i^1 \not\subseteq B_i^k$ for any $i \in \{1, \ldots, n\}$ and $k \in \{0,1\}$. Now, take two $\{0,1\}$-vectors $v = (k_1, \ldots, k_n)$ and $w = (k'_1, \ldots, k'_n)$ that differ at a position $j \in \{1, \ldots, n\}$ and w.l.o.g. assume that $k_j = 0$ and $k'_j = 1$. Suppose now that there exists a tree t that witnesses both the facts $q \not\subseteq p_v$ and $q \not\subseteq p_w$ i.e., t satisfies q but neither p_v and p_w. Let s_j be the j-th branch of q i.e., the branch using A_j^1. Since there is an embedding of q into t, take the path in which the branch s_j is embedded into

$$\pi = r \cdot a_1 \cdot a_1 \cdot a_1 \cdot \ldots a_{j-1} \cdot a_{j-1} \cdot a_{j-1} \cdot a_j \cdot \tau \cdot a_j \cdot a_{j+1} \cdot a_{j+1} \cdot a_{j+1} \cdot \ldots a_n \cdot a_n \cdot a_n,$$

where τ is a possibly empty path fragment. Note that if τ is empty, then $\pi \preccurlyeq p_v$, and thus, $t \preccurlyeq p_v$. However, if τ is not empty then $\pi \preccurlyeq p_w$, and thus, $t \preccurlyeq p_w$. This contradicts the assumption that t does not satisfy both p_v and p_w. Consequently, every query $p \in W$ requires a unique positive example to distinguish it from q.

Now, for two twig queries p and q by $p \cap q$ we denote the twig query obtained by joining p and q at the root node (which has the same label). The set of queries U is defined as $U = \{q \cap p \mid p \in W\}$. Because $L(p \cap q) = L(p) \cap L(q)$, every element of U is properly contained in q and every element of U still requires a separate example to distinguish it from q. ◀

5 Anchored twig queries

In this section, we present the class of anchored twig queries and argue that they constitute a subclass that is functionally very close to twig queries. We also present the construction

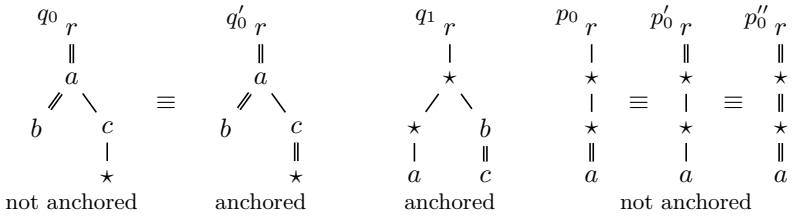


Figure 3 Anchored and non-anchored twig queries.

of representative documents for a given anchored twig query and recall the relationships between their structure and semantics with their important computational implications.

The class of anchored twig queries imposes restrictions on the mutual use of $\star$ and the descendant edges.

▶ **Definition 5.1.** A twig query is *anchored* if the following two conditions are satisfied:

1. A //-edge can be incident to a $\star$-node only if the node is a leaf.
2. A $\star$-node may be a leaf only if it is incident to a //-edge.

By *AnchTwig* we denote the set of all anchored twig queries.

A number of anchored and non-anchored queries is presented in Fig. 3. Note that the second condition requiring a $\star$ leaf to be incident to a descendant edge is merely technical: if the rule is violated by some $\star$ leaf, we can change its edge to a descendant edge and obtain an equivalent query (cf. $q_0 \equiv q_0'$ in Fig. 3).

In essence, anchored twig queries do not allow the descendant edges touch $\star$ except for leaves and thus cannot express conditions on ancestry of nodes with a minimal distance between them. For instance the query p_0 (Fig. 3) checks for the existence of a node labeled a at depth ≥ 2. We do not believe this restriction to be significant one from the practical point of view.

The reason we use anchored queries is the close relationship between the structure of the query and its semantics [19]: containment is equivalent to the existence of embedding. More precisely, for any $p, q \in \mathit{AnchTwig}$ we have $p \subseteq q$ iff $p \preccurlyeq q$. The same relationship does not hold for queries that are not anchored e.g., the non-anchored query q_0 and the anchored query q_0' in Fig. 3 are equivalent and in particular $q_0' \subseteq q_0$ but there is no embedding of q_0 into q_0'. Consequently, testing the containment is reduced to testing the existence of an embedding, and therefore, is in PTIME. This stands in contrast with coNP-completeness of the containment of arbitrary twigs [17, 15].

The main tool used in proving the equivalence of containment and embedding for anchored queries are *containment characterizing trees* [13, 19]. For an anchored query q of height k we construct two trees:

t_0^q is obtained from q by replacing every descendant edge by a child edge and every $\star$ with a fresh label a_0 that is not used by q.

t_1^q is obtained from q by replacing every $\star$ by a_1 and every descendant edge by a a_2-path of length k, a_1 and a_2 are two different fresh labels not used in q and different from a_0.

An example of the construction is presented in Fig. 4.

The instrumental result follows.

▶ **Lemma 5.2** ([19]). *Take any anchored query q and construct t_q^1 as described above. For any query p whose height is bounded by the height of q and that does not use the labels a_1 and a_2, $\mathsf{t}_1^q \preccurlyeq p$ implies $q \preccurlyeq p$.*

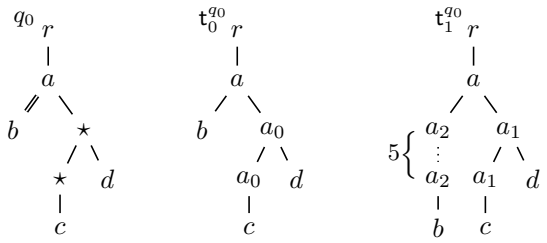


Figure 4 Construction of containment characterizing trees.

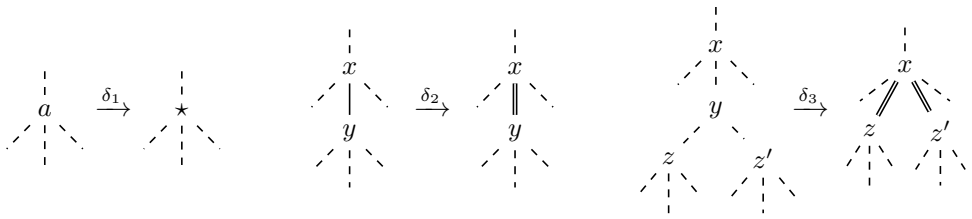


Figure 5 One-step generalization operations ($\rightarrow$).

The proof consists of an *anchoring* technique that normalizes the embedding of p into t_1^q and then translates it to an embedding of p into q. Note that if $\mathsf{t}_0^q \preccurlyeq p$, then the height of p is bounded by the height of q and p does not use a_1 and a_2. Therefore if both t_0^q and t_1^q satisfy p, then $q \subseteq p$.

The most important implication of Lemma 5.2 is that by using only two positive examples t_0^q and t_1^q we only need to care about the queries that properly contain q and provide negative examples to distinguish q from queries properly containing q. Although their number might be quite large, we focus only on the most specific ones:

$$\Phi(q) = \{q' \in \mathit{AnchTwig} \mid q \subsetneq q' \wedge \nexists q''.\ q \subsetneq q'' \subsetneq q'\}.$$

From the view of the semi-lattice of anchored twig queries, we wish to gain an understanding of its topology to characterize the (outbound) neighborhood of a query.

6 Generalization operations

In this section, we introduce a set of generalization operations and show their connection with an injective embeddings and how the operations allow us to navigate the semi-lattice of anchored twig queries under injective semantics. We use those findings to succinctly characterize anchored twig queries under the injective semantics.

We employ simple generalization operations that we first define on arbitrary twig queries and later on tailor them to anchored twig queries. There are 3 operations, presented in Fig. 5, where dashed lines indicate optional arbitrary edges, x, y and z may have arbitrary labels in $\Sigma \cup \{\star\}$ while a ranges over Σ:

δ_1 changes the label of a non-$\star$ node to $\star$;
δ_2 changes a child edge to a descendant edges;
δ_3 removes a node and connects all its children to the parent node with a descendant edge;

We say that q is *one-step generalization* of p, in symbols $p \rightarrow q$, iff q is obtained by performing one of the 3 generalization operations.

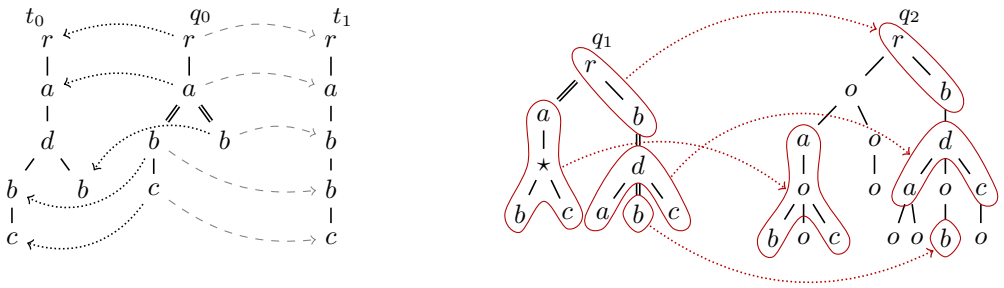


Figure 6 Injective embeddings $t_0 \preccurlyeq_\circ q_0$ and $q_2 \preccurlyeq_\circ q_1$.

6.1 Injective embeddings

There is a close connection between applying sequences of generalization operations and the existence of a special kind of injective embeddings. Formally, an embedding of λ of q into p is *injective* if it additionally satisfies the condition:

5. $\lambda(n_1)$ is an ancestor of $\lambda(n_2)$ in p if and only if n_1 is an ancestor of n_2 in q, for any two nodes n_1 and n_2 of q.

We write $p \preccurlyeq_\circ q$ if there is an injective embedding of q into p. We define analogously the injective embeddings of a twig query into a tree. Fig. 6 presents an example of an injective embedding of q_0 into t_0.

We point out that any injective embedding is an injective function but the converse does not necessarily holds (cf. q_0 and t_1 in Fig. 6). This however will not lead to confusion as we do not consider embeddings that are injective functions while violating condition 5. We define the *injective semantics* of twig queries as $L_\circ(q) = \{t \in \mathit{Tree} \mid t \preccurlyeq_\circ q\}$ and by $q \subseteq_\circ p$ denote $L_\circ(q) \subseteq L_\circ(p)$. The anchoring technique of Lemma 5.2 can be easily adapted to injective embeddings because injective embeddings are under closed composition.

▶ **Corollary 6.1.** *For any $p, q \in \mathit{AnchTwig}$, $p \subseteq_\circ q$ iff $p \preccurlyeq_\circ q$ iff $p \to^* q$.*

The connection between the generalization operations and injective embeddings is quite natural. While it is quite obvious that $p \to^* q$ implies $p \preccurlyeq_\circ q$, the converse is also true because the existence of an injective embedding of q into p ensures that all fragments of q match areas of p in a configuration that allows to easily identify the generalization operations required to transform p into q, cf. the injective embedding of q_1 into q_2 in Fig. 6. Hence,

▶ **Lemma 6.2.** *For arbitrary twig queries $p, q \in \mathit{Twig}$ we have $p \to^* q$ iff $p \preccurlyeq_\circ q$.*

Finally, we point out, however, alternative types of injective embeddings have been identified and used in the literature (cf. [12]), for our purposes any type of injective embeddings can be used. However, the set of necessary basic generalization operations (Fig. 5) depends on the chosen type of injective embedding, and we have chosen the type of injective embeddings known as *ancestor-preserving* embeddings because the generalization operations seems to be the most natural.

6.2 Generalizations of anchored twig queries

We wish to use the connection between generalization operations and the containment in order to map out the semi-lattice $\langle \mathit{AnchTwig}, \subseteq_\circ \rangle$ of anchored twig queries under the injective

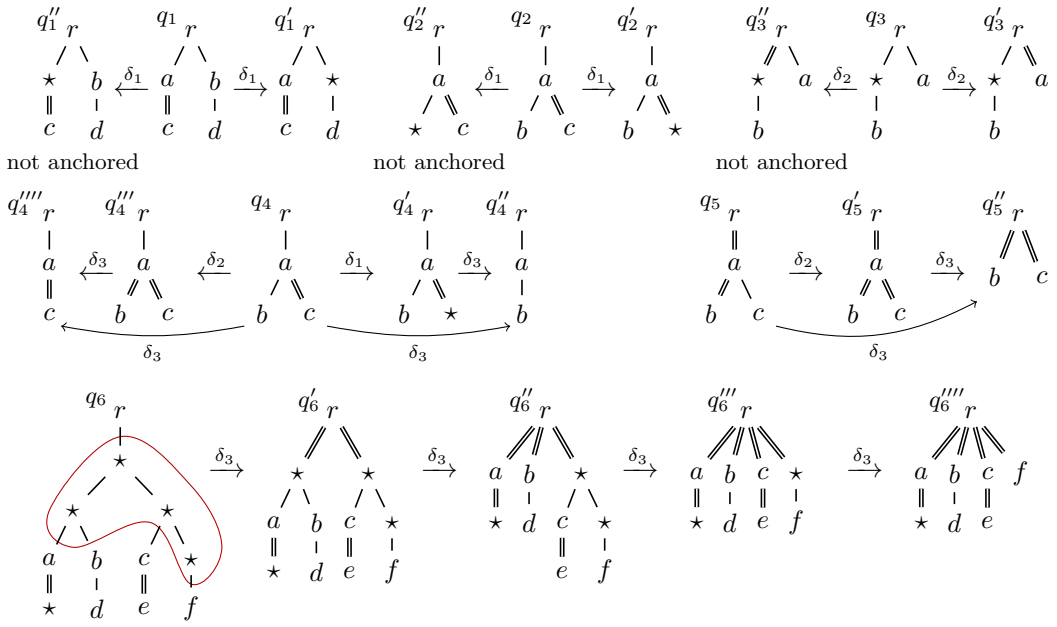


Figure 7 Applying generalization operations to anchored twig queries.

semantics. More precisely, for a given anchored query p we wish to know the set of anchored queries $\Phi_\circ(p)$ in the immediate (outgoing) neighborhood of p.

$$\Phi_\circ(q) = \{q' \in \mathit{AnchTwig} \mid q \subsetneq_\circ q' \wedge \nexists q'' \in \mathit{AnchTwig}.\ q \subsetneq_\circ q'' \subsetneq_\circ q'\}.$$

Lemma 6.2 encourages us to approach this challenge with the use of generalization operations tailored to anchored queries.

Given two anchored queries $p, q \in \mathit{AnchTwig}$, we say that q is an *immediate anchored generalization* of p, in symbols $p \lhd_\circ q$, iff $p \to^+ q$ and there is no $z \in \mathit{AnchTwig}$ such that $p \to^+ z$ and $z \to^+ q$. Essentially, the immediate anchored generalization relation defines the Hasse diagram of the semi-lattice of the anchored twig queries under injective semantics, and therefore, $\Phi_\circ(p) = \{p' \in \mathit{AnchTwig} \mid p \lhd_\circ p'\}$. We next show how $\lhd_\circ$ can be defined in terms of a small number of macros consisting of sequences of generalization operations.

To obtain an immediate anchored generalization of an anchored twig query we apply diligently the generalizations operations, making sure that: 1) the end result is an anchored twig query and 2) there is no intermediate anchored twig query that can be obtained on an alternative path. In particular, the first two generalization operations δ_1 and δ_2 can be used as long as they do not yield a query that is not anchored (cf. q_1, q_2, and q_3 in Fig. 7). The δ_3 is more involved. First of all, under certain conditions applying δ_3 is not authorized because an intermediate query can be reached with operations δ_1 or δ_2 (cf. q_4 and q_5 in Fig. 7). Furthermore, while possibly violating the structural constraints of anchored queries, δ_3 can be applied to a non-leaf $\star$ node provided that δ_3 is applied to all neighboring $\star$-nodes (cf. q_6 in Fig. 7).

We state formally the manner in which the generalizations should be used on anchored twig queries.

▶ **Lemma 6.3** ($\lhd_\circ$-operations). *For any anchored twig queries $p, q \in \mathit{AnchTwig}$ we have that $p \lhd_\circ q$ iff q is obtained from p by applying:*

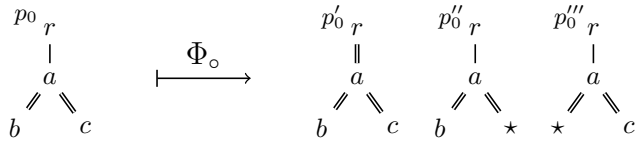


Figure 8 Immediate anchored generalizations.

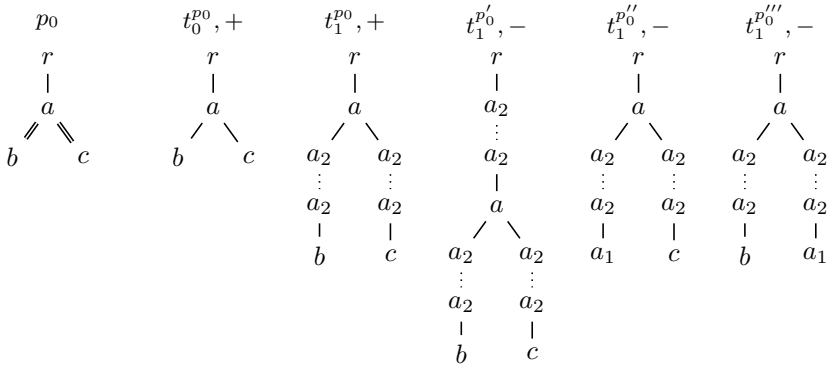


Figure 9 Characterizing query under the injective semantics.

1. δ_1 *to an inner node incident to child edges only;*
2. δ_1 *to a leaf node incident to a descendant edge;*
3. δ_2 *to an edge that is not incident to a* $\star$ *node;*
4. δ_3 *to a leaf node only if it is a* $\star$ *node or if its parent is a* $\star$ *node with other children;*
5. δ_3 *to a non-*$\star$ *node incident to descendant edges only;*
6. δ_3 *in a exhaustive manner to a connected area of* $\star$ *nodes.*

In the sequel we refer to the macros from Lemma 6.3 as $\lhd_\circ$-operations. We point out that from the point of view of $\lhd_\circ$-operations, any connected area of $\star$ nodes is seen as one particular node and we identify it with its top most $\star$ node. Note that the number of possible different applications of $\lhd_\circ$-operations to a query p is $O(size(p))$, and consequently, the $\Phi_\circ(p)$ contains a polynomial number of queries each of size bounded by the size of p. An example of construction of $\Phi_\circ$ is presented in Fig. 8.

6.3 Characterizability of AnchTwig under injective semantics

Because the number of possible immediate anchored generalizations of a query p is $O(size(p))$, we can characterize any anchored twig query p under injective semantics with the following set of examples (with all t_1^q's trees using a_2-chains of the same length as t_1^p):

$$Char_\circ(p) = \{(\mathsf{t}_0^p, +), (\mathsf{t}_1^p, +)\} \cup \bigcup\{(\mathsf{t}_1^q, -) \mid q \in \Phi_\circ(p)\}.$$

An example of construction of the characterizing set of examples is presented in Fig. 9 for the query p_0 from Fig. 8.

To show that $Char_\circ(p)$ does indeed characterizes p, take any query q consistent with $Char_\circ(p)$. Since q is satisfied by both t_0^p and t_1^p, $p \preccurlyeq_\circ q$ and if p would be properly contained by q, then q would satisfy one of the negative examples in $Char_\circ(p)$.

▶ **Theorem 6.4.** *Anchored twig queries are succinctly characterizable under injective semantics.*

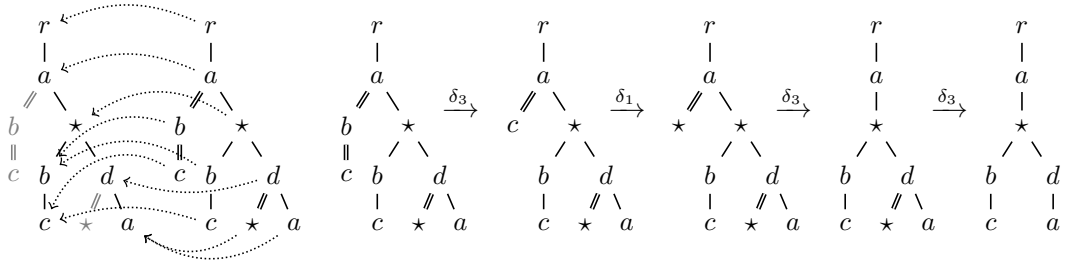


Figure 10 Reducing a query.

7 Characterizability of Anchored Twig queries

In this section, we extend the approach presented in the previous section to the standard semantics of twig queries. The main challenge is to handle possible overlaps of non-injective embeddings and we introduce a duplication operation whose controlled use ensures succinct characterizability. There is, however, a lesser challenge that we need to handle first, making sure that applying a $\lhd_\circ$-operation yields a more general query. Solving this challenge also shows that minimization of anchored twig queries is tractable and can be implemented using generalization operations which further illustrates the good behavior of the class of anchored twig queries.

7.1 Reducing anchored twig queries

If we are to base the solution of characterizability on $\lhd_\circ$-operations, we must be aware of the difference between the two semantics, how it can affect the use of $\lhd_\circ$-operations, and how to overcome the difficulties that arise.

The difference between the semantics can be illustrated on the example of query q_0 in Fig. 1b: under the injective semantics it ensures the existence of a node b with at least two different children, while such constraint cannot be expressed with the standard semantics (note that neither of the embeddings in Fig. 6 and is ancestor-preserving). In fact, the leaf $\star$ node in the query q_0 is redundant, can be removed, and a smaller equivalent query (under the standard semantics) is obtained.

The implications on use of $\lhd_\circ$-operations are as follows: if we take an anchored query p and any p' obtained by applying a $\lhd_\circ$-operation., then p needs not properly included in p' because p and p' may be equivalent under the standard semantics. To address this obstacle we *reduce* the query p: iteratively apply generalization operations (cf. Fig. 10), following $\lhd_\circ$ at each step, as long as the query remains equivalent to the original one. Note that a $\lhd_\circ$-operation may remove some nodes, rename nodes to $\star$, and change child edges to descendant edges. Essentially, it is a monotone process that finishes after $O(size(p))$ steps. An anchored twig query p is *reduced* if it is the end result of this procedure, or more precisely, if for no $p' \in \Phi_\circ(p)$ we have $p' \preccurlyeq p$.

Interestingly, not only are reduced queries suitable for use in $\Phi_\circ$ and can be obtained efficiently, but also they are the unique minimal canonical representatives of all equivalent queries.

Indeed, we show that an anchored twig query p is not minimal iff there exists an embedding of p into p other than the identity map, and furthermore it maps at least two nodes to the same target (an overlap). We can then carefully construct the image p' of this embedding, an anchored query whose set of nodes is the range of the embedding (see example in Fig. 10).

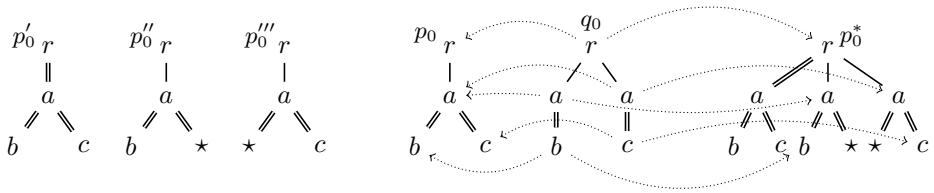


Figure 11 Immediate anchored generalizations versus an embedding with overlap $p_0 \preccurlyeq q_0$.

This query is smaller than p and the identity map is an injective embedding of p' into p, thus $p \lhd_\circ^* p'$. We point out that tractability of minimization of anchored twig queries is not a novel result. [11] presents a class of twig queries properly containing *AnchTwig* for which minimization is tractable. While the technique is similar to the presented above (but not identical), it can possibly be used as a more efficient alternative only because we shown that our reducing method also produces the minimal query.

▶ **Theorem 7.1.** *For every anchored twig query q there exists a unique reduced equivalent anchored twig query. Furthermore, this query is the size-minimal query equivalent to q and can be obtained in time polynomial in the size of q.*

7.2 Duplication operation

From now on, we use the standard semantics only. While using $\lhd_\circ$-operations on a reduced query p does construct a set of most specific queries properly containing p, it does not contain all such queries. Take for instance the query p_0 in Fig. 11 together with its immediate anchored generalizations $\Phi_\circ(p_0) = \{p_0', p_0'', p_0'''\}$. Now take the query q_0 (Fig. 11) that has an embedding into p_0, is not equivalent to p_0 (there is no embedding of p_0 into q_0), and note that it cannot be embedded into any of the immediate anchored generalizations of p_0.

This is because the embedding of q_0 into p_0 overlaps at two a nodes of q_0 whose subqueries have been obtained with applying different generalization operations.

We address the problem of the overlap with what could be seen as a adding a *duplication operation* to our system: this operation replaces a subquery rooted at a given node by a number of identical copies (including the same type of the edge to the parent). Such an operation would not, however, yield a query more general than the original query, but merely an equivalent query with redundancies, and therefore, not even reduced. Consequently, we investigate an augmented variant that applies the duplication operation and then generalizes every copy. In Fig. 11 the query p_0^* is a query obtained as a result of applying this operation to p_0 at node a. We point out, however, that the definition of this operation is not sufficiently precise: it is mutually dependent on the definition of generalization, which ideally should use the new operation. Also, unlike previously used operations which could only decrease the size of the input query, this operation has the potential to increase the size the query and when used recursively multiple time, the bounds on the size of the output query are not clear. We next settle these concerns with an appropriate recursive definition of query generalization.

We define generalizations of a query recursively on its subqueries. A subquery p of q at a node $n \in N_q \setminus \{root_q\}$ is essentially the query rooted at n but having additionally the incoming edge (from the parent) which can be altered by applying $\lhd_\circ$-operations to the root node of q. Naturally, we apply only those operations that are allowed in the context of the complete query q (thus as if knowing the label of the parent of the root node of p).

We fix a query q and let p be its subquery. A *minimal generalization* of p is any query obtained by either:

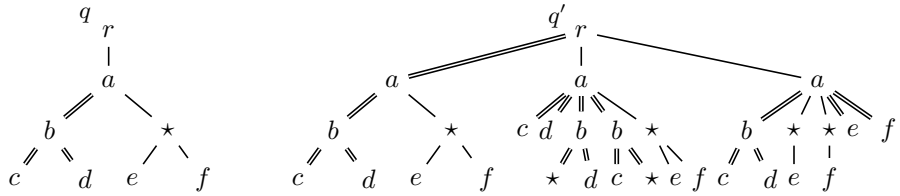


Figure 12 Constructing the minimal generalization q' of a query q ($q \triangleleft q'$).

1. applying a $\triangleleft_\circ$-operation at the root node of p (appropriately to the context of the root node in q)
2. replacing the subquery p' rooted at a child of the root node of p with the set of all *minimal generalizations* of p'.

The *minimal generalizations* of q are obtained by using only the second part of the definition (because we do not apply generalization operation to the root node). We write $q \triangleleft q'$ to indicate that q' is a minimal generalization of q. An example of constructing the minimal generalization of a query is presented in Fig. 12.

A reduced anchored twig query q has at most a linear number of minimal generalizations (equal to the number of children of the root node). It is not clear how big they can be given that duplication takes place. We prove a polynomial bound on the size of q' such that $q \triangleleft q'$. Let n be a node of q, with the path $n = n_0, n_1, \ldots, n_k = root_q$ from n to the root of q, and let ℓ_i be the number of children of n_i for $i \in \{1, \ldots, k\}$. We show with an inductive proof that q' contains $O(\ell_1 + \cdots + \ell_k)$ copies of the node n. Since $\ell_1 + \cdots + \ell_k$ is bounded by the number of nodes of q, each node is duplicated at most $size(q)$ times, and therefore, q' is of size $O(size(q)^2)$.

With an inductive proof on the structure of an arbitrary embedding between two queries that are not equivalent, we show that the minimal generalizations of a query q are the only outbound neighbors of q in the semi-lattice of anchored twig queries under the standard semantics.

▶ **Lemma 7.2.** *For any anchored twig query q, $\Phi(q) = \{q' \in AnchTwig \mid q \triangleleft q'\}$.*

Consequently, any anchored twig query q can be characterized with a polynomially-sized set of examples $Char(q) = \{(\mathtt{t}_0^q, +), (t_1^q, +)\} \cup \bigcup\{(\mathtt{t}_1^p, -) \mid p \in \Phi(q)\}$ and their sizes are polynomially bounded by the size of q.

▶ **Theorem 7.3.** *Anchored twig queries are succinctly characterizable.*

8 Conclusions and future work

In the present paper, we have identified and studied a novel problem of characterizing queries with examples. Our results have demonstrated that characterizability is a measure of richness of the query class, which is closely related to its expressive power. We have show that while union of twig queries are not characterizable, twigs alone are but may require exponential numbers of examples. Through the study of embeddings and generalization operations we have shown that the class of anchored twig queries is characterizable with a polynomially sized sets of examples.

Future work. We envision a number of possible future directions. We would like to extend our study of embeddings to other types of queries that employ this mechanism of defining

their semantics: conjunctive relational queries and regular path queries for graphs. Naturally, the goal would to be investigate the problem of characterizability of those database models. We also intend to explore the use of the proposed constructions of characterizing sets of examples in the context of grammatical inference [6, 7, 19].

References

1 A. Abouzied, D. Angluin, Ch. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *Proceedings of the 32Nd Symposium on Principles of Database Systems*, PODS '13, pages 49–60. ACM, 2013.

2 S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB Journal*, 11(4):315–331, 2002.

3 M. Anthony, G. Brightwell, D. Cohen, and J. Shawe-Taylor. On exact specification by examples. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 311–318, New York, NY, USA, 1992. ACM.

4 S. Cho, S. Amer-Yahia, L. V. S. Lakshmanan, and D. Srivastava. Optimizing the secure evaluation of twig queries. In *International Conference on Very Large Data Bases (VLDB)*, pages 490–501, 2002.

5 S. Cohen and Y. Y. Weiss. Certain and possible XPath answers. In *International Conference on Database Theory (ICDT)*, 2013.

6 C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

7 E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

8 E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302 – 320, 1978.

9 S. A. Goldman and M. J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20 – 31, 1995.

10 S. A. Goldman, R. L. Rivest, and R. E. Schapire. Learning binary relations and total orders. *SIAM J. Comput.*, 22(5):1006–1034, 1993.

11 B. Kimelfeld and Y. Sagiv. Revisiting redundancy and minimization in an xpath fragment. In *EDBT 2008, 11th International Conference on Extending Database Technology*, pages 61–72, 2008.

12 J. Michaliszyn, A. Muscholl, S. Staworko, P. Wieczorek, and Z. Wu. On injective embeddings of tree patterns. *CoRR*, abs/1204.4948, 2012.

13 G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.

14 F. Neven. Automata, logic, and XML. In *Workshop on Computer Science Logic (CSL)*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.

15 F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *International Conference on Database Theory (ICDT)*, pages 315–329. Springer-Verlag, 2003.

16 S. Salzberg, A. L. Delcher, D. G. Heath, and S. Kasif. Learning with a helpful teacher. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence.*, pages 705–711, 1991.

17 T. Schwentick. XPath query containment. *SIGMOD Record*, 33(1):101–109, 2004.

18 A. Shinohara and S. Miyano. Teachability in computational learning. *New Generation Comput.*, 8(4):337–347, 1991.

19 S. Staworko and P. Wieczorek. Learning twig and path queries. In *International Conference on Database Theory (ICDT)*, March 2012.

20 B. Ten Cate, V. Dalmau, and P. Kolaitis. Learning schema mappings. In *International Conference on Database Theory (ICDT)*, March 2012.

The Product Homomorphism Problem and Applications

Balder ten Cate[1] and Victor Dalmau[2]

1 **LogicBlox Inc. and UC Santa Cruz**
2 **Universitat Pompeu Fabra**

Abstract

The *product homomorphism problem* (PHP) takes as input a finite collection of structures $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and a structure $\mathbf{B}$, and asks if there is a homomorphism from the direct product $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_n$ to $\mathbf{B}$. We pinpoint the computational complexity of this problem. Our motivation stems from the fact that PHP naturally arises in different areas of database theory. In particular, it is equivalent to the problem of determining whether a relation is definable by a conjunctive query, and the existence of a schema mapping that fits a given collection of positive and negative data examples. We apply our results to obtain complexity bounds for these problems.

1998 ACM Subject Classification H.2 Database Management, G.2 Discrete Mathematics

Keywords and phrases Homomorphisms, Direct Product, Data Examples, Definability, Conjunctive Queries, Schema Mappings

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.161

1 Introduction

Structure identification [7] refers to the general problem of finding a structural description of some data. When the data is relational, this task is usually formalized as the problem of determining whether a given relation can be represented in a given logical formalism. The CQ-definability problem (also called existential inverse satisfiability problem in [6] and PP-definability problem in [13]) is one of the most studied variants of this problem (see [5, 6, 10, 13]). The input of the CQ-definability problem is a relation S and a finite set $R_1, \ldots, R_m$ of relations over the same domain than S, and the question is to decide whether S is expressible by a conjunctive query over the instance I that consists of $R_1, \ldots, R_m$. This question is not only relevant in the context of databases, but is also pertinent to constraint satisfaction. From a constraint satisfaction perspective, CQ-definability can be viewed as asking whether relation S can be expressed as the projection of the set of solutions of a constraint satisfaction instance that uses relations from $R_1, \ldots, R_m$ in its constraints [5] or, equivalently, whether S belongs to the *expressive power* of $R_1, \ldots, R_n$ [10]. In constraint satisfaction, the notion of expressive power plays an important role in the so-called algebraic approach in the study of the constraint satisfaction problem (see [10]). Another collection of problems that can be viewed as instances of structure identification is the *fitting problem for schema mappings*, where the input is a finite collection of data examples (I, J), where I and J are database instances over a source schema and a target schema, respectively, and the question is whether there exists a schema mapping that fits these data examples.

It turns out that the problems described above are closely related to a certain algorithmic problem from graph theory: given a collection of graphs, or more generally, relational structures $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$, is there a homomorphism from the direct product $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_n$ to $\mathbf{B}$. This problem, which we will refer to as the *product homomorphism problem* (PHP), is

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 161–176

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

clearly decidable in NExpTime: it suffices to materialize the (exponentially large) direct product $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_n$ and guess a homomorphism from it to $\mathbf{B}$. It was recently shown [13] that PHP is NExpTime-complete if the arity of the relations in the schema is unbounded. In this paper, we provide a simplified proof, and we establish that the same lower bound holds under various restrictions, including for a fixed schema. We use this to establish tight complexity bounds for CQ-definability as well as for various variants of the fitting problem for schema mappings.

After reviewing basic definitions in Section 2, we study the product homomorphism problem in Section 3. We then proceed with applications to instance-level query definability (Section 4) and to fitting problems for schema mappings (Section 5). We conclude in Section 6.

2 Preliminaries

Schemas, Structures, Database Instances, and Homomorphisms

A *schema* is a nonempty finite set of relation symbols τ of specified arities. A (finite) *structure* $\mathbf{A}$ of the schema τ (or, τ-structure), consists of a finite set A, called the *domain* of $\mathbf{A}$ and a relation $R^{\mathbf{A}} \subseteq A^r$ for every relation symbol $R \in \tau$ where r is the arity of R. Throughout the paper we will use the same boldface and slanted capital letters to denote a structure and its domain respectively.

Let $\mathbf{A}$ and $\mathbf{B}$ be structures of the same schema τ. A *homomorphism* h from $\mathbf{A}$ to $\mathbf{B}$, denoted $h : \mathbf{A} \to \mathbf{B}$, is a function from A to B, such that for every $R \in \mathbf{R}$ and every tuple $a = (a_1, \ldots, a_r) \in R^{\mathbf{A}}$, we have that tuple $h(a)$, defined as $(h(a_1), \ldots, h(a_r))$, belongs to $R^{\mathbf{B}}$. We shall write $\mathbf{A} \to \mathbf{B}$ to indicate that there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. When $\mathbf{A} \to \mathbf{B}$ and $\mathbf{B} \to \mathbf{A}$, then we say that $\mathbf{A}$ and $\mathbf{B}$ are *homomorphically equivalent*.

In Section 4 and Section 5 we will consider applications involving *database instances*. Recall that a database instance is a finite structure with an unspecified domain. In other words, the specification of a database instance includes the relations but does not include the domain. For the purposes of this paper, the difference between finite structures and database instances is inessential. This is because all notions and results in this paper are invariant for homomorphic equivalence, and modulo homomorphic equivalence, the domain of a structure can always be assumed to coincide with the *active domain*, that is, the set of all elements occurring in the relations of the structure. We will therefore freely switch from speaking about structures to speaking about instances in Section 4 and Section 5.

Direct products

An n-ary tuple a on A is any element of A^n (for $n \geq 1$). For every $1 \leq i \leq n$ we shall use $a[i]$ to denote its ith component of a. Let $R \subseteq A^n$ be a n-ary relation on A. Then the *projection*, $\mathrm{pr}_{i_1,\ldots,i_j} R$ over coordinates $i_1, \ldots, i_j \in \{1, \ldots, n\}$ is the j-ary relation defined as $\{(a[i_1], \ldots, a[i_j]) \mid a \in R\}$.

Let $a = (a_1, \ldots, a_n) \in A^n$ and $b = (b_1, \ldots, b_n) \in B^n$ be tuples of the same arity. The *direct product* $a \otimes b$ is the n-ary tuple on $A \otimes B$ defined as $((a_1, b_1), \ldots, (a_n, b_n))$. Similarly, if $R \subseteq A^n$ and $S \subseteq B^n$ are n-ary relations then the direct product, $R \otimes S$ is defined to be $\{a \otimes b \mid a \in R, b \in S\}$. The direct product, $\mathbf{A} \otimes \mathbf{B}$, of $\mathbf{A}$ and $\mathbf{B}$ is the τ-structure with domain $A \otimes B$ such that $R^{\mathbf{A}\otimes\mathbf{B}} = R^{\mathbf{A}} \otimes R^{\mathbf{B}}$ for every $R \in \tau$. We shall use $\Pi_{1 \leq i \leq n} R_i$ as a shorthand of $R_1 \otimes \cdots \otimes R_n$ (note that the $\otimes$ operation is associative up to isomorphism). Furthermore, we shall use R^n to denote $\Pi_{1 \leq i \leq n} R$.

The direct product construction is of fundamental importance in the study of graphs and homomorphisms, as it turns out to capture the meet (or, least upper bound) operation of the lattice of structures ordered by homomorphic embedding. That is, for all structures $\mathbf{B}_1, \ldots, \mathbf{B}_n$, we have (i) $\Pi_{1 \leq i \leq n} \mathbf{B}_i \to \mathbf{B}_i$ for all $1 \leq i \leq n$, and (ii) for all structures $\mathbf{A}$, if $\mathbf{A} \to \mathbf{B}_i$ for all $1 \leq i \leq n$, then $\mathbf{A} \to \Pi_{1 \leq i \leq n} \mathbf{B}_i$.

It is important to distinguish direct products from the construction that is usually referred to as *cartesian product* is database theory. Let $R \subseteq A^r$ and $S \subseteq B^s$ be relations of possibly different arity. The *cartesian product* of R and S is the $r+s$-ary relation on $A \cup B$ defined as

$$R \times S = \{(a_1, \ldots, a_r, b_1, \ldots, b_s) \mid (a_1, \ldots, a_r) \in R, (b_1, \ldots, b_s) \in S\}.$$

Conjunctive queries

An n-ary *conjunctive query* q (for $n \geq 0$) is specified by a first-order formula of the form $\phi(x_1, \ldots, x_n) = \exists y_1, \ldots, y_m(\alpha_1 \wedge \cdots \wedge \alpha_k)$, with $m \geq 0$ and $k \geq 1$, where $\alpha_1, \ldots, \alpha_k$ are relational atomic formulas (i.e., atomic formulas not involving equality), and such that each variable x_i occurs in at least one atom α_j. We denote by $q(\mathbf{A})$ the n-ary relation over A defined by $q(\mathbf{A}) = \{a \in A^n \mid \mathbf{A} \models \phi(a[1], \ldots, a[n])\}$. For simplicity, we assume that the atoms in a conjunctive query do not contain any constants (although our results, suitably adapted, can be shown to apply to queries with constants as well). A fundamental property of conjunctive queries is that they are preserved by homomorphisms: if $h : \mathbf{A} \to \mathbf{B}$ is a homomorphism and $a \in q(\mathbf{A})$, then $(h(a[1]), \ldots, h(a[n])) \in q(\mathbf{B})$.

3 The Product Homomorphism Problem

The *product homomorphism problem* (PHP) takes as input a finite collection of relational structures $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and another relational structure $\mathbf{B}$, all over the same schema, and asks whether there is a homomorphism from the direct product $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_n$ to $\mathbf{B}$. This problem is clearly solvable in non-deterministic exponential time. It follows from results in [13] that the problem is NExpTime-complete. The proof, based on a reduction from an exponential tiling problem, uses structures of bounded domain size but relations of unbounded arity. We provide a self-contained proof of NExpTime-hardness of PHP, and we show that it holds already for directed graphs, as well as for structures of bounded arity with a bounded domain size (but without a bound on the number of relations). More precisely, we obtain:

▶ **Theorem 1.** *The PHP is* NExpTime*-complete [13]. The lower bound holds already for*

1. *structures with binary relations and a bounded domain size;*
2. *structures with a single relation and a bounded domain size;*
3. *structures with a single binary relation*

This completes the picture, since PHP is solvable in polynomial time when all three of the above parameters (i.e., number of relations, arity, and domain size) are bounded, as follows from the fact that, in this case, there are, up to isomorphism, only a bounded number of possible structures that can be part of the input.

Theorem 1(1) is proved by an adaptation of the technique used in [13]. Theorem 1(2) is proved by a reduction from 1(1). Theorem 1(3) is proved by a reduction from 1(2).

3.1 Proof of Theorem 1(1)

Proof. The proof will be a reduction from the exponential tiling problem which we now define. A *domino system* is a finite structure $\mathbf{D} = (D, H^{\mathbf{D}}, V^{\mathbf{D}})$ where the elements in its

domain, D, are called *tile types* and $H^{\mathbf{D}}, V^{\mathbf{D}}$ are binary relations on D, called the horizontal and vertical adjacency relation, respectively. For $N > 1$, we use $[N]$ to denote the set $\{0, \ldots, N-1\}$. A *tiling* of $[N] \otimes [N]$ by $\mathbf{D}$ is any mapping $\rho : [N] \otimes [N] \to D$ satisfying the following conditions:

- $(\rho(i,j), \rho(i,j+1)) \in H^{\mathbf{D}}$ for every $i \in [N]$ and $j \in [N-1]$
- $(\rho(i,j), \rho(i+1,j)) \in V^{\mathbf{D}}$ for every $i \in [N-1]$ and $j \in [N]$

The input of an exponential tiling problem is constituted by a domino system $\mathbf{D}$, an integer $N > 1$ (written in binary), and a sequence $T_0, \ldots, T_{n-1} \in D$, with $n \leq N$. The question is whether there is a tiling ρ of $[N] \otimes [N]$ such that $\rho(0, j) = T_j$ for every $j \in [n]$.

It is shown in [4] that the exponential tiling problem is NExpTime-hard. As discussed in [13], there exists a single domino system $\mathbf{D}$ such that the problem remains coNExpTime-hard even if the domino system in the input is required to be $\mathbf{D}$ and $N = 2^m$ for some $m > 1$.

Let us introduce a bit of notation. For every $k \in [N]$, let us denote by $\mathbf{b}_k \in \{0,1\}^m$ the m-bit binary representation of k, which we assume starts with the bit of highest weight. Also, we shall use $\Pi_\ell \mathbf{A}_\ell$ as a shorthand for $\Pi_{0 \leq \ell \leq 2m} \mathbf{A}_\ell$

Given an instance $(\mathbf{D}, N, T_0, \ldots, T_{n-1}), N = 2^m$ of the exponential tiling problem we construct in polynomial time an instance $\mathbf{A}_1, \ldots, \mathbf{A}_{2m}$, $\mathbf{B}$, of the PHP. Each one of the structures, $\mathbf{A}_1, \ldots, \mathbf{A}_{2m}$, has domain $\{0,1\}$. In this way, there is a correspondence between $[N] \otimes [N]$ and the domain of $\Pi_\ell \mathbf{A}_\ell$. More precisely, we associate to every element $(i,j) \in [N] \otimes [N]$ the tuple $(x_1, \ldots, x_{2m}) \in \{0,1\}^{2m}$ where $(x_1, \ldots, x_m) = \mathbf{b}_i$ and $(x_{m+1}, \ldots, x_{2m}) = \mathbf{b}_j$.

The domain of $\mathbf{B}$ will be the set, D, of tile types, so that a map h from $\Pi_\ell A_\ell$ to B can be viewed as a way of assigning a tile type to each position on the $[N] \otimes [N]$ grid. Furthermore, by endowing the structures involved with suitable relations, we will ensure that every such mapping h is an homomorphism from $\Pi_\ell \mathbf{A}_\ell$ to $\mathbf{B}$ if and only if it corresponds to a valid tiling.

For ease of exposition, we will also allow unary relations in the instance of the PHP. It is straightforward to replace the unary relations by binary ones.

Let $\mathcal{H}, \mathcal{V}$ be binary relations on the domain $\{0,1\}^{2m}$ that denote the horizontal and vertical successor relations on $[N] \otimes [N]$. Formally

$$\mathcal{H} = \{(\mathbf{b}_i \mathbf{b}_j, \mathbf{b}_i \mathbf{b}_{j+1}) \mid i \in [N], j \in [N-1]\}$$

$$\mathcal{V} = \{(\mathbf{b}_i \mathbf{b}_j, \mathbf{b}_{i+1} \mathbf{b}_j) \mid i \in [N-1], j \in [N]\}$$

where $\mathbf{b}_i \mathbf{b}_j \in \{0,1\}^{2m}$ denotes the $2m$-ary bit string obtained concatenating $\mathbf{b}_i$ and $\mathbf{b}_j$. Also, let $\mathcal{P}_0 = \{\mathbf{b}_0 \mathbf{b}_0\}, \ldots, \mathcal{P}_{n-1} = \{\mathbf{b}_0 \mathbf{b}_{n-1}\}$ be unary singleton relations on the domain $\{0,1\}^{2m}$ that denote the first n cells in the zero row of $[N] \otimes [N]$.

In order to make our reduction work, we need to somehow make sure that the relations $\mathcal{H}, \mathcal{V}, \mathcal{P}_0, \ldots, \mathcal{P}_{n-1}$ are "available" in the product structure $\Pi_\ell \mathbf{A}_\ell$, by choosing the relations in structures $\mathbf{A}_1, \ldots, \mathbf{A}_{2m}$ appropriately.

Let us say that an r-ary relation R over domain $\{0,1\}^{2m}$ is *factorizable* if it can be represented as a direct product $R_1 \otimes \cdots \otimes R_{2m}$ where each R_ℓ is an r-ary relation over $\{0,1\}$.

Intuitively, this means that if we include in each structure $\mathbf{A}_\ell$ the relation R_ℓ, then the product structure $\Pi_\ell \mathbf{A}_\ell$ will contain the relation R. Each of the unary relations $\mathcal{P}_0, \ldots, \mathcal{P}_n$, being a singleton, is trivially factorizable. Indeed, for every $j \in [n]$, $\mathcal{P}_j = \{0\}^m \otimes \{\mathbf{b}_j[1]\} \otimes \cdots \otimes \{\mathbf{b}_j[m]\}$.

The binary relation $\mathcal{H}$ is *not* factorizable. However, it turns out to be a union of a small number of factorizable relations, which will suffice for our purposes. For each $k \in [m]$, let $\mathcal{H}_k = \mathcal{H} \cap (\{0,1\}^{2m-k-1} \otimes \{0\} \otimes \{1\}^k \otimes \{0,1\}^{2m})$. In words, $\mathcal{H}_k$ is the subrelation of $\mathcal{H}$ that contains all those $(\mathbf{b}_i \mathbf{b}_j, \mathbf{b}_i \mathbf{b}_{j+1}) \in \mathcal{H}$ such that $\mathbf{b}_j$ finishes with a zero followed by k

ones. Note that for every $(\mathbf{b}_i\mathbf{b}_j, \mathbf{b}_i\mathbf{b}_{j+1}) \in \mathcal{H}$, $\mathbf{b}_j$ must contain one zero. It follows that $\mathcal{H} = \bigcup_{k\in[m]} \mathcal{H}_k$. Furthermore, it is easy to see that $\mathcal{H}_k$ factorizes as

$$\mathcal{H}_k = \mathsf{id}^{2m-k-1} \otimes \{(0,1)\} \otimes \{(1,0)\}^k$$

where $\mathsf{id} = \{(0,0),(1,1)\}$ is the equality relation on $\{0,1\}$.

Similarly, we can express $\mathcal{V}$ as $\bigcup_{k\in[m]} \mathcal{V}_k$ where

$$\mathcal{V}_k = \mathsf{id}^{m-k-1} \otimes \{(0,1)\} \otimes \{(1,0)\}^k \otimes \mathsf{id}^m .$$

We are now ready to define the structures $\mathbf{A}_1, \ldots, \mathbf{A}_{2m}$ and $\mathbf{B}$. The scheme consists of the relations $H_0, \ldots, H_{m-1}, V_0, \ldots, V_{m-1}, P_0, \ldots P_{n-1}$. As mentioned above, the domain of $\mathbf{A}_\ell$ is $\{0,1\}$ for every $1 \leq \ell \leq 2m$. For $k \in [n], 1 \leq \ell \leq 2m$, we define

$$P_k^{\mathbf{A}_\ell} = \begin{cases} \{\mathbf{b}_k[\ell - m]\} & \text{if } m < \ell \\ \{0\} & \text{otherwise} \end{cases}$$

For $k \in [m], 1 \leq \ell \leq 2m$, we define

$$H_k^{\mathbf{A}_\ell} = \begin{cases} \{(1,0)\} & 2m - k < \ell \\ \{(0,1)\} & \ell = 2m - k \\ \mathsf{id} & \text{otherwise} \end{cases}$$

$$V_k^{\mathbf{A}_\ell} = \begin{cases} \{(1,0)\} & m - k < \ell \leq m \\ \{(0,1)\} & \ell = m - k \\ \mathsf{id} & \text{otherwise} \end{cases}$$

The domain of structure $\mathbf{B}$ is the set, D, of all tile types. For $k \in [n]$ we define $P_k^{\mathbf{B}} = \{T_k\}$, and for $k \in [m]$, we define $H_k^{\mathbf{B}} = H^{\mathbf{D}}$, and $V_k^{\mathbf{B}} = V^{\mathbf{D}}$.

It follows from the definitions that $P_k^{\Pi_\ell \mathbf{A}_\ell} = \mathcal{P}_k$ for all $k \in [n]$ and that $H_k^{\Pi_\ell \mathbf{A}_\ell} = \mathcal{H}_k$ and $V_k^{\Pi_\ell \mathbf{A}_\ell} = \mathcal{V}_k$ for every $k \in [m]$. It follows that there is a homomorphism from $\Pi_\ell \mathbf{A}_\ell$ to $\mathbf{B}$ if and only if there is a valid tiling. The reduction we have just defined can be easily carried out in polynomial time. ◀

3.2 Proof of Theorem 1(2)

Proof. The proof proceeds by a reduction from the PHP with bounded domain size (Theorem 1(1)). We shall show how to construct, given an instance $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$ of the PHP another one with only one relation in each structure. Furthermore, the reduction will only increase the domain size of each structure by one.

Let $R_1, \ldots, R_k$ be the relation symbols in the scheme. The most immediate way to do the reduction consists in to replace, in every structure $\mathbf{C}$ among $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$, relations $R_1^{\mathbf{C}}, \ldots, R_k^{\mathbf{C}}$ by a single relation $R_1^{\mathbf{C}} \times \cdots \times R_k^{\mathbf{C}}$. It is not difficult to see that a mapping $h : \Pi_i A_i \to B$ is an homomorphism in the original instance if and only is an homomorphism in the instance obtained after the transformation. However, this transformation cannot be carried out in polynomial time as computing the cartesian product of k relations requires time exponential on k. To overcome this difficulty we shall apply first an step which will reduce the number of relations to 2.

We can assume that all relation symbols $R_1, \ldots, R_k$ have arity 2. This assumption is not essential but simplifies slightly the presentation. Let $\mathbf{C}$ be any structure among $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$ and let 0 be a fresh element not occurring in $B \cup \bigcup_{1\leq i\leq n} A_i$. We denote by $\mathbf{C}^*$ the structure with domain $C \cup \{0\}$ with the following relations:

(i) a unary relation $P^{\mathbf{C}^*} = C$
(ii) a $2k$-ary relation $R^{\mathbf{C}^*}$ defined as

$$R^{\mathbf{C}^*} = \{0^{2k}\} \cup \bigcup_{1 \leq j \leq k} \{0^{2(j-1)}\} \times R_j^{\mathbf{C}} \times \{0^{2(k-j)}\}\,.$$

That is, $R^{\mathbf{C}^*}$ contains all-zeroes tuple $(0, \cdots, 0)$, and, for every $1 \leq j \leq k$ and every tuple $(a, b) \in R_j$ $(1 \leq j \leq k)$, the $2k$-ary tuple $(a_1, \ldots, a_{2k})$ where all coordinates are 0 with the exception of a_{2j-1} and a_{2j} which are a and b respectively.

This transformation can be carried out in polynomial time and it increases the domain of each structure with at most one element. We claim that $\Pi_i \mathbf{A}_i^* \to \mathbf{B}^*$ if and only if $\Pi_i \mathbf{A}_i \to \mathbf{B}$.

In one direction, suppose h^* is an homomorphism from $\Pi_i \mathbf{A}_i^*$ to $\mathbf{B}^*$. It follows that h^* must map every element of $\Pi_i A_i = P^{\Pi_i \mathbf{A}_i^*}$ to an element of $B = P^{\mathbf{B}^*}$. It is then easy to see that the restriction h of h^* with domain $\Pi_i A_i$ is in fact a homomorphism from $\Pi_i \mathbf{A}_i$ to $\mathbf{B}$. Indeed, let $1 \leq j \leq k$ and let $(a, b) \in R_j^{\Pi_i \mathbf{A}_i}$. It follows by the definition of R, that $\mathrm{pr}_{2j-1,2j}\, R^{\mathbf{C}^*} = R_j^{\mathbf{C}} \cup \{(0,0)\}$ for every $\mathbf{C}$ among $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$. Consequently we have that $(a, b) \in \mathrm{pr}_{2j-1,2j}\, R^{\Pi_i \mathbf{A}*_i}$. It follows that h maps (a, b) to a tuple in $R_j^{\mathbf{B}} \cup \{(0,0)\}$. Since this tuple must be in B^2 we are done.

Conversely, suppose h is an homomorphism from $\Pi_i \mathbf{A}_i$ to $\mathbf{B}$. Let h^* be the map from $\Pi_i A_i^*$ to B^* that extends h such that every element in $\Pi_i A_i^*$ containing a 0 is sent to the element 0 of B^*. Formally, $h^*(a)$ is defined to be $h(a)$ whenever $a \in \Pi_i A_i$ and 0 otherwise. We shall see that h^* is an homomorphism from $\Pi_i \mathbf{A}_i^*$ to $\mathbf{B}^*$. Let a be any element in $P^{\Pi_i \mathbf{A}_i^*}$. It follows from the definition of relation P that $a \in \Pi_i A_i$. Then, $h^*(a)$ is $h(a)$ which necessarily belongs to $B = P^{\mathbf{B}^*}$. Now, let $a = (a_1, \ldots, a_{2k})$ be any tuple in $R^{\Pi_i \mathbf{A}_i^*}$ and let J be the set containing all coordinates $j \in \{1, \ldots, 2k\}$ such that $a_j \in \Pi_i A_i$. It follows from the definition of R that J is empty or is of the form $\{a_{2j-1}, a_{2j}\}$ for some $j \in \{1, \ldots, k\}$. If $J = \emptyset$ then $h^*(a) = (0, \ldots, 0) \in R^{\mathbf{B}^*}$ (h is applied component-wise). Now assume that $J = \{a_{2j-1}, a_{2j}\}$. Note that $h^*(a_i)$ is $h(a_i)$ if $i \in \{2j-1, 2j\}$ and 0 otherwise. We have that $(a_{2j-1}, a_{2j}) \in R_j^{\Pi_i \mathbf{A}}$ and hence $(h(a_{2j-1}), h(a_{2j})) \in R_\ell^{\mathbf{B}}$. Consequently, $h^*(a) \in R^{\mathbf{B}}$. ◀

3.3 Proof of Theorem 1(3)

Proof. We shall give a reduction from the PHP with a single relation (Theorem 1(2)). Let $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$ be an instance of the PHP over a scheme containing a single r-ary relation R. We may assume without loss of generality that, for each structure $\mathbf{C} = (C, R^{\mathbf{C}})$ among $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$, the projection of $R^{\mathbf{C}}$ to the first coordinate is the entire domain C. This is because we can always replace the r-ary relation R by the $(r+1)$-ary relation $C \times R$. This transformation can be carried out in polynomial time and it does not affect the existence or non-existence of a homomorphism from $\Pi_i \mathbf{A}_i$ to $\mathbf{B}$.

For every $i \in \{1, \ldots, n\}$, we denote by $G(\mathbf{A}_i)$ the digraph defined as follows. The nodes of $G(\mathbf{A}_i)$ include all elements in A_i. Furthermore, for every tuple $t \in R^{\mathbf{A}_i}$, $G(\mathbf{A}_i)$ contains r additional nodes, which we denote $t^1, \ldots, t^r$. Furthermore, we include the following edges:

- (t^j, t^{j+1}) for every $1 \leq j < r$.
- $(t[j], t^j)$ for every $1 \leq j \leq r$.

We define $G(\mathbf{B})$ as the digraph obtained from $\mathbf{B}$ in the same way, except that we further add $r-1$ additional elements $s^1, \ldots, s^{r-1}$ called *sink nodes*. We also add edge (s^j, s^{j+1}) for every $1 \leq j < r-1$. Furthermore we add an edge from every element in B to every sink node.

Claim: There is a homomorphism $h' : \Pi_i G(\mathbf{A}_i) \to G(\mathbf{B})$ if and only if there is a homomorphism $h : \Pi_i \mathbf{A}_i \to \mathbf{B}$.

In the remainder, we prove this claim, which immediately implies the theorem. We start with the more difficult direction. Let h be a homomorphism from $\Pi_i \mathbf{A}_i$ to $\mathbf{B}$. We shall define from h a homomorphism h' from $\Pi_i G(\mathbf{A}_i)$ to $G(\mathbf{B})$. Let $v = (v_1, \ldots, v_n)$ be a node of $\Pi_i G(\mathbf{A}_i)$.

- If $v_i \in A_i$ for every $1 \leq i \leq n$ then we say that v is of "type 1". In this case we define $h'(v) = h(v)$.
- If, for every $1 \leq i \leq n$, $v_i = t_i^{j_i}$ where $t_i \in R^{\mathbf{A}_i}$ and $j_i \in \{1, \ldots, r\}$ then:
  - If, in addition, there exists some j such that $j_i = j$ for every $1 \leq i \leq n$ then we say that v is of "type 2". Note that $\Pi_i t_i$ is a tuple in $R^{P i_i \mathbf{A}_i}$ and hence $h(\Pi_i t_i)$ (where h is applied component-wise) is a tuple in $R^{\mathbf{B}}$. In this case, define $h'(v)$ to be $h(\Pi_i t_i)^j$.
  - Otherwise we say that v is of "type 3" and we set $h(v')$ to the sink node s^j where $j = \min\{j_i \mid 1 \leq i \leq n\}$. Observe that, in this case, necessarily $j \leq r - 1$.
- If v is not of any of the previous types then we say that is of "type 4". In this case, we shall prove there exists a vertex u of type 1 such that for every vertex w of type 2 the following holds:

$$(v, w) \text{ is an edge of } \Pi_i G(\mathbf{A}_i) \Rightarrow (u, w) \text{ is an edge of } \Pi_i G(\mathbf{A}_i)\,.$$

  In this case we set $h'(v) = h'(u)$. Let us define u to be $(u_1, \ldots, u_n)$ where for every $1 \leq i \leq n$, u_i is defined as follows: If $v_i \in A_i$ then set $u_i = v_i$. Otherwise, $v_i = t_i^{j_i}$ for some $t_i \in R^{\mathbf{A}_i}$. Set u_i to be $t_i[j_i + 1]$ if $j_i < r$ and to be any arbitrary element in A_i otherwise. Let $w = (w_1, \ldots, w_n)$ be any node of type 2. We shall prove that for every $1 \leq i \leq n$, if (v_i, w_i) is an edge of $G(\mathbf{A}_i)$ then so is (u_i, w_i). The claim is obvious whenever $u_i = v_i$. Assume now that $v_i = t_i^{j_i}$ for some $t_i \in R^{\mathbf{A}_i}$. Since w is of type 2 it follows that $w_i = t_i^{j_i+1}$. The claim follows from the fact that $G(\mathbf{A}_i)$ contains edge $(t_i[j_i + 1], t_i^{j_i+1})$.

Let us prove that h' is indeed a homomorphism. Let (u, v) be an edge in $\Pi_i G(\mathbf{A}_i)$ and let $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$. We shall prove that $(h(u), h(v))$ belongs to $G(\mathbf{B})$ by means of a case analysis on the types of u and v. Notice that v is necessarliy of type 2 or 3 since nodes of type 1 or 4 do not have incoming edges.

- u is of type 1. If v is of type 3 the claim follows from the fact that $G(\mathbf{B})$ has an edge from every element in B to every sink vertex. Assume now that v is of type 2, that is, v is of the form $(t_1^j, \ldots, t_n^j)$. Since (u, v) is an edge of $\Pi_i G(\mathbf{A}_i)$ and u is of type 1 it follows that $u_i = t_i[j]$ for every $1 \leq i \leq n$. Hence $u = (\Pi_i t_i)[j]$ and, since h defines a homomorphism, $h(u)$ is the jth component of $h(\Pi_i t_i)$ (h is applied component-wise). Since $h'(u) = h(u)$, it follows that $G(\mathbf{B})$ contains the edge from $h'(u)$ to $h'(v) = h(\Pi_i t_i)^j$.
- u is of type 2. Then necessarily there exists $t_1, \ldots, t_n$ and j such that $u = (t_1^j, \ldots, t_n^j)$ and $v = (t_1^{j+1}, \ldots, t_n^{j+1})$ and the claim follows directly from the definitions.
- u is of type 3. Then v is necessarily of type 3 as well. Furthermore, it follows that if $h'(u)$ is s^j then necessarily $h'(v) = s^{j+1}$.
- u is of type 4. It follows directly from the definition of $h'(u)$ and the fact that every vertex of type 3 is mapped by h' to a sink node.

Conversely, let h' be a homomorphism from $\Pi_i G(\mathbf{A}_i)$ to $G(\mathbf{B})$. We start by showing that there exists a conjunctive query q with r free variables $x_1, \ldots, x_r$ such that for every $\mathbf{C}$ among $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$,

$$q(G(\mathbf{C})) = R^{\mathbf{C}}\,. \tag{1}$$

Let $\mathbf{C}$ be among $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$. Consider first the unary conjunctive query $p(z_1)$ stating that there is a directed path of lenght r starting at z_1. Formally,

$$p(z_1) = \exists z_2, \ldots, z_{r+1}(\bigwedge_{1 \leq j \leq r} E(z_j, z_{j+1}))$$

where E is the edge relation.

By the construction of $G(\mathbf{C})$ it follows that $p(G(C))$ is precisely the projection of $R^{\mathbf{C}}$ to the fist coordinate, which we have assumed to be C. Now we define q to be the conjunctive query

$$q(x_1, \ldots, x_r) = \exists y_1 \ldots y_r(\bigwedge_{1 \leq j \leq r} p(x_j) \bigwedge_{1 \leq j \leq r} E(x_j, y_j) \wedge \bigwedge_{1 \leq j < r} E(y_j, y_{j+1}))$$

where we assume that $p(x_1), \ldots, p(x_n)$ use different existential variables. It is not difficult to see that q satisfies (1).

With the help of q it is very easy to complete the proof. Indeed, Now, let t be any tuple in $R^{\Pi_i \mathbf{A}_i}$. It follows that $t = \Pi_i t_i$, where $t_i \in R^{\mathbf{A}_i}$ for every $1 \leq i \leq n$. It follows from (1) that $t_i \in q(G(\mathbf{A}_i))$ for every $1 \leq i \leq n$. Then, $t \in q(\Pi_i G(\mathbf{A}_i))$ and therefore, since conjunctive queries are preserved by homomorphisms, $h(t)$ belongs to $q(G(\mathbf{B}))$. It follows from (1) that $h(t) \in R^{\mathbf{B}}$.

Note that for every digraph among $G(\mathbf{A}_1), \ldots, G(\mathbf{A}_m), G(\mathbf{B})$ the maximum length of a directed path is r. This will be used in the proof of Theorem 2. ◀

4 First application: instance-level query definability

Instance-level query definability refers to definability of relations inside a given database instance, with respect to some query language. We focus here on the *CQ-definability problem*, which consists in deciding, given a database instance I and a relation S over the active domain of I, whether there is a conjuctive query q such that $q(I) = S$. Recall that a database instance, for present purposes, can be defined as a finite structure (note that conjunctive queries are domain-independent).

It has been long known that the CQ-definability problem is decidable in coNExpTime (see discussion and references in [13]). For the sake of completeness we include a short description of the algorithm that places the problem in coNExpTime. To describe it, we need to introduce the notion of polymorphism. A *polymorphism* of a relation $S \subseteq D^r$ is any operation $f : D^k \to D$, $k \geq 0$ such that the following holds: for every k (not necessarily different) tuples $t_1, \cdots, t_k \in S$, the tuple, $f(t_1, \cdots, t_k)$, obtained by applying f componentwise to $t_1, \cdots, t_k$, belongs also to S. It is well known (see for example [13]) that $q(I) = S$ for some conjunctive query q if and only if every m-ary operation that is a polymorphism of all relations in I is also a polymorphism of S, where m is the size (number of tuples) of S. The coNExpTime algorithm for the CQ-definability is a straightforward application of the previous result: the algorithm, basically, guesses operation f and verifies that f is a polymorphism of all relations in I but not of S.

Additionally, it was shown in [13] that the CQ-definability problem is coNExpTime-complete, even for database instances with a bounded active domain size. However, the proof used relations of arbitrarily large arity. We show that the same problem is coNExpTime-complete for a fixed schema (but without a bound on the size of the domains of the database instances).

▶ **Theorem 2.** *The CQ-definability problem is co*NExpTime*-hard already for unary queries over a fixed schema consisting of a single binary relation.*

Proof. We shall give a reduction from the PHP with a single binary relation (Theorem 1(3)). Let $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}$ be the input of PHP where the schema contains only a binary relation R. Inspection of the proof of Theorem 1(3) shows that we may assume that, in each of these stuctures, the maximum length of a directed path is precisely r, for some fixed natural number r. Let $\mathbf{C}$ be the database instance consisting of the disjoint union of $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and $\mathbf{B}$, extended with the facts $R(a_i, x)$ for all $i \leq n$ and $x \in \mathbf{A}_i$, and $R(b, x)$ for all $x \in \mathbf{B}$, where $a_1, \ldots, a_n$ and b are fresh elements. Observe that each a_i, and also b, by construction, has an outgoing path of length $r+1$, while no other elements have an outgoing path of length $r+1$. We make use of this below. Let $S = \{a_1, \ldots, a_n\}$. Then we claim that $\mathbf{A}_1 \otimes ... \otimes \mathbf{A}_n \to \mathbf{B}$ if and only if S is not definable inside $\mathbf{C}$ by a conjunctive query. In one direction, if $\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n \to \mathbf{B}$ then clearly S is not definable by a conjunctive query, because, by homomorphism preservation, the same conjunctive query would have to select b. On the other hand, if $\mathbf{A}_1 \otimes ... \otimes \mathbf{A}_n \not\to \mathbf{B}$, then we can construct a query q defining S as follows: first we take $q_1 = \exists y_1, \ldots, y_k\, \psi(y_1, \ldots, y_k)$ to be the canonical Boolean conjunctive query of $\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n$, and, then, we define $q(x)$ to be the unary conjunctive query

$$\exists y_1 \ldots, y_k\, (R(x, y_1) \wedge \cdots \wedge R(x, y_k) \wedge \psi(y_1, \ldots, y_k))$$

expressing that q_1 holds in the submodel of $\mathbf{C}$ consisting of all elements reachable (in one step) from the element denoted by x. By construction, $q(\mathbf{C})$ includes all of S and excludes b. It is also easy to see that $q(\mathbf{C})$ contains no elements other than $a_1, \ldots, a_n$ and b (we are using here the fact that structures $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and $\mathbf{B}$ do not contain a path of length $r+1$). Therefore, q defines S. ◀

It is also natural to consider a *generalized* version of CQ-definability, where the input is a finite sequence of pairs $(I_1, S_1), \ldots, (I_n, S_n)$, where each I_i is a database instance and each S_i is a relation over the active domain of I_i (all of the same arity), and the task is to decide whether there is a conjunctive query q, such that, for all $i \leq n$, $q(I_i) = S_i$.

▶ **Theorem 3.** *The generalized CQ-definability problem is co*NExpTime*-complete.*

Proof. The lower bound follows immediately from Theorem 2. For the upper bound: let $(I_1, S_1), \ldots, (I_n, S_n)$ be a given input for the generalized CQ-definability problem. We may assume without loss of generality that the active domains of $I_1, \ldots, I_n$ are disjoint. We extend the schema with an additional binary relation E, and construct a new instance I that is the disjoint union of $I_1, \ldots, I_n$, where E is interpreted as the equivalence relation consisting of all pairs (a, b), such that a and b belong to the active domain of the same instance I_i. Furthermore, let $S = S_1 \cup \cdots \cup S_n$.

Let q be any conjunctive query, over the original schema, such that $q(I_i) = S_i$ for all $i \leq n$. Let q' be obtained from q by adding conjuncts $E(x, y)$ for all pairs of variables x, y occurring (free or bound) in the query q. Then it is easy to show that $q'(I) = S$. Conversely, let q be any conjunctive query (possibly referring to the binary relation E) such that $q(I) = S$. Let q' be obtained from q by dropping all conjuncts involving the relation E. Then it is easy to show that $q'(I_i) = S_i$ for all $i \leq n$. We conclude that $(I_1, S_1), \ldots, (I_n, S_n)$ is a yes-instance for the generalized CQ-definability problem if and only if (I, S) is a yes-instance for the CQ-definability problem. ◀

Analogous to the CQ-definbility problem, one can consider the FO-definability problem, where the task is to decide, given a database instance I and a relation S over the domain of I, whether there is a first-order query q such that $q(I) = S$. This problem was considered, under the name BP-PAIR, in [9], where it was observed that the Banchilon-Paredaens completeness

theorem [3, 12] implies that this problem is in coNP and co-GI-hard, where GI is the class of problems reducible to the graph-isomorphism problem. The same holds for the generalized FO-definability problem, also known as BP-PAIRS. Determining the exact complexity of BP-PAIR and BP-PAIRS is an open problem [9]. Recently, in [2], instance-level definability was studied for regular path queries and for conjunctive regular path queries in the context of graph databases.

5 Second application: the fitting problem for schema mappings

The fitting problem for schema mappings was introduced and studied in [1]. We briefly review the relevant definitions. Fix two disjoint finite relational schemas, $\mathbf{S}$ and $\mathbf{T}$, which we will call the *source schema* and the *target schema*. A *GLAV (Global-and-Local-As-View) constraint* is a first-order sentence of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$$

where $\phi(\mathbf{x})$ is a conjunction of one or more atomic relational formulas over the source schema containing all the variables in $\mathbf{x}$, and where $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of one or more relational atomic formulas over the target schema containing all variables in $\mathbf{y}$. As in the case of conjunctive queries, for simplicity, we assume that the atoms in a GLAV constraint do not contain constants. A *GLAV schema mapping* is a finite set of GLAV constraints.

GLAV schema mappings are used extensively in *data exchange* and in *data integration* [11]. They provide the formal foundation for these data-interoperability tasks by specifying the relationships between the two schemas. Two important special cases of GLAV schema mappings are *GAV (Global-As-View) schema mappings* and *LAV (Local-As-View) schema mappings*. These consists of GAV constraints, and LAV constraints, respectively. A GAV constraint is a GLAV constraint whose consequent $\psi(\mathbf{x})$ consists of a single atomic formula without existential quantifiers, while a LAV constraint is a GLAV constraint whose antecedent $\phi(\mathbf{x}, \mathbf{y})$ consists of a single atomic formula. Finally, a 1GAV schema mapping is a GAV schema mapping such that each relation from the target schema occurs in only one constraint. We can think of a GAV schema mapping (or, a 1GAV schema mapping) as associating, to each target relation, a UCQ view over the source (respectively, a CQ view over the source), while we can think of a LAV schema mapping as associating, to each source relation, a CQ view over the target. GAV, LAV, and GLAV schema mappings are the most widely studied, and the most widely used, types of schema mappings.

Consider a schema mapping M and a pair (I, J), where I and J are database instances over the source $\mathbf{S}$ and the target schema $\mathbf{T}$, respectively. When (I, J), viewed as a single database instance over the union of the two schemas, satisfies the constraints of M, then we say that J is a *solution* of I (with respect to M). We say that J is a *universal solution* for I (with respect to a schema mapping M), if J is a solution for I, and for every solution J' of I, there is a homomorphism from $h : J \rightarrow J'$, where $h(a) = a$ for all a in the active domain of I. It was argued in [8] that universal solutions are the preferred solutions in data exchange. Furthermore, it was shown in [8] that, in the case of GLAV schema mappings, every source instance has a universal solution, and a universal solution can be constructed in polynomial time (data complexity).

Several methodologies have been proposed and used for obtaining schema mappings in practice. In particular, in [1], a *data example*-driven approach to schema mapping design is advocated. An (unlabelled) *data example*, here, is a pair (I, J), where I is a finite structure over the source schema and J is a finite structure over the target schema. Data examples

data examples	LAV	GAV	1GAV	GLAV
universal	NP-cmp [1]	coNP-cmp [1]	coNExpTime-cmp*	Π_2^p-cmp [1]
pos. & neg.	coNExpTime-cmp*	coNP-cmp*	coNExpTime-cmp*	in coN2ExpTime and coNExpTime-hard*

Figure 1 Complexity of variants of the fitting problem for schema mappings (* marks new results).

can be used in different ways to describe a schema mapping. We say that a data example (I, J) is a *positive example* for a schema mapping M if J is a solution for I with respect to M, and it is a *negative example* otherwise. Finally, (I, J) is a *universal example* for M if J is a universal solution for I with respect to M.

The *fitting problem for GLAV (GAV, LAV) schema mappings with positive/negative/universal examples* is the problem where the input is a finite collection of data examples, each labeled as being a positive example or a negative example or a universal example, and the problem is to decide whether there exists a GLAV (GAV, 1GAV, LAV) schema mapping that is consistent with this marking (in other words, that "fits" these data examples). We will follow the literature by considering the *data complexity* of this problem, where the input is the collection of marked data examples, while the source and target schema are assumed to be fixed. The fitting problem for universal examples was studied in [1]. It was shown there that this problem is coNP-complete for GAV schema mappings; NP-complete for LAV schema mappings; and Π_2^p-complete for GLAV schema mappings. It was also shown in [1] that if, for a given set of universal examples, a fitting GLAV (or GAV, LAV) schema mapping exists, then there is one whose size is linear in the combined size of the data examples.

Although it is argued in [1] that universal examples are the most natural and well-behaved type of data examples, it is also important to consider fitting problems where the input is a collection of positive and negative examples. Indeed, for richer schema mapping languages (beyond GLAV), in general, a given source instance may not *have* a universal solution, and hence, we cannot always work with universal examples. Below, we determine the complexity of the fitting problem with positive and negative examples, for the various schema mapping languages introduced above. The main results are summarized in Figure 1.

First, we establish some convenient lemmas. First, the *direct product* construction naturally generalizes to instances with designated elements. Let $n, m \geq 0$ and let $(I_1, \mathbf{a}_1), \ldots, (I_n, \mathbf{a}_n)$, where each I_n is an instance, over the same schema, and where each $\mathbf{a}_i$ is a sequence of m elements of the active domain of I_i ($m \geq 0$). We denote by $\Pi_{1 \leq i \leq n}(I_i, \mathbf{a}_i)$ the pair $(\Pi_{1 \leq i \leq n}(I_i), \mathbf{b})$, where $\mathbf{b}$ is the m-tuple that is the direct product of $\mathbf{a}_1, \ldots, \mathbf{a}_n$.

We will make use of the following fundamental notion from database theory: the *canonical query* of $(I, \mathbf{a})$ is the query $q(\mathbf{x}) = \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ obtained by associating a first-order variable to each element of the active domain of I, taking ψ to be conjunction of all facts of I using these variables, and existentially quantifying all variables corresponding to elements that do not belong to $\mathbf{a}$. Note that the free variables $\mathbf{x}$ of the query are the variables that correspond to the elements in $\mathbf{a}$.

▶ **Theorem 4.** **(i)** *Let E be a finite collection of positive and negative data examples over a fixed source and target schema. If there exists a LAV schema mapping that fits E, then there exists one of size at most $2^{O(n)}$, where n is the total size of the data examples in E.*

(ii) *The fitting problem for LAV schema mappings with positive and negative examples (over a fixed source and target schema) is co*NExpTime*-complete.*

Proof. (i) Let E be a finite set of positive and negative labeled data examples with source and target schemas $\mathbf{S}$ and $\mathbf{T}$. Let F be the finite set consisting of all atomic formulas over $\mathbf{S}$, modulo variable renaming. For every $R(\mathbf{x}) \in F$, let $(J^*_{R(\mathbf{x})}, \mathbf{a}^*_{R(\mathbf{x})})$ be the direct product $\Pi\{(J, \mathbf{a}) \mid (I, J) \in E$ is a positive data example and $I \models R(\mathbf{a})\}$. Let $q_{R(\mathbf{x})}$ be the canonical query of $(J^*_{R(\mathbf{x})}, \mathbf{a}^*_{R(\mathbf{x})})$. Take M^* to be the schema mapping consisting of all LAV constraints of the form $\forall\mathbf{x}(R(\mathbf{x}) \rightarrow q_{R(\mathbf{x})}(\mathbf{x}))$, for $R(\mathbf{x}) \in F$.

Note that the size of M^* is single exponential in $||E||$: since the schemas $\mathbf{S}$ and $\mathbf{T}$ are fixed, the number of formulas over $\mathbf{S}$, modulo variable renaming, is bounded. Moreover, the cardinality of S is bounded linearly by the number of facts in the data examples belonging to E. It follows that M^* consists of polynomially many LAV constraints, each of at most singly exponential size.

Furthermore, it follows from the construction of M^* that for every positively labeled data example $(I, J) \in E$, J is a solution for I with respect to M^*. Indeed, consider any LAV constraint $\forall\mathbf{x}(R(\mathbf{x}) \rightarrow q_{R(\mathbf{x})}(\mathbf{x}))$ of M^*, and suppose $R(\mathbf{a}) \in I$. By construction, $(J^*_{R(\mathbf{x})}, \mathbf{a}^*_{R(\mathbf{x})}) \rightarrow (J, \mathbf{a})$, which means that $q_{R(\mathbf{x})}(\mathbf{a})$ is satisfied in J.

It remains to show that every negatively labeled data example $(I, J) \in E$ falsifies at least one LAV constraint from M^*. For the sake of a contradiction, assume that this is not the case. Let M be the LAV schema mapping that fits E, which we have assumed exists. We know that (I, J) falsifies at least one LAV constraint from M. Let this LAV constraint be of the form $\forall\mathbf{x}(R(\mathbf{x}) \rightarrow \exists\mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$, and let $R(\mathbf{a}) \in I$ be witness to the falsehood of this constraint in (I, J). Since (I, J) satisfies all constraints of M^*, we know that $q_{R(\mathbf{x})}$, as we defined earlier, is satisfied in $(J, \mathbf{a})$. In other words, $(J^*_{R(\mathbf{x})}, \mathbf{a}^*_{R(\mathbf{x})}) \rightarrow (J, \mathbf{a})$. It follows that $\exists\mathbf{y}\psi(\mathbf{x}, \mathbf{y})$ is *not* satisfied in $(J^*_{R(\mathbf{x})}, \mathbf{a}^*_{R(\mathbf{x})})$, in other words, the canonical instance of $\exists\mathbf{y}\psi(\mathbf{x}, \mathbf{y})$ does *not* homomorphically map to $(J^*_{R(\mathbf{x})}, \mathbf{a}^*_{R(\mathbf{x})})$. At the same time, we know that $\exists\mathbf{y}\psi(\mathbf{x}, \mathbf{y})$ maps homomorphically into each factor instance of which $(J^*_{R(\mathbf{x})}, \mathbf{a}^*_{R(\mathbf{x})})$ is the direct product. This contradicts the basic property of direct products we described in Section 2, namely that every structure that maps homomorphically into a collection of structures, maps homomorphically into their direct product. Therefore, we have reached a contradiction.

(ii) The coNExpTime upper bound follows from the proof of item (i): we construct M^* from the given data examples, as described above. As we noted, it follows from the construction that M^* fits all positively labeled data examples in E. Therefore, we only need to verify that M^* fits the negatively labeled data examples in E. We use here the fact that the problem of checking that a LAV constraint is satisfied in a data example belongs to NP (which follows from the fact that LAV constraints are existential FO sentences).

Lower bound: by reduction from Theorem 1(3): let $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and $\mathbf{B}$ be given, with a single binary relation. Let $\mathbf{T}$ be the schema of these structures, and let $\mathbf{S}$ be the schema consisting of a single unary relation P. By the way, in this case, every LAV schema mapping is equivalent to one that consists of a single LAV constraint (this is because there is only one possible left-hand side for the LAV constraints, due to the particular choice of $\mathbf{S}$, and multiple LAV constraints with the same left-hand side can be combined using conjunction of the right-hand sides). Note that this already shows that in essence, here, we are concerned with finding a fitting conjunctive query again (namely the right-hand side of the unique LAV constraint). For each $i \leq n$, consider the data example $(\{P(0)\}, \mathbf{A}_i)$ where 0 is some fresh value. The PHP then reduces to the complement of the question whether there is a LAV schema mapping that fits the positively labeled examples $(\{P(0)\}, \mathbf{A}_i)$ for $i \leq n$ and the negatively labeled example $(\{P(0)\}, \mathbf{B})$. ◀

In contrast, the situation for GAV is quite different:

▶ **Theorem 5.** 1. *Let E be a finite collection of positive and negative data examples over a fixed source and target schema. If there exists a GAV schema mapping that fits E, then there exists one whose size is polynomial in the total size of the data examples in E.*
2. *The fitting problem for GAV schema mappings with positive and negative examples is co*NP*-complete.*

Proof. For the complexity upper bound, we use the following decision procedure: let a finite set of labeled data examples be given. For each negatively labeled example (I, J), we verify that there is a target relation R and a tuple t of appropriate arity over the domain of I such that (i) $R(t)$ is absent in J; and (ii) for every positively labeled example (I', J') (from the given set of examples) and homomorphism $h : I \to I'$, $R(h(t))$ belongs to J'. If this holds, we answer *Yes*, otherwise *No*. Note that item (ii) involves a coNP test. Since this coNP test is performed at most polynomially many times, this places the entire problem in coNP.

If the procedure answers *Yes*, then a fitting GAV schema mapping indeed exists, namely the schema mapping containing, for each negatively labeled example (I, J), the GAV constraint whose left-hand side is the canonical query of I (where each constant is replaced by a corresponding fresh universally quantified variable) and whose right-hand side is the chosen missing fact $R(t)$ (where each constant is again replaced by the corresponding universally quantified variable). It follows from item (ii) that this GAV constraint is indeed satisfied in all the positively labeled examples. The GAV schema mapping consisting of all GAV constraints constructed in this way (one for each negatively labeled data example) therefore fits E. Note that this schema mapping is of poynomial size.

Conversely, if there is a fitting GAV schema mapping, then the above procedure will answer *Yes*: in order for a GAV schema mapping to fit the examples, it must contain, for each negatively labeled example (I, J), a GAV constraint that fails in (I, J). The conclusion of this constraint (together with the homomorphism witnessing its failure in (I, J)) provides the fact $R(t)$ that is missing in J. Since the same GAV constraint holds in all positive data examples, item (ii) above holds as well.

The coNP-hardness is shown by a reduction from (the complement of) the NP-complete graph homomorphism problem. Indeed, is easy to see that, for any two graphs G, G', we have that $G \to G'$ if and only if there is no fitting GAV schema mapping for the set consisting of the positively labeled data example $(G', \emptyset)$ and the negatively labeled data example $(G, \emptyset)$. ◀

1GAV schema mappings have not been previously considered in the context of schema mapping discovery. Therefore, we study the fitting problem both in the case with positive and negative examples and with universal examples.

▶ **Theorem 6.** *The following problems are co*NExpTime*-complete:*
1. *the fitting problem for 1GAV schema mappings with positive and negative examples*
2. *the fitting problem for 1GAV schema mappings with universal examples*

In both settings it holds that, if there is a fitting schema mapping for a given set of data examples, then there is one of whose size is exponential in the total size of the data examples.

Proof. Before we start, we note that, if M is a GAV schema mapping (in particular, a 1GAV schema mapping), then every source instance I has a universal solution J, such that the active domain of J is included in the active domain of I (indeed, the *chase* procedure described in [1] produces such universal solutions for GAV schema mappings). Moreover, if J is a universal solution for I and the active domain of J is included in the active domain of I, the definition of universality implies that J is a subinstance of every solution of I. In addition, for every pair of instances I, J, we have that J is a universal solution for I with

respect to a GAV schema mapping M if and only if J' is a universal solution for I with respect to M, where J' is the subinstance of J induced by the active domain of I. We will make use of these observations below.

First we show how coNExpTime-hardness of the fitting problem for 1GAV schema mappings with universal examples, by reduction from the CQ-definability problem (cf. Theorem 2) Let (I, S) be a given input for the CQ-definability problem (for a fixed schema $\mathbf{S}$), and let n be the arity of the relation S. Let $\mathbf{T}$ be the schema that the single n-ary relation symbol T, and let J be the $\mathbf{T}$-instance given by the relation S. First, observe that, since the target schema $\mathbf{T}$ consists of a single relation, every 1GAV schema mapping, in this case, consists of at most one GAV constraint, which is of the form $\forall \mathbf{x}\mathbf{y}(\phi(\mathbf{x}, \mathbf{y} \rightarrow T\mathbf{x})$ (where there may be a repetition of variables in the T-atom). We can associate to this GAV constraint a conjunctive query $q(\mathbf{x}) = \exists \mathbf{y}\phi(\mathbf{x}, \mathbf{y})$ (and, conversely, each conjunctive query is associated to a GAV schema mapping in this way). It is easy to see that a 1GAV schema mapping M fits the universal example (I, J), if and only if the corresponding conjunctive query $q(\mathbf{x})$ is such that $q(I) = S$.

Next, we reduce the fitting problem for 1GAV schema mappings with universal examples to the fitting problem for 1GAV schema mappings with positive and negative examples. Let E be a collection of universal examples over source and target schemas $\mathbf{S}, \mathbf{T}$. By our earlier observations, we may assume without loss of generality that the data examples $(I, J) \in E$ are such that the active domain of J is included in the active domain of I. We define a set E' of positive and negative examples. For each $(I, J) \in E$, we include in E' as positive examples the pair (I, J) itself; and we include in E' as negative examples all pairs (I, J') where J' is a $\mathbf{T}$-instance over the active domain of I such that $J \not\subseteq J'$. It is easy to show that a 1GAV schema mapping M fits the positive and negative data examples in E' if and only if M fits the universal examples in E. Moreover, the combined side of the data examples in E' is polynomial in the combined size of the data examples in E, given that the schemas $\mathbf{S}, \mathbf{T}$ are fixed.

Finally, we will show that the fitting problem for 1GAV schema mappings with positive and negative examples is in coNExpTime, and we will establish the existence of single exponential size fitting 1GAV schema mappings. Let E be a finite set of data examples over schemas $\mathbf{S}$ and $\mathbf{T}$. We will restrict attention to the case where $\mathbf{T} = \{T\}$ (the generalization to the case with several target relations is straightforward). Let n be the arity of T. For each negatively labeled data example $(I, J) \in E$, by a *missing fact* of (I, J) we will mean a fact $T(\mathbf{a})$, with values $\mathbf{a}$ from the active domain of I, that does *not* belong to J. If a negatively labeled example $(I, J) \in E$ has no missing fact, then it is easy to see that no 1GAV schema mapping (and in fact, no GLAV schema mapping) fits E. Therefore, we may assume that each negatively labeled data example has at least one missing fact. Let F be the set of all functions that map each negatively labeled data example in E to one of its missing facts. Note that F consists of singly exponentially many maps. We can associate to each map $f \in F$ a 1GAV schema mapping M_f, namely the schema mapping consisting of the 1GAV constraint $\forall \mathbf{x}(q_f(\mathbf{x}) \rightarrow T\mathbf{x}))$, where q_f is the canonical query of the direct product $\Pi\{(I, \mathbf{a}) \mid (I, J) \in E$ is a negatively labeled data example and $f(I, J) = T\mathbf{a}\}$. It follows from the construction that M_f fits every negatively labeled data example in E.

Claim: If there is a 1GAV schema mapping that fits E, then, for some $f \in F$, M_f fits E.

Note that the size of M_f is single exponential. The claim also implies the coNExpTime complexity upper bound we are after: to check that there is *no* fitting 1GAV schema mapping for E, it suffices to guess, for each $f \in F$, a positively labeled data example (I, J) and a variable assignment witnessing the fact that $(I, J) \not\models \forall \mathbf{x}(q_f(\mathbf{x}) \rightarrow T\mathbf{x}))$. Note that the exponentially many exponential-size non-deterministic guesses can be collected into a single exponential size non-deterministic guess.

To prove the claim, let M' be a 1GAV schema mapping that fits E, and let its constraint be of the form $\forall\vec{x}(\phi(\mathbf{x}) \to T(\mathbf{x}))$. Let f be the map that sends each negatively labeled data example $(I, J) \in E$ to a missing target fact that witnesses the violation of the constraint in (I, J). Recall that q_f is the canonical query of $(I^*, \mathbf{a}^*) = \Pi\{(I, \mathbf{a}) \mid (I, J) \in E, I \models \phi(\mathbf{a}), J \not\models T(\mathbf{a}), f(I, J) = T(\mathbf{a})\}$. It follows that q_f is falsified in all negatively labeled data examples $(I, J) \in E$, and hence, M' fits all negatively labeled data examples in E (we had already noted that M' fits all positively labeled data examples). ◀

The coNExpTime lower bound for the LAV case applies to the GLAV case as well (the examples involved have a source instance that consists of a single fact, and it is easy to see that, for such data examples, every GLAV constraint is equivalent to a LAV constraint of at most the same size). The upper bound technique used in the LAV case can also be adapted for GLAV schema mappings, but it no longer yields matching complexity and size bounds.

▶ **Theorem 7.** 1. *Let E be a finite collection of data examples over a fixed source and target schema. If there exists a GLAV schema mapping that fits E, then there exists one of size at most $2^{2^{O(n)}}$, where n is the total size of the data examples in E.*
2. *The fitting problem for GLAV schema mappings with positive and negative examples is in co*N2ExpTime *and co*NExpTime*-hard.*

Proof. (sketch) Clearly, a fitting GLAV schema mapping needs to contain at most one constraint per negative example $(I, J) \in E$. The left-hand side of that GLAV constraint can, without loss of generality, be taken to be the canonical conjunctive query of I. The right-hand side must be a CQ that fails in J under the natural variable assignment. Here, we can apply the same technique as in the proof of Theorem 4: we can take the right-hand side of the constraint to be the canonical query of the direct product of all J' with $(I', J') \in E$ is a positive data example and $I \to I'$. The same arguments used in the proof of Theorem 4 show that, if there exists any GLAV schema mapping that fits E, then the GLAV schema mapping constructed here fits E.

Since the number of homomorphisms $I \to I'$ is in general exponential, the above construction, in general, involves taking the direct product of exponentially many instances. This gives us a double exponential size GLAV constraint. By the same arguments used in the proof of Theorem 4, we can derive a coN2ExpTime complexity upperbound for the fitting problem. ◀

6 Conclusion

We provided a detailed classification of the complexity of PHP under various restrictions. We used these results to obtain tight complexity bounds for instance-level query definability problems and for fitting problems for schema mappings. The precise complexity of the fitting problem for GLAV schema mappings with respect to positive and negative data examples is left as an open problem.

Acknowledgements. We are grateful to Ross Willard for discussions on the topic and for comments on an earlier draft. Ten Cate is supported by NSF grant IIS-1217869. Dalmau is supported by MICCIN grant TIN2013-48031-C4-1.

References

1 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Designing and refining schema mappings via data examples. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 133–144, New York, NY, USA, 2011. ACM.

2 Timos Antonopoulos, Frank Neven, and Frédéric Servais. Definability problems for graph query languages. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 141–152, New York, NY, USA, 2013. ACM.

3 F. Banchilon. On the completeness of query languages for relational databases. In *Proceedings of MFCS*, pages 112–123, 1978.

4 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

5 Nadia Creignou, Phokion G. Kolaitis, and Bruno Zanuttini. Structure identification of boolean relations and plain bases for co-clones. *J. Comput. Syst. Sci.*, 74(7):1103–1115, 2008.

6 Victor Dalmau. *Computational Complexity of Problems over Generalized Formulas*. PhD thesis, Universitat Politècnica de Catalunya, 2000.

7 Rina Dechter and Judea Pearl. Structure identification in relational data. *Artif. Intell.*, 58(1-3):237–270, 1992.

8 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89 – 124, 2005. Database Theory.

9 G.H.L. Fletcher, M. Gyssens, J. Paredaens, and D. Van Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *Knowledge and Data Engineering, IEEE Transactions on*, 21(6):939 –942, june 2009.

10 Peter Jeavons, David A. Cohen, and Marc Gyssens. How to determine the expressive power of constraints. *Constraints*, 4(2):113–131, 1999.

11 Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, pages 61–75, New York, NY, USA, 2005. ACM.

12 J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107 – 111, 1978.

13 Ross Willard. Testing expressibility is hard. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2010.

Regular Queries on Graph Databases

Juan L. Reutter[1], Miguel Romero[2], and Moshe Y. Vardi[3]

1 **Pontificia Universidad Católica de Chile**
jreutter@ing.puc.cl
2 **Universidad de Chile**
mromero@dcc.uchile.cl
3 **Rice University**
vardi@cs.rice.edu

Abstract

Graph databases are currently one of the most popular paradigms for storing data. One of the key conceptual differences between graph and relational databases is the focus on navigational queries that ask whether some nodes are connected by paths satisfying certain restrictions. This focus has driven the definition of several different query languages and the subsequent study of their fundamental properties.

We define the graph query language of *Regular Queries*, which is a natural extension of unions of conjunctive 2-way regular path queries (UC2RPQs) and unions of conjunctive nested 2-way regular path queries (UCN2RPQs). Regular queries allow expressing complex regular patterns between nodes. We formalize regular queries as nonrecursive Datalog programs with *transitive closure* rules. This language has been previously considered, but its algorithmic properties are not well understood.

Our main contribution is to show *elementary* tight bounds for the containment problem for regular queries. Specifically, we show that this problem is 2Expspace-complete. For all extensions of regular queries known to date, the containment problem turns out to be non-elementary. Together with the fact that evaluating regular queries is not harder than evaluating UCN2RPQs, our results show that regular queries achieve a good balance between expressiveness and complexity, and constitute a well-behaved class that deserves further investigation.

1998 ACM Subject Classification H.2.3 Languages – Query languages

Keywords and phrases Graph databases, conjunctive regular path queries, regular queries, containment

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.177

1 Introduction

Graph databases have become a popular data model over the last decade. Important applications include the Semantic Web [3, 4], social network analysis [27], biological networks [34], and several others. The standard model for a graph database is as an edge-labeled directed graph [12, 30]: nodes represent objects and a labeled edge between nodes represents the fact that a particular type of relationship holds between these two objects. For a survey of graph databases, see [1, 5].

Conceptually, graph databases differ from relational databases in that the topology of the data is as important as the data itself. Thus, typical graph database queries are *navigational*, asking whether some nodes are connected by paths satisfying some specific properties. The most basic query language for graph databases is that of *regular-path queries* (RPQs) [24], which selects pairs of nodes that are connected by a path conforming to a

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 177–194

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

regular expression. A natural extension of RPQs is the class of *two-way regular-path queries* (2RPQs), which enable navigation of inverse relationships [14, 15]. In analogy to *conjunctive queries* (CQs) and *union of* CQs (UCQs), the class of *union of conjunctive* two-way regular path queries (UC2RPQs) enable us to perform unions, joins and projections over 2RPQs [14]. The navigational features present in these languages are considered essential in any reasonable graph query language [5].

More expressive languages have been studied, for example, in the context of knowledge bases and description logics [11, 9, 8]. The class of *nested two-way regular path queries* (N2RPQs) and the corresponding class of *union of conjunctive* N2RPQs (UCN2RPQs), extends C2RPQs with an *existential test operator*, inspired in the language XPath [37, 7]. Typical results show that CN2RPQs is a well-behaved class, as it increases the expressive power of C2RPQs without increasing the complexity of evaluation or containment [11, 8]. Yet the regular patterns detected by 2RPQs and N2RPQs are still quite simple: they speak of paths and a restricted form of trees. Thus, these languages cannot express queries involving more complex regular patterns.

One key property that the query classes of UC2RPQs and UCN2RPQs fail to have is that of *algebraic closure*. To see that, note that the relational algebra is defined as the closure of a set of relational operators [2]. Also, the class of CQs is closed under select, project, and join, while UCQs are also closed under union [2]. Similarly, the class of 2RPQs is closed under concatenation, union, and transitive closure. In contrast, UC2RPQs and UCN2RPQs are not closed under transitive closure. For example, the transitive closure of a binary UC2RPQ query is not a UC2RPQ query. Thus, UC2RPQs and UCN2RPQs are not natural classes of graph database queries.

In this paper we study the language of *Regular Queries* (RQ), which result from closing the class of UC2RPQs also under transitive closure. We believe that RQ fully captures regular patterns over graph databases. We define RQ as *binary nonrecursive Datalog* programs with the addition of *transitive closure* rules of the form $S(x, y) \leftarrow R^+(x, y)$. Such a rule defines S as the transitive closure of the predicate R (which may be defined by other rules). The class of RQ queries is a natural extension of UC2RPQs and UCN2RPQs and can express many interesting properties that UCN2RPQs can not (see e.g. [11, 38, 36]), but its algorithmic properties until now have not been well understood.

It is easy to see that the complexity of evaluation of regular queries is the same as for UC2RPQs: NP-complete in combined complexity and NLogspace-complete in data complexity. This is a direct consequence of the fact that regular queries are subsumed by binary *linear* Datalog [21, 19]. Nevertheless, the precise complexity of checking the containment of regular queries has been open so far. This is the focus of this paper.

The *containment problem* for queries asks, given two queries Ω and Ω', whether the answer of Ω is contained in the answer of Ω', over *all* graph databases. Checking query containment is crucial in several contexts, such as query optimization, view-based query answering, querying incomplete databases, and implications of dependencies [13, 18, 28, 31, 26, 32]. A non-elementary upper bound for regular query containment follows from [22, 23]. But, is the complexity of the containment problem elementary or non-elementary? Given the importance of the query-containment problem, a non-elementary lower bound for containment of regular queries would suggest that the class may be too powerful to be useful in practice.

Our main technical contribution is to show elementary tight bounds for the containment problem of regular queries. We attack this problem by considering an equivalent query language, called *nested unions of* C2RPQs (nested UC2RPQs), which is of independent interest. The intuition is that a regular query can be unfolded to construct an equivalent

nested UC2RPQ. This is remiscent of the connection between nonrecursive Datalog and UCQs: each nonrecursive program can be unfolded to construct an equivalent UCQ [2]. Unfoldings of regular queries may involve an exponential blow up in size, and thus we can think of regular queries as a succinct version of nested UC2RPQs. We also analyze the impact of this succinctness on the complexity of the containment problem.

Remarkably, despite the considerable gain in expressive power that comes with nesting UC2RPQs, we are able to show that checking containment of nested UC2RPQs is Expspace-complete, i.e, it is not harder than checking containment of UC2RPQs [14]. Our proof is based on two novel ideas:

1. We reduce containment of nested UC2RPQs, to containment of a 2RPQ in a nested UC2RPQ. The reduction is based on a *serialization* technique, where we represent expansions of nested UC2RPQs as strings, and modify the original nested UC2RPQs accordingly.
2. We show that checking containment of a 2RPQ E in a nested UC2RPQ Γ can be done in Expspace. Here, we exploit automata-theoretic techniques used before, e.g. in [14, 16, 19] to show that containment of UC2RPQs is in Expspace. Nevertheless, our proof requires a deep understanding and a significant refinement of these techniques. The essence of the proof is a suitable representation of the *partial mappings* of Γ to the expansions of E. This representation is robust against nesting and does not involve a non-elementary blow up in size.

This result yields a 2Expspace upper bound for containment of regular queries, and we also provide a matching lower bound. Thus, the succinctness of regular queries is inherent and the containment problem is 2Expspace-complete.

There are several other languages that are either more expressive or incomparable to regular queries. One of the oldest is *GraphLog* [21], which is equivalent to *first-order logic with transitive closure*. More recent languages include *extended* CRPQs [6], which extends CRPQs with *path variables*, XPath for graph databases [35, 33], and algebraic languages such as [29, 36]. Although all these languages have interesting evaluation properties, the containment problem for all of them is undecidable. Another body of research has focused on fragments of Datalog with decidable containment problems. In fact, regular queries were investigated in [11] (under the name of *nested positive* 2RPQs), but the precise complexity of checking containment was left open, with non-elementary tight bounds provided only for strict generalizations of regular queries [38, 10, 11]. Interestingly, the containment problem is non-elementary even for *positive* first-order logic with *unary* transitive closure [11], which is a slight generalization of regular queries. Thus, regular queries seems to be the most expressive fragment of first-order logic with transitive closure that is known to have an elementary containment problem.

Organization. We present preliminaries in Section 2. In Section 3 we introduce regular queries. The containment problem of regular queries is analyzed in Section 4. Finally, in Section 5 we conclude the paper by discussing realistic restrictions on regular queries and future work. Due to space limitations, we only present the main ideas and intuitions behind our proofs. Complete proofs will be given in the full version of the paper.

2 Preliminaries

Graph databases. Let Σ be a finite alphabet. A *graph database* $\mathcal{G}$ over Σ is a pair (V, E), where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$. Thus, each edge in $\mathcal{G}$ is a triple

$(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an a-labeled edge from v to v' in $\mathcal{G}$. We define the finite alphabet $\Sigma^{\pm} = \Sigma \cup \{a^- \mid a \in \Sigma\}$, that is, $\Sigma^{\pm}$ is the extension of Σ with the *inverse* of each symbol. The *completion* $\mathcal{G}^{\pm}$ of a graph database $\mathcal{G}$ over Σ, is a graph database over $\Sigma^{\pm}$ that is obtained from $\mathcal{G}$ by adding the edge (v', a^-, v) for each edge (v, a, v') in $\mathcal{G}$.

Conjunctive Queries. We assume familiarity with relational schemas and relational databases. A *conjunctive query* (CQ) is a formula in the $\exists, \wedge$-fragment of first-order logic. We adopt a rule-based notation: A CQ $\theta(x_1, \ldots, x_n)$ over the relational schema σ is a rule of the form $\theta(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_m(\bar{y}_m)$, where $\bar{x}$ are the free variables, the variables in some $\bar{y}_i$ not mentioned in $\bar{x}$ are the existential quantified variables and R_i is a predicate symbol in σ, for each $1 \leq i \leq m$. The *answer* of a CQ $\theta(x_1, \ldots, x_n)$ over a relational database $\mathcal{D}$ is the set $\theta(\mathcal{D}) = \{\bar{a} \mid \mathcal{D} \models \theta(\bar{a})\}$, of tuples that satisfies θ in $\mathcal{D}$. As usual, if θ is a *boolean* CQ, that is, it has no free variables, we identify the answer false with the empty relation, and true with the relation containing the 0-ary tuple.

We want to use CQs for querying graph databases over a finite alphabet Σ. In order to do this, given an alphabet Σ, we define the schema $\sigma(\Sigma)$ that consists of one binary predicate symbol E_a, for each symbol $a \in \Sigma$. For readability purposes, we identify E_a with a, for each symbol $a \in \Sigma$. Each graph database $\mathcal{G} = (V, E)$ over Σ can be represented as a relational database $\mathcal{D}(\mathcal{G})$ over the schema $\sigma(\Sigma)$: The database $\mathcal{D}(\mathcal{G})$ consists of all facts of the form $E_a(v, v')$ such that (v, a, v') is an edge in $\mathcal{G}$.

A conjunctive query over Σ is simply a conjunctive query over $\sigma(\Sigma^{\pm})$. The answer $\theta(\mathcal{G})$ of a CQ θ over $\mathcal{G}$ is $\theta(\mathcal{D}(\mathcal{G}^{\pm}))$. A union of CQs (UCQ) Θ over Σ is a set $\{\theta_1(\bar{x}), \ldots, \theta_k(\bar{x})\}$ of CQs over Σ with the same free variables. The answer $\Theta(\mathcal{G})$ is $\bigcup_{1 \leq j \leq k} \theta_j(\mathcal{G})$, for each graph database $\mathcal{G}$.

A (U)CQ with equality is a (U)CQ where *equality atoms* of the form $y = y'$ are allowed. Although each CQ with equality can be transformed into an equivalent CQ (without equality) via identification of variables, in some cases it will be useful to work directly with CQs with equality. If φ is a CQ with equality, then its associated CQ (without equality) is denoted by $\mathsf{neq}(\varphi)$.

C2RPQs. The basic mechanism for querying graph databases is the class of *two-way regular path queries*, or 2RPQs [15]. A 2RPQ E over Σ is a regular expression over $\Sigma^{\pm}$. Intuitively, E computes the pairs of nodes connected by a path that conforms to the regular language $L(E)$ defined by E. Formally, the *answer* $E(\mathcal{G})$ of a 2RPQ E over a graph database $\mathcal{G} = (V, E)$ is the set of pairs (v, v') of nodes in V for which there is a word $r_1 \cdots r_p \in L(E)$ such that (v, v') is in the answer of the CQ $\theta(x, y) \leftarrow r_1(x, z_1), \ldots, r_p(z_{p-1}, y)$ over $\mathcal{G}$. Note that, when $p = 0$, $r_1 \cdots r_p = \varepsilon$ and $\theta(x, y)$ becomes $x = y$.

The analogue of CQs in the context of graph databases is the class of *conjunctive* 2RPQs, or C2RPQs [14]. A C2RPQ is a CQ where each atom is a 2RPQ, instead of a symbol in $\sigma(\Sigma^{\pm})$. Formally, a C2RPQ $\gamma(\bar{x})$ over Σ is rule of the form $\gamma(\bar{x}) \leftarrow E_1(y_1, y_1'), \ldots, E_m(y_m, y_m')$, where $\bar{x}$ are the free variables, the variables in $\{y_1, y_1', \ldots, y_m, y_m'\}$ not mentioned in $\bar{x}$ are the existential quantified variables and E_i is a 2RPQ over Σ, for each $1 \leq i \leq m$. The *answer* $\gamma(\mathcal{G})$ of γ over a graph database $\mathcal{G}$ is defined in the obvious way. A union of C2RPQs (UC2RPQ) Γ over Σ is a finite set $\{\gamma_1(\bar{x}), \ldots, \gamma_k(\bar{x})\}$ of C2RPQs over Σ with the same free variables. We define $\Gamma(\mathcal{G})$ to be $\bigcup_{1 \leq j \leq k} \gamma_j(\mathcal{G})$, for each graph database $\mathcal{G}$.

Datalog. While UC2RPQs extends UCQs with a limited form of transitive closure, Datalog extends UCQs with full recursion. A Datalog *program* Π consists of a finite set of rules

of the form $S(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_m(\bar{y}_m)$, where $S, R_1, \ldots, R_m$ are predicate symbols and $\bar{x}, \bar{y}_1, \ldots, \bar{y}_m$ are tuples of variables. A predicate that occurs in the head of a rule is called *intensional* predicate. The rest of the predicates are called *extensional* predicates. IDB(Π) and EDB(Π) denote the set of intensional and extensional predicates, respectively. We assume that each program has a distinguished intensional predicate called Ans.

Let P be an intensional predicate of a Datalog program Π and $\mathcal{D}$ a database with schema EDB(Π). For $i \geq 0$, $P^i_\Pi(\mathcal{D})$ denote the collection of facts about the intensional predicate P that can be deduced from $\mathcal{D}$ by at most i applications of the rules in Π. Let $P^\infty_\Pi(\mathcal{D})$ be $\bigcup_{i\geq 0} P^i_\Pi(\mathcal{D})$. Then, the *answer* $\Pi(\mathcal{D})$ of Π over $\mathcal{D}$ is $Ans^\infty_\Pi(\mathcal{D})$.

A predicate P *depends* on a predicate Q in a Datalog program Π, if Q occurs in the body of a rule ρ of Π and P is the predicate at the head of ρ. The *dependence graph* of Π is a directed graph whose nodes are the predicates of Π and whose edges capture the dependence relation: there is an edge from Q to P if P depends on Q. A program Π is *nonrecursive* if its dependence graph is acyclic, that is, no predicate depends recursively on itself. It is well known that each nonrecursive program can be expressed as a UCQ, via unfolding of the program.

A (nonrecursive) Datalog program over a finite alphabet Σ is a (nonrecursive) Datalog program Π such that EDB(Π) $= \sigma(\Sigma^\pm)$. The *answer* $\Pi(\mathcal{G})$ of a (nonrecursive) Datalog program Π over a graph database $\mathcal{G}$ over Σ is $\Pi(\mathcal{D}(\mathcal{G}^\pm))$.

▶ Remark. Typically, when we analyze a problem involving 2RPQs over Σ, we shall assume that 2RPQs are represented as *one-way nondeterministic finite automata* (1NFA) over alphabet $\Sigma^\pm$. This does not affect the complexity of problems as we can translate a regular expression to an equivalent automaton in polynomial time.

3 Regular Queries

We now introduce the class of *Regular Queries* (RQs) and establish some basic results regarding the complexity of evaluation.

▶ **Definition 1** (Regular query). A *transitive closure rule* is a rule of the form $S(x, y) \leftarrow R^+(x, y)$, where S, R are predicate symbols and x, y are variables. We extend the notions of *intensional predicate*, *extensional predicate* and *dependence graph* to a finite set of Datalog and transitive closure rules in the natural way. A *regular query* Ω over a finite alphabet Σ is a finite set of Datalog and transitive closure rules such that:

1. The extensional predicates of Ω belong to $\sigma(\Sigma^\pm)$.
2. There is a distinguished intensional predicate called Ans of arbitrary arity.
3. All intensional predicates, except maybe Ans, have arity 2.
4. The dependence graph of Ω is acyclic.

The semantics of regular queries is based on the semantics of Datalog. We define a translation function λ that transforms a Datalog rule or a transitive closure rule into a set of Datalog rules. If ρ is a Datalog rule then $\lambda(\rho) = \{\rho\}$. When ρ is a transitive closure rule of the form $\rho = S(x, y) \leftarrow R^+(x, y)$, then $\lambda(\rho)$ contains the rules $\{S(x, y) \leftarrow R(x, y), S(x, y) \leftarrow S(x, z), R(z, y)\}$. We translate each regular query $\Omega = \{\rho_1, \ldots, \rho_k\}$ into a Datalog program $\lambda(\Omega) = \lambda(\rho_1) \cup \cdots \cup \lambda(\rho_k)$. Then, the *answer* $\Omega(\mathcal{G})$ of a regular query Ω over a graph database $\mathcal{G}$ is the answer of $\lambda(\Omega)$ over $\mathcal{G}$.

▶ **Example 2.** Suppose we have a graph database of persons and its relationships. We have relations *knows*, *is a friend* and *is a good friend*, abbreviated k, f and g, respectively. Thus our alphabet is $\Sigma = \{k, f, g\}$. The following query returns all the pairs of persons connected

by a chain of *potential friends*: p and p' are potential friends, if either they are friends, or p just knows p', but they are connected with the same person by a chain of good friends.

$$G(x,y) \leftarrow g(x,y) \qquad R(x,y) \leftarrow f(x,y) \qquad Ans(x,y) \leftarrow R^+(x,y)$$
$$S(x,y) \leftarrow G^+(x,y) \qquad R(x,y) \leftarrow k(x,y), S(x,z), S(y,z)$$

By using ideas from [11], it can be shown that this query cannot be expressed by any UCN2RPQ. ◀

Recall that the *evaluation problem* for regular queries asks, given a RQ Ω, a graph database $\mathcal{G}$ and a tuple $\bar{t}$, whether $\bar{t} \in \Omega(\mathcal{G})$. Each regular query can be translated to a Datalog program, and in fact, this program is *binary* (all intensional predicates have arity 2, except maybe by *Ans*) and *linear* (in the sense of [21]). As a consequence, we can derive tight complexity bound for the evaluation problem [19, 21]. Interestingly, RQs are not harder to evaluate than UCN2RPQs.

▶ **Theorem 3.** *The evaluation problem for regular queries is* NP*-complete in combined complexity and* NLogspace*-complete in data complexity.*

4 Containment of Regular Queries

Given regular queries Ω and Ω' over alphabet Σ, we say that Ω is *contained* in Ω' if $\Omega(\mathcal{G}) \subseteq \Omega'(\mathcal{G})$, for each graph database $\mathcal{G}$ over Σ. The *containment problem* for regular queries asks, given two RQs Ω and Ω', whether Ω is contained in Ω'. We devote the rest of this paper to proving the following theorem:

▶ **Theorem 4.** *The containment problem for regular queries is* 2Expspace*-complete.*

We attack this problem by considering an equivalent language, called *nested* UC2RPQs. As mentioned in the Introduction, each regular query can be unfolded to construct an equivalent exponentially-sized nested UC2RPQ. Thus by considering first nested UC2RPQs, we study the impact of succinctness in the complexity of the containment problem.

4.1 Containment of Nested UC2RPQs

To define formally the class of nested UC2RPQs we introduce a special type of rule. An *extended C2RPQ rule* is a rule of the form $S(x_1, \ldots, x_n) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$, where, for each $1 \leq i \leq m$, either R_i is a 2RPQ or R_i is of the form $R_i = P_i^+$, where P_i is a binary predicate symbol. For a finite set Γ of extended C2RPQ rules, we define *intensional predicates* and the *dependence graph* in the obvious way; now the 2RPQs mentioned in Γ play the role of extensional predicates.

▶ **Definition 5** (Nested UC2RPQ)**.** A *nested UC2RPQ* Γ over Σ is a finite set of extended C2RPQ rules such that:

1. All 2RPQs mentioned in Γ are defined over Σ.
2. There is a distinguished intensional predicate called *Ans* of arbitrary arity.
3. All intensional predicates, except possibly for *Ans*, have arity 2.
4. The dependence graph of Γ is acyclic.
5. For each intensional predicate S, there is a unique occurrence of S over rule bodies of Γ.

Conditions (1)-(4) describe the basic structure of a nested UC2RPQ, and are analogous to that of regular queries. The key condition is Condition (5). It disallows the use of a predicate several times in different parts of the program. If the predicate S was already defined, then S can be used in the body of only one rule, and in the body of that rule, it can be used only once. It is important to note that S can occur several times in the head of rules, that is, it can be defined by more than one rule.

The semantics of nested UC2RPQs is defined in terms of the semantics of regular queries. For each 2RPQ E, let $\delta(E)$ be an equivalent regular query. We define a translation function λ that maps an extended C2RPQ rule to a set of Datalog or transitive closure rules. Let ρ be an extended C2RPQ rule of the form $S(x_1, \ldots, x_n) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$, then $\lambda(\rho)$ contains the following rules:

- One rule of the form $S(x_1, \ldots, x_n) \leftarrow P_1(y_1, y_1'), \ldots, P_m(y_m, y_m')$, where $P_1, \ldots, P_m$ are distinct fresh predicate symbols.
- For each $1 \leq i \leq m$, if $R_i = E$ is a 2RPQ, then we add rules in $\delta(E)$, where all the predicate symbols in $\delta(E)$ expect by Ans are fresh symbols, and Ans is renamed to P_i.
- If $R_i = Q^+$, for a predicate symbol Q, then we add the rule $P_i(x, y) \leftarrow Q^+(x, y)$.

We translate each nested UC2RPQ $\Gamma = \{\rho_1, \ldots, \rho_k\}$ to a regular query $\lambda(\Gamma) = \lambda(\rho_1) \cup \cdots \cup \lambda(\rho_k)$. Then, the *answer* $\Gamma(\mathcal{G})$ of a nested UC2RPQ over a graph database $\mathcal{G}$, is the answer of $\lambda(\Gamma)$ over $\mathcal{G}$.

▶ **Example 6.** The following nested UC2RPQ corresponds to the unfolding of the query in Example 2. Observe how the two occurrences of S now become two occurrences of distinct predicates G_1 and G_2.

$$G_1(x, y) \leftarrow g(x, y) \qquad R(x, y) \leftarrow f(x, y) \qquad\qquad Ans(x, y) \leftarrow R^+(x, y)$$
$$G_2(x, y) \leftarrow g(x, y) \qquad R(x, y) \leftarrow k(x, y), G_1^+(x, z), G_2^+(y, z)$$

◀

In this section, we provide tight complexity bounds for the containment problem for nested UC2RPQs. As it turns out, checking containment of nested UC2RPQs is not harder than checking containment of UC2RPQs.

▶ **Theorem 7.** *The containment problem for nested UC2RPQs is* EXPSPACE*-complete.*

The lower bound holds trivially as containment of UC2RPQs is already EXPSPACE-hard. In order to prove the EXPSPACE upper bound, we use the following approach:

1. We note that containment of nested UC2RPQs can be reduced to containment of boolean nested UC2RPQs.
2. We show that containment of boolean nested UC2RPQs can be reduced to containment of a *boolean* 2RPQ in a boolean nested UC2RPQ. The semantics of a boolean 2RPQ is defined in the obvious way: the result is `true` if the answer of the 2RPQ is nonempty.
3. We prove an EXPSPACE upper bound for containment of a boolean 2RPQ in a boolean nested UC2RPQ.

Step (1) is straightforward and makes our subsequent arguments and definitions significantly simpler. Thus, until the end of this section, we focus on boolean queries. When it is clear from the context, we write 2RPQ and nested UC2RPQ, instead of boolean 2RPQ and boolean nested UC2RPQ.

Step (2) is based on a *serialization* technique, where we represent expansions of nested UC2RPQs as strings, and modify the original nested UC2RPQs accordingly. For step (3)

we combine automata-theoretic techniques [14] with a suitable representation of the *partial mappings* from a nested UC2RPQ to an expansion of a 2RPQ. This representation is robust against nesting, in the sense that does not involve a non-elementary blow up in size.

4.1.1 Reduction to Containment of 2RPQs in nested UC2RPQs

We now show that checking containment of two nested UC2RPQs Γ and Γ' over Σ can be reduced to checking containment of a 2RPQ $\tilde{E}$ in a nested UC2RPQ $\tilde{\Gamma}$ over a larger alphabet Δ. We start by defining the notion of *expansion*, which is central in the analysis of nested UC2RPQs.

Let Γ be a nested UC2RPQ over alphabet Σ and let S be an intensional predicate. We denote by $\mathsf{rules}(S)$ the set of rules in Γ such that S occurs in the head of the rule. An *expansion* φ of S is a CQ with equality over Σ of the form

$$\varphi(x_1, \ldots, x_n) \leftarrow \varphi_1(y_1, y_1'), \ldots, \varphi_m(y_m, y_m')$$

such that there is a rule $\rho \in \mathsf{rules}(S)$ of the form $S(x_1, \ldots, x_n) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$ and the following two conditions hold (note that $n = 0$ if $S = Ans$; otherwise, $n = 2$):

1. For each $1 \leq i \leq m$, if $R_i = E$ is a 2RPQ, then $\varphi_i(y_i, y_i')$ is a CQ with equality of the form
$$\varphi_i(y_i, y_i') \leftarrow r_1(y_i, z_1), r_2(z_1, z_2), \ldots, r_p(z_{p-1}, y_i')$$
where, $p \geq 0$, $r_1 \cdots r_p \in L(E)$, and the z_j's are fresh variables. When $p = 0$, we have that $r_1 \cdots r_p = \varepsilon$, and $\varphi_i(y_i, y_i')$ becomes $y_i = y_i'$.
2. If $R_i = Q^+$ for an intensional predicate Q, then $\varphi_i(y_i, y_i')$ is a CQ with equality of the form
$$\varphi_i(y_i, y_i') \leftarrow \phi_1(w_0, w_1), \phi_2(w_1, w_2), \ldots, \phi_q(w_{q-1}, w_q)$$
where $q \geq 1$, $w_0 = y_i$, $w_q = y_i'$, $w_1, \ldots, w_{q-1}$ are fresh variables and, for each $1 \leq j \leq q$, there is an expansion $\zeta(t_1, t_2)$ of Q such that $\phi_j(w_{j-1}, w_j)$ is the CQ obtained from $\zeta(t_1, t_2)$ by renaming t_1, t_2 by w_{j-1}, w_j, respectively, and renaming the rest of the variables by new fresh variables. In particular, the quantified variables of distinct ϕ_i and ϕ_j are disjoint.

An *expansion* of a nested UC2RPQ is an expansion of its predicate Ans. In particular, any expansion of a nested UC2RPQ is a boolean query. The intuition is that an expansion of a nested UC2RPQ is simply an expansion of its associated Datalog program [19, 16]. Containment of nested UC2RPQs can be characterized in terms of containment of CQs. This is an easy consequence of the semantics of CQs [17, 39] and the fact that each nested UC2RPQ is equivalent to the union of its expansions.

▶ **Proposition 8.** *Let Γ and Γ' be two nested UC2RPQs. Then, Γ is contained in Γ' if and only if, for each expansion φ of Γ, there exists an expansion φ' of Γ' and a containment mapping from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$.*

Here, the definition of containment mapping is slightly different to the usual definition [17], due to the presence of inverses:

▶ **Definition 9.** If θ and θ' are two boolean CQs over Σ, then a *containment mapping* μ from θ' to θ is a mapping from the variables of θ' to the variables of θ such that, for each atom $r(y, y')$ in θ', with $r \in \Sigma^{\pm}$, either $r(\mu(y), \mu(y'))$ is in θ or $r^-(\mu(y'), \mu(y))$ is in θ.

Given two nested UC2RPQs Γ and Γ' over Σ, we shall construct a 2RPQ $\tilde{E}$ and a nested UC2RPQ $\tilde{\Gamma}$ such that Γ is contained in Γ' if and only if $\tilde{E}$ is contained in $\tilde{\Gamma}$. Our reduction is based on two ideas:

1. Expansions of Γ can be "serialized" and represented by *serialized expansions*, which are strings over a larger alphabet Δ. More importantly, serialized expansions constitute a regular language. Thus, we can construct a 2RPQ $\tilde{E}$ such that $L(\tilde{E})$ is precisely the set of serialized expansions of Γ. This technique has been already used before [14, 15].
2. Now we need to serialize Γ'. Proposition 8 basically tell us that Γ is contained in Γ' iff Γ' can be "mapped" to each expansion of Γ. We have replaced Γ by $\tilde{E}$. Thus, expansions of Γ are replaced by serialized expansions. By modifying the 2RPQs mentioned in Γ', we construct a nested UC2RPQ $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to a serialized expansion W of Γ iff Γ' can be mapped to the expansion of Γ represented by W. This is a new technique and constitutes the crux of the reduction.

Serialization of Γ

We can represent each expansion φ of Γ, by a serialized expansion, that is, a string over a suitable alphabet Δ. Suppose first that Γ is simply a C2RPQ, that is, a single rule of the form $Ans() \leftarrow E_1(y_1, y_1'), \ldots, E_m(y_m, y_m')$. Then, an expansion φ corresponds to choose a string $r^i = r_1^i, \ldots, r_{p_i}^i \in L(E_i)$, for each $1 \leq i \leq m$. This can be represented by the string (a similar representation is used in [14])

$$\$y_1 r_1^1 \cdots r_{p_1}^1 y_1' \$ y_2 r_1^2 \cdots r_{p_2}^2 y_2' \$ \cdots \$ y_m r_1^m \cdots r_{p_m}^m y_m' \$$$

Thus the alphabet Δ contains $\Sigma^{\pm}$, a separator \$, and the variable set of Γ. Assume now that Γ consists of two rules of the form

$$P(x, y) \leftarrow F_1(t_1, t_1'), \ldots, F_n(t_n, t_n') \qquad Ans() \leftarrow P^+(y_1, y_1'), E_2(y_2, y_2'), \ldots, E_m(y_m, y_m')$$

Suppose φ is the expansion of Γ resulting from choosing strings $r^i \in L(E_i)$, for each $2 \leq i \leq m$, and for the atom $P^+(y_1, y_1')$, choosing two intermediate variables z_1 and z_2, and three expansions $\varphi_1, \varphi_2, \varphi_3$ of P. Then, we represent φ by the string

$$\$y_1 W_1 \star \$ \star W_2 \star \$ \star W_3 y_1' \$ y_2 r^2 y_2' \$ \cdots \$ y_m r^m y_m' \$$$

where W_1, W_2, W_3 are strings representing $\varphi_1, \varphi_2, \varphi_3$, respectively, as defined before. Now, the alphabet Δ contains additionally the separator $\star$ that represents fresh intermediate variables that appear when we expand a transitive closure atom.

Applying this idea recursively, we can define serialized expansions for a general nested UC2RPQ Γ. Additionally, it can be shown that serialized expansions can be detected by a 1NFA $\tilde{E}$ over Δ. Moreover, we can construct $\tilde{E}$ such that its size is polynomial in the size of Γ. We say that $\tilde{E}$ is the *serialization* of Γ.

Serialization of Γ'

We replaced Γ by $\tilde{E}$. Thus we replaced expansions of Γ by expansions of $\tilde{E}$, which are of the form $\theta^W() \leftarrow w_1(x_0, x_1), w_2(x_1, x_2), \ldots, w_n(x_{n-1}, x_n)$, for a serialized expansion $W = w_1 \cdots w_n$. Suppose W represents an expansion φ of Γ. Our goal is to construct $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to θ^W iff Γ' can be mapped to $\mathsf{neq}(\varphi)$. To construct $\tilde{\Gamma}$, we modify Γ' in order to translate mappings from Γ' to $\mathsf{neq}(\varphi)$, into mappings from $\tilde{\Gamma}$ to θ^W (and vice versa). Next we explain the main difficulties in the construction of $\tilde{\Gamma}$.

Let W be a serialized expansion representing an expansion φ of Γ. Note that some symbols in W, as the variable symbols or the $\star$ symbol, represent a particular variable in φ. If the i-th symbol of W represents a variable in φ, we denote this variable by $\mathsf{var}(i)$. Note

also that equality atoms in φ define equivalence classes over the variables of φ. We write $y \equiv_\varphi y'$ when the variables y, y' belong to the same equivalence class. Thus it could be possible that $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$, or even that $\mathsf{var}(i) = \mathsf{var}(j)$, for two distinct positions $i < j$ in W. This implies that $\mathsf{var}(i)$ and $\mathsf{var}(j)$ are renamed exactly to the same variable in $\mathsf{neq}(\varphi)$. Then, in order to simulate mappings from Γ' to $\mathsf{neq}(\varphi)$, we have to consider positions i and j as equivalent, that is, we must be able to "jump" between positions i and j, whenever necessary.

To overcome this problem, we introduce the notion of *equality string*. Equality strings are strings over $\Delta^\pm$ with the following key property. For positions $1 \leq i < j \leq |W|$, $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$ iff there is an equality string that can be "folded" into W from i to j. Intuitively, a string α can be folded into W if α can be read in W by a two-way automaton that outputs symbol r, each time it is read from left-to-right, and symbol r^-, each time it is read from right-to-left. For instance, consider the string $W = \$y_1 b^- a y_2 \$$. Then, $by_1^- y_1 b^- aa^-$ can be folded into W from 3 to 4, and $b^- aa^- a y_2 \$$ can be folded into W from 3 to 6.

Next we give some intuition about equality strings. Equality strings are concatenations of *basic* equality strings. There are several types of basic equality strings, each one detecting a particular type of connection between variables. For example, suppose Γ is a query over alphabet $\{a, b\}$ defined by the rules

$$Q(t,s) \leftarrow b(t,s) \qquad\qquad Ans() \leftarrow a(x,y), a + \varepsilon(x,z), Q^+(w,z), aa(a+b)(w,u)$$

and consider the expansion φ of Γ obtained by choosing the strings ε and aab for the second and last atom, respectively, and no intermediate variables for $Q^+(w,z)$. This expansion is represented by the following serialized expansion W

$$\$x \cdot a \cdot y\$x \cdot z\$w \cdot \$t \cdot b \cdot s\$ \cdot z\$w \cdot aab \cdot u\$$$

Note that x and z are equivalent in φ, since it contains the atom $x = z$ (as we chose the string ε). Then any occurrence of x is equivalent to any occurrence of z in W. In particular, the first occurrence of x is equivalent to the last occurrence of z. This is witnessed by an "horizontal" equality string $x \cdot a \cdot y\$x \cdot z\$w \cdot \$t \cdot b \cdot s\$ \cdot z$, that is, a string satisfying the regular expression $x\Delta^* xz\Delta^* z$. In general, we have to consider expressions of the form $x\Delta^* xz_1\Delta^* z_1 z_2 \Delta^* \cdots z_k z \Delta^* z$, for a sequence of variables $z_1, \ldots, z_k$. Note also that the first occurrence of w is equivalent to the occurrence of t in W. This is because t is the first free variable of the expansion $b(t,s)$, which is renamed to w in φ by definition. This is witnessed by a "downward" equality string $w\$t$. In these two examples, we need to know that x and z are in the same "level" and that the "level" of t is the level of w minus 1. In order to achieve this, we incorporate levels to the symbols in Δ, thus the actual definition of Δ is slightly more involved that the one presented here.

Now we are ready to serialize the nested UC2RPQ Γ'. Let $w = w_1 \cdots w_p$ be a string over $\Sigma^\pm$. The *serialization* of w, denoted by $\mathsf{serial}(w)$, is the set of strings over $\Delta^\pm$ of the form $\alpha_0 w_1 \alpha_1 w_2 \alpha_2 \cdots \alpha_{p-1} w_p \alpha_p$, where, for each $0 \leq i \leq p$, the string α_i is either ε or an equality string. If L is a language over $\Sigma^\pm$, then $\mathsf{serial}(L)$ is the language over $\Delta^\pm$ defined by $\mathsf{serial}(L) = \{w' \mid w' \in \mathsf{serial}(w), \text{ for some } w \in L\}$. It can be shown that if $\mathcal{A}$ is a 1NFA accepting the language L over $\Sigma^\pm$, then there exists a 1NFA $\mathcal{A}'$ over $\Delta^\pm$ accepting precisely $\mathsf{serial}(L)$. Moreover, the size of $\mathcal{A}'$ is polynomial in the size of $\mathcal{A}$ and Δ.

The *serialization* $\tilde{\Gamma}$ of Γ' is the nested UC2RPQ over Δ obtained from Γ' by replacing each 2RPQ E in Γ' by $\mathsf{serial}(E)$. It is important to note that the serialization $\tilde{\Gamma}$ of Γ' depends on both Γ and Γ'. This is because it depends on Δ (which, at the same time, depends on Γ). Observe also that the size of $\tilde{\Gamma}$ is polynomial in the size of Δ and Γ', and thus polynomial in

the size of Γ and Γ'. Furthermore, $\tilde{E}$ and $\tilde{\Gamma}$ can be constructed in polynomial time from Γ and Γ'. The following proposition concludes our reduction.

▶ **Proposition 10.** *Let $\tilde{E}$ and $\tilde{\Gamma}$ be the serialization of Γ and Γ', respectively. Then, Γ is contained in Γ' if and only if $\tilde{E}$ is contained in $\tilde{\Gamma}$.*

The idea behind Proposition 10 is as follows. Suppose Γ is contained in Γ'. Let $\theta^W() \leftarrow w_1(x_1, x_2), w_2(x_3, x_4), \ldots, w_n(x_n, x_{n+1})$ be an expansion of $\tilde{E}$, where $W = w_1 \cdots w_n$ is a serialized expansion representing φ. We know that there is a containment mapping μ from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$, for some expansion φ' of Γ'. We have to find an expansion ψ of $\tilde{\Gamma}$ and a containment mapping ν from $\mathsf{neq}(\psi)$ to θ^W. The structure of $\mathsf{neq}(\psi)$ and $\mathsf{neq}(\varphi')$ is the same, except for the strings chosen when we expand 2RPQs. The *internal variables* of an expansion are the fresh variables that appears when we expand 2RPQs (the z_j's in the definition of expansion). The rest of the variables are *external variables*. Thus, the idea is that the external variables of $\mathsf{neq}(\psi)$ and $\mathsf{neq}(\varphi')$ coincide, but the internal variables differ according to the string chosen in both expansions. Now, for each external variable t in $\mathsf{neq}(\psi)$ we define $\nu(t)$ as follows. Let $s = \mu(t)$ (we can apply μ to t as t is also an external variable of $\mathsf{neq}(\varphi')$). Let y be a variable in φ that is renamed to s in the construction of $\mathsf{neq}(\varphi)$. Then, we define $\nu(t)$ to be x_j, for some $1 \leq j \leq n$ such that $\mathsf{var}(j) = y$.

To conclude, we need to define the expansions of 2RPQs in $\mathsf{neq}(\psi)$ and extend ν to the internal variables. Suppose that in $\mathsf{neq}(\varphi')$ we expand a 2RPQ E, between external variables t and t', into the CQ $a_1(t, z_2), a_2(z_2, z_3), \ldots, a_k(z_k, t')$. We want to define an expansion $b_1(t, o_2), b_2(o_2, o_3), \ldots, b_\ell(o_k, t')$ of $\mathsf{serial}(E)$ and extend ν to $\{o_2, \ldots, o_k\}$ ($\nu(t)$ and $\nu(t')$ are already defined). This amounts to finding a folding of $B = b_1 \cdots b_\ell$ into W from i to j, where $\nu(t) = x_i$ and $\nu(t') = x_j$. We define B and this folding simultaneously. We start with $B = a_1 \cdots a_k$ and analyze B from left to right. We examine the values $\mu(t), \mu(z_2), \ldots, \mu(z_k), \mu(t')$ in that order, and according to these values we fold B into W. If all the values are internal variables, there is no problem: we can easily fold B into W. If we see an external variable, we have a problem: we need to "jump" to an equivalent position in order to continue our folding. Thus, we can interleave a suitable equality string α so we can continue our folding into W. In this way, we end up with a string $B \in L(\mathsf{serial}(E))$ (since we only interleave equality strings with $a_1 \cdots a_k \in L(E)$) and with a folding of B into W, as required. The other direction of the proposition is similar.

4.1.2 Containment of 2RPQs in nested UC2RPQs: Upper Bound

Next we show that containment of a 2RPQ in a nested UC2RPQ is in Expspace. We exploit automata-theoretic techniques along the lines of [14, 19, 16]. The main idea is to reduce the complement of the containment problem of a 2RPQ E in a nested UC2RPQ Γ to the non-emptiness of a suitable doubly exponential-sized 1NFA $\mathcal{A}$. The crux of the construction is an intermediate automaton $\mathcal{A}_\Gamma$ that accepts all the (potential) expansions θ of E such that there is a mapping from Γ to θ.

To show that checking containment of two UC2RPQs Γ' and Γ is in Expspace, Calvanese et al. exploit *two-way* NFAs (2NFAs) in order to construct an automaton $\mathcal{A}_\Gamma$ that accepts expansions θ of Γ' such that Γ can be mapped to θ [14]. Moreover, they annotate the expansions with variables of Γ, which indicate the potential mapping of Γ to the expansion. This is possible because the number of possible annotations is bounded, as the number of variables in the queries is bounded. But when Γ is a nested UC2RPQ the number of variables involved in a mapping from Γ to θ is not bounded anymore, and thus it is by no means obvious how to extend the techniques of [14] in this case. Instead, we follow a different

approach, and construct $\mathcal{A}_\Gamma$ directly as a 1NFA. The automaton $\mathcal{A}_\Gamma$ scans the input θ from left to right and in each step, it guesses a "partial mapping" from Γ to θ. This is formalized with the notion of *cut*, which we define next.

We use the fact that the expansions for a 2RPQ E over Σ are CQs of a very particular form, that we call *linear CQs*: they are sequences $r_1(z_1, z_2), \ldots, r_p(z_p, z_{p+1})$ where $r_1 \cdots r_p \in L(E)$ and each z_j is distinct. Thus if we want to decide whether a 2RPQ E is contained in a nested UC2RPQ Γ, we need only to look at those expansions of Γ that can be *flattened* into a linear CQ, i.e., those that can actually be mapped to some linear CQ. Formally, given an expansion φ of Γ, a *linearization* of φ is a linear CQ θ such that there is a containment mapping from $\mathsf{neq}(\varphi)$ to θ. Furthermore, the set of linearizations of Γ is the union of all linearizations of all the expansions of Γ. As we mentioned before, the idea of this proof is to show that the set of linearizations of a nested UC2RPQ Γ can be characterized by an 1NFA $\mathcal{A}_\Gamma$.

The *depth* of a nested UC2RPQ is the maximum length of a directed path from some 2RPQ to the *Ans* predicate in its dependence graph, minus 1. For instance, the query in Example 6 has depth 2.

Cuts. Let Γ be a nested UC2RPQ of depth 0, defined by the rules

$$\begin{array}{rcl} Ans(x_1, \ldots, x_n) & \leftarrow & \Gamma^1_1(u^1_1, v^1_1), \ldots, \Gamma^1_{m_1}(u^1_{m_1}, v^1_{m_1}), \\ \vdots & & \vdots \\ Ans(x_1, \ldots, x_n) & \leftarrow & \Gamma^\ell_1(u^\ell_1, v^\ell_1), \ldots, \Gamma^\ell_{m_\ell}(u^\ell_{m_\ell}, v^\ell_{m_\ell}), \end{array} \tag{1}$$

Let $\mathcal{A}^i_j$ be a 1NFA associated with the 2RPQ Γ^i_j, for each $1 \leq i \leq \ell$ and $1 \leq j \leq m_i$. Let $\mathrm{V}ars(\Gamma, i)$ be the set of variables appearing in the i-th rule of query (1) above. A *cut* of Γ is an ℓ-tuple $(C^1, \ldots, C^\ell)$, where each of the C^i's is either $\perp$ or a triple of form $(\mathrm{Prev}^i, \mathrm{Same}^i, \mathcal{S}^i)$, with $\mathrm{Same}^i \subseteq \mathrm{Prev}^i \subseteq \mathrm{V}ars(\Gamma, i)$ and where $\mathcal{S}^i$ is an m_i-tuple $\mathcal{S}^i = (s^i_1, \ldots, s^i_{m_i})$ containing a state of each of the $\mathcal{A}^i_j$s, $1 \leq j \leq m_i$.

Using the definition of cuts for UC2RPQs, we can naturally extend the definition for cuts of queries of depth > 0. Assume that the rules in Γ with the answer predicate in the head follow form (1) above (note that now some of the Γ^i_js might be predicates P^+ instead of RPQs). Then cuts for Γ are again ℓ-tuples $(C^1, \ldots, C^\ell)$, the only thing that changes is the definition of $\mathcal{S}^i = (s^i_1, \ldots, s^i_{m_i})$, for the cases when Γ^i_j is not a 2RPQ but a predicate of form $P^+(x, y)$. In this case, s^i_j is a cut of query P. Furthermore, *initial cuts* are those in which each Prev^i is empty, and *final cuts* are those in which at least one of the Prev^is is equal to $\mathrm{V}ars(\Gamma, i)$. A cut *marks* a variable x if in all C^is that are not $\perp$ we have that x belongs to Same^i.

Let us now give some intuition on the notion of cuts. Suppose Γ is a C2RPQ and μ is a mapping from Γ to a linearization θ of Γ. If we look at position k in θ, then the partial mapping of μ until this position can be represented by a cut $(\mathrm{Prev}, \mathrm{Same}, \mathcal{S})$: Prev are the variables that are mapped to positions smaller or equal than k and Same are the variables that are mapped precisely to position k. The intuition of $\mathcal{S}$ is as follows. We know that 2RPQs of the form $F(y, y')$ with $y, y' \in (\mathrm{Prev} \cup \mathrm{Same})$ are already satisfied when looking up to k. The only information that is missing is that of the 2RPQs that are "cut" by $(\mathrm{Prev} \cup \mathrm{Same})$, that is, the 2RPQs of the form $F(y, y')$ such that $y \in (\mathrm{Prev} \cup \mathrm{Same})$ and $y' \notin (\mathrm{Prev} \cup \mathrm{Same})$ (or vice versa). Suppose that μ expands F in a linearization given by the string $r_1 \cdots r_p$ and let $s_1, \ldots, s_{p+1}$ be an accepting run of F over $r_1 \cdots r_p$. Consider the mapping μ over $r_1 \cdots r_p$ and suppose r_j is the last symbol to be mapped in positions smaller that k. Then $\mathcal{S}$ contains the state s_j. Note that this is the only information we need to extend this partial mapping

to the one at position $k+1$, and eventually to the global mapping μ. Since we are dealing with UC2RPQs we also need to account for the case when a certain disjunct of Γ cannot be mapped to a linearization: in this case the corresponding triple of the cut is set to $\perp$.

The notion of cut is crucial for two reasons. First, transitions between cuts can be captured by a 1NFA $\mathcal{A}_\Gamma$. Second, it is easy to see that the size of each cut is polynomial in the size of Γ (here we use the fact that Γ is a nested UC2RPQ, instead of a regular query). This implies that the set of all cuts, denoted by $Cuts(\Gamma)$, is of exponential size in the size Γ. This is important to obtain our desired EXPSPACE upper bound.

Transition system based on cuts

Looking to characterize the set of linearizations of nested UC2RPQs, our next step is to define a transition system $T_{(\Gamma,w)}$ defined over cuts of Γ and positions of a word w over $\Sigma^{\pm}$, i.e., over pairs from $Cuts(\Gamma) \times \{1, \ldots, p+1\}$.

Let us shed light on the intuition behind the system. We note first that our transition system, while non-deterministic, can only *advance* to configurations relating greater or equal positions in w. The idea is that a run of $T_{(\Gamma,w)}$ should non-deterministically guess the greatest cuts, in terms of variables in Prev, that can be mapped to each position in w. For the same reason, the transition system can only move towards configurations in which Prev is not smaller that previous configurations.

▶ **Example 11.** Consider query $\Gamma(x,y) \leftarrow g^+(x,z), g^+(y,z)$, stating that there is a path labeled with g^+ between both x and z, and y and z. It is not difficult to see that the CQ $\theta = g(x,x'), g(x',z), g(y',z), g(y,y')$ is a linearization of Γ. The string associated to θ is $w = ggg^-g^-$. A valid run for $T_{(\Gamma,w)}$ over w starts in the initial cut in position 1. It moves to cut $(\{x\},\{x\},\mathcal{S}_1)$, where $\mathcal{S}_1$ only needs to store the state where the automaton for query g^+ is, after having read a g. We then advance to cut $(\{x,z\},\{z\},\mathcal{S}_2)$ in position 3, $(\{x,z\},\{\},\mathcal{S}_3)$ in position 4, where now $\mathcal{S}_3$ again needs to store a state of the automaton for g^+, and finally to cut $(\{x,z,y\},\{y\},\mathcal{S}_4)$ in position 5. Since this last cut is final, we determine that $T_{(\Gamma,w)}$ can advance form an initial state to a final state.

There is a technicality when formally defining this intuition, as variables x and y of Γ need not be mapped right at the start or the end of the the computation of $T_{(\Gamma,w)}$. Thus to state the correctness of this system we need to define a special type of run. Formally, given a nested UC2RPQ $\Gamma(x,y)$ and a word w, then pairs (C,p) and (C',p') define an *accepting run* for Γ over w if the following holds:

- C marks x and C' marks y
- There is an initial cut C_I and a position p_I of w such that one can go from (C_I,p_I) to (C,p) by means of $T_{(\Gamma,w)}$.
- There is a final cut C_F and a position p_F of w such that one can go from (C',p') to (C_F,p_F) by means of $T_{(\Gamma,w)}$.

The following lemma states the correctness of our system.

▶ **Lemma 12.** *Let $\Gamma(x,y)$ be a nested UC2RPQ and $w = r_1, \ldots, r_p$ a string over Σ. There are pairs (C,i) and (C',i') over $Cuts(\Gamma) \times \{1,\ldots,p+1\}$ that define an* accepting run *for Γ over w if and only if there is an expansion φ of Γ and a containment mapping from $\mathsf{neq}(\varphi)$ to the linear CQ $Q_w = r_1(z_1,z_2),\ldots r_p(z_p,z_{p+1})$ that maps x and y to variables z_i and $z_{i'}$.*

Cut Automata. A straightforward idea to continue with the proof is to use the system $T_{(\Gamma,w)}$ to create a deterministic finite automaton that accepts all strings that represent the set of linearizations of Γ. However, after a careful analysis one realizes that doing this in a straightforward way results in a much more expensive algorithm, so a little bit of extra work has to be done to avoid additional exponential blowups in our algorithm. In a nutshell, the idea is to extend the alphabet with information about cuts. Given Γ and w, we add to each symbol of w the information about which cuts are reachable from configurations of the form (C, p) in $T_{(\Gamma,w)}$, where $1 \leq p \leq |w| + 1$.

Formally, from $\Sigma^{\pm}$ we construct the extended alphabet $\Sigma(\Gamma) \times \Sigma^{\pm}$ as follows. Assume that $\mathrm{C}uts(\Gamma)$ contains a number N of cuts. Then

- If Γ is a nested UC2RPQ of depth 0, $\Sigma(\Gamma)$ is an $N + 2$ tuple of subsets of $\mathrm{C}uts(\Gamma)$, i.e., $\Sigma(\Gamma) = (2^{\mathrm{C}uts(\Gamma)})^{N+2}$. In other words, $\Sigma(\Gamma)$ contains a subset of $\mathrm{C}uts(\Gamma)$ for each cut in $\mathrm{C}uts(\Gamma)$ plus two additional subsets.
- Otherwise assume that Γ is of form (1), and that $\mathcal{P}$ is the set of predicates occurring in the rules of Γ that are not 2RPQs. Then $\Sigma(\Gamma) = (2^{\mathrm{C}uts(\Gamma)})^{N+2} \times \bigtimes_{P \in \mathcal{P}} \Sigma(P)$. In other words, it is the cartesian product of the set $(2^{\mathrm{C}uts(\Gamma)})^{N+2}$ with each $\Sigma(P)$, for every predicate P that is a subquery of Γ.

▶ **Lemma 13.** *Let Γ be a nested U2CRPQ. Then the number of different symbols in $\Sigma(\Gamma)$ is double exponential w.r.t. the size of Γ. Furthermore, each symbol in $\Sigma(\Gamma)$ is of size exponential w.r.t. Γ.*

Let us now give some intuition regarding this construction. Let w be a string from $\Sigma(\Gamma) \times \Sigma^{\pm}$, and let $\tau(w)$ be its projection over $\Sigma^{\pm}$. Each symbol (u_p, a_p), for $u_p \in \Sigma(\Gamma)$ contains, in particular, $N + 2$ subsets of $\mathrm{C}uts(\Gamma)$, where $N = |\mathrm{C}uts(\Gamma)|$. The first subset of $\mathrm{C}uts(\Gamma)$ represent all those cuts C such that there is a position $\hat{p}$ in w and an initial cut $\hat{C}$ of Γ such that (C, p) is reachable from $(\hat{C}, \hat{p})$ using $T_{(\Gamma,\tau(w))}$. The second subset of $\mathrm{C}uts(\Gamma)$ contains all those cuts C such that there is a final cut $\hat{C}$ and a position $\hat{p}$ so that $(\hat{C}, \hat{p})$ can be reached from (C, p) using $T_{(\Gamma,\tau(w))}$. We have N subsets remaining, namely one for each cut C of Γ. For each such cut C, its corresponding subset $\mathcal{C}$ contains all those cuts $\hat{C}$ for which the configuration $(\hat{C}, p)$ is reachable from (C, p) using $T_{(\Gamma,\tau(w))}$.

If a string w from $\Sigma(\Gamma) \times \Sigma^{\pm}$ satisfies the above conditions, we say that w has *valid annotations* w.r.t. Γ. We show:

▶ **Lemma 14.** *Let Γ be a nested U2CRPQ. Then the language of all strings over $\Sigma(\Gamma) \times \Sigma^{\pm}$ that have* valid annotations *w.r.t. Γ is regular. Furthermore, one can build an 1NFA that checks this language of size double-exponential in the size of Γ.*

We can finally proceed to build the desired 1NFA A_Γ that gives the strings corresponding to the linearizations of a nested UC2RPQ Γ. This 1NFA needs to simulate the system $T_{(\Gamma,w)}$, from an initial cut of a nested UC2RPQ Γ to a final cut in which all variables have already been mapped. Of course, we have to check, for every subquery $\Gamma^i_j(u^i_j, v^i_j)$ of Γ, whether this query is indeed satisfied when starting in the position assigned to variable u^i_j and finishing on the position assigned to v^i_j. Since we might need to check for more than one such query at any given point, synchronizing all these checks is non-trivial. We do it by relying on the annotations added to strings, as explained above.

▶ **Lemma 15.** *Given a nested UC2RPQ Γ over Σ, one can construct, in exponential time, a 1NFA A_Γ over alphabet $\Sigma(\Gamma) \times \Sigma^{\pm}$ such that A_Γ accepts a string $w = r_1, \ldots, r_p$ with valid annotations w.r.t. Γ if and only if there are pairs (C, i) and (C', i') over $Cuts(\Gamma) \times \{1, \ldots, p+1\}$ that define an accepting run for Γ over w.*

Together with Lemma 12, this Lemma implies that A_Γ characterizes all those strings w that have *valid annotations* w.r.t. Γ such that $\tau(w)$ is accepted by Γ.

Main Proof. To decide containment of a 2RPQ E in a nested UC2RPQ Γ, we proceed as follows:

- Build an 1NFA $\mathcal{A}_E$ for E, extended so that it works with the alphabet $\Sigma(\Gamma) \times \Sigma^{\pm}$.
- Build the 1NFA $\mathcal{A}_{\Sigma(\Gamma)}$ that checks for strings over $\Sigma(\Gamma) \times \Sigma^{\pm}$ that are valid w.r.t. Γ.
- Build the 1NFA $\mathcal{A}_\Gamma$, and complement it, obtaining the automaton $(\mathcal{A}_\Gamma)^C$.

The language of $(\mathcal{A}_\Gamma)^C$ intersected with the language of $\mathcal{A}_{\Sigma(\Gamma)}$ is precisely those strings w with valid annotations such that its projection $\tau(w)$ over $\Sigma^{\pm}$ does not correspond to any linearization of Γ. Thus, if we intersect this language with the one of $\mathcal{A}_E$, we have that the resulting intersection is nonempty if and only if there is an expansion q for E that does not correspond to any of the linearizations of Γ, i.e., if E is not contained in Γ.

Even though some of these automata can be of doubly exponential size w.r.t. E and Γ, we can perform this algorithm in EXPSPACE using a standard on-the-fly implementation. ◀

4.2 Containment of Regular Queries: upper and lower bounds

Expressing a regular query as a nested UC2RPQ may involve an exponential blow up in size. Next we formalize this. Analogous to the case of nested UC2RPQs, the *depth* of a RQ is the maximum length of a directed path from an extensional predicate to the *Ans* predicate in its dependence graph, minus 1. For instance, the query in Example 2 has depth 2. The *height* of a RQ or a nested UC2RPQ is the maximum size of $\mathsf{rules}(S)$ over all intensional predicates S. Recall that $\mathsf{rules}(S)$ is the set of rules whose heads mention the predicate S. Finally, the *width* of a RQ or a nested UC2RPQ is the maximum size of a rule body.

▶ **Proposition 16.** *Let Ω be a regular query. Let h, w and d be the height, the width and the depth of Ω, respectively. Then, Ω is equivalent to a nested UC2RPQ Γ of height at most $h^{O(w^d)}$, width at most w^{d+1}, and depth at most d. In particular, the number of rules in Γ is double-exponential in the size of Ω.*

In view of this result, we cannot use Theorem 7 directly to show a 2EXPSPACE upper bound for the containment problem of two regular queries, as unfolding a regular query Ω may result in a nested UC2RPQ Γ that is of double-exponential size with respect to Ω, and thus the number of cuts in Γ might be of triple-exponential size with respect to Ω.

However, we can further refine the construction we have shown so that the number of cuts depends exponentially only in the width and depth of Γ, but not on the height. The idea is to define cuts of nested UC2RPQs not as a tuple $(C^1, \ldots, C^\ell)$ for each of the ℓ rules in $\mathsf{rules}(Ans)$, but rather as a single triple $(\text{Prev}, \text{Same}, \mathcal{S})$. Intuitively, this corresponds to guessing a priori which disjunct or rule is to be used when witnessing linearizations of Γ. Using this modified construction and Proposition 16 we can then show that the number of cuts is again double-exponential in the size of Ω, which immediately leads to a 2EXPSPACE algorithm, following the ideas of the proof of Theorem 7.

In summary, the 2EXPSPACE algorithm for containment of regular queries works as follows (for simplicity we assume that the inputs are boolean queries).

1. Given regular queries Ω and Ω', we unfold these queries to construct nested UC2RPQs Γ and Γ'. By Proposition 16, we have that (i) $|\Gamma'|$ is doubly-exponential in $|\Omega'|$, (ii) the width of Γ' is single-exponential in $|\Omega'|$, and (iii) the depth of Γ' is polynomial in $|\Omega'|$. Similarly for Γ and Ω.

2. We construct from Γ and Γ' the 2RPQ $\tilde{E}$ and the nested UC2RPQ $\tilde{\Gamma}$ defined in the reduction from Section 4.1.1. This is a polynomial-time reduction, thus basically all the bounds from (1) still hold: $|\tilde{E}|$ is doubly-exponential in $|\Omega|$; $|\tilde{\Gamma}|$ is doubly-exponential in $|\Omega|$ and $|\Omega'|$ (note that $\tilde{\Gamma}$ now depends on Ω too); the width of $\tilde{\Gamma}$ is single-exponential in $|\Omega'|$; the depth of $\tilde{\Gamma}$ is polynomial in $|\Omega'|$.
3. With the refined definition of cut explained above, we obtain that, for a nested UC2RPQ Γ'', the number of cuts is roughly $|\Gamma''|^{w^d}$, where w and d is the width and depth of Γ'', respectively. Thus the number of cuts of our query $\tilde{\Gamma}$ is doubly-exponential in $|\Omega|$ and $|\Omega'|$. It follows that the size of the automata $\mathcal{A}_{\tilde{E}}$, $\mathcal{A}_{\Sigma(\tilde{\Gamma})}$ and $(\mathcal{A}_{\tilde{\Gamma}})^C$ from Section 4.1.2 is triple-exponential in $|\Omega|$ and $|\Omega'|$. To decide whether $\Omega \subseteq \Omega'$, we check the intersection of these automata for emptiness, which can be done in 2Expspace.

By combining techniques from [19, 14], we can show a matching lower bound for containment of regular queries. This conclude the proof of Theorem 4.

5 Conclusions

The results in this paper show that regular queries achieve a good balance between expressiveness and complexity: they are sufficiently expressive to subsume UC2RPQs and UCN2RPQs, and they are not harder to evaluate than UCN2RPQs. While checking containment of regular queries is harder than checking containment of UC2RPQs, it is still elementary. Moreover, all generalizations of regular queries known to date worsen the complexity of the containment problem to non-elementary or even undecidable. Thus we believe that regular queries constitutes a well-behaved class that deserve further investigation.

There are several realistic restrictions on regular queries that lead to better complexity bounds. For instance, it is easy to see that regular queries of *bounded treewidth* [20, 25] can be evaluated in polynomial time in the size of the query and the database. For $k \geq 1$, a regular query has treewidth at most k, if the *underlying graph* of each rule has treewidth at most k. Thus the good behavior of bounded treewidth C2RPQs [6] extends to regular queries. Another natural restriction is that of bounded depth. As a corollary of our results in Section 4, we have that containment for regular queries of bounded depth is Expspace-complete. This is very interesting, as in many situations it may be natural to express regular queries as nested UC2RPQs or to consider regular queries of small depth. In these cases, checking containment is Expspace-complete, the same as for UC2RPQs. Note also that from Theorem 7, it follows that containment of UCN2RPQs is Expspace-complete, which was not known to date.

An interesting research direction is to study the containment problem of a Datalog program in a regular query. Decidability of this problem follows from [22, 23], nevertheless the precise complexity is open. Although it is not clear how to extend the techniques presented in this paper to containment of Datalog in regular queries, we conjecture that this problem is elementary.

Acknowledgements. We'd like to thank Pablo Barceló for helpful discussions on regular queries. The third author is grateful to Vincent Jugé for early discussions on containment of regular queries. First and second authors are supported by the Millennium Nucleus Center for Semantic Web Research Grant NC120004.

References

1 R. Angles, C. Gutierrez. Survey of graph database models. In *ACM Comput. Surv.*, 40(1):1–39, 2008.
2 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3 M. Arenas, J. Peréz. Querying semantic web data with SPARQL. In *PODS* 2011, pages 305–316.
4 M. Arenas, C. Gutierrez, D. Miranker, J. Peréz, J. Sequeda. Querying Semantic Data on the Web? *SIGMOD Record*, 41(4): 6–17 (2012).
5 P. Barceló. Querying graph databases. In *PODS* 2013, pages 175–188.
6 P. Barceló, L. Libkin, A. Lin, P. Wood. Expressive languages for path queries over graph-structured data. In *ACM TODS* 38(4), 2012.
7 P. Barceló, J. Pérez, J. L. Reutter. Relative Expressiveness of Nested Regular Expressions. In *AMW 2012*, pages 180–195.
8 M. Bienvenu, D. Calvanese, M. Ortiz, M. Simkus. Nested regular path queries in description logics. In *KR* 2014.
9 M. Bienvenu, M. Ortiz, M. Simkus. Conjunctive regular path queries in lightweight description logics. In *IJCAI* 2013, pages 761–767.
10 P. Bourhis, M. Krotzsch, S. Rudolph. Query containment for highly expressive datalog fragments. CoRR abs/1406.7801 (2014).
11 P. Bourhis, M. Krotzsch, S. Rudolph. How to Best Nest Regular Path Queries. In DL 2014, Poster.
12 P. Buneman. Semistructured data. In *PODS* 1997, pages 117–121.
13 P. Buneman, S. B. Davidson, G. G. Hillebrand, D. Suciu. A query language and optimization techniques for unstructured data. In *SIGMOD* 1996, pages 505-516.
14 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'00*, pages 176–185.
15 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443–465, 2002.
16 D. Calvanese, G. de Giacomo, M. Y. Vardi. Decidable containment of recursive queries. *Theor. Comput. Sci.* 336(1), pages 33–56, 2005.
17 A. Chandra, Ph. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC* 1977, pp. 77–90.
18 S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim. Optimizing queries with materialized views. In *ICDE* 1995, pages 190-200.
19 S. Chaudhuri, M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *J. Comput. Syst. Sci.* 54(1), pages 61–78, 1997.
20 C. Chekuri, A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.* 239(2), pages 211–229, 2000.
21 M. Consens, A. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS* 1990, pages 404–416.
22 B. Courcelle. The monadic second-order theory of graphs I – Recognizable sets of finite graphs. *Inform. and Comput.,* 85 (1972), pp. 263–267.
23 B. Courcelle. Recursive queries and context-free graph grammars *Theoret. Comput. Sci.*, 78 (1991), pp. 217–244.
24 I. Cruz, A. Mendelzon, P. Wood. A graphical query language supporting recursion. In *SIGMOD Record*, 16(3):323–330, 1987.
25 V. Dalmau, P. Kolaitis, M. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP* 2002, pp. 310–326.

26 R. Fagin, M. Y. Vardi. The theory of data dependencies - An overview. In *ICALP* 1984, pages 1–22.

27 W. Fan. Graph pattern matching revised for social network analysis. In *ICDT* 2012, pages 8–21.

28 M. F. Fernández, D. Florescu, A. Y. Levy, D. Suciu. Verifying integrity constraints on web sites. In *IJCAI* 1999, pages 614–619.

29 G. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, Y. Wu. Relative expressive power of navigational querying on graphs. In *ICDT* 2011, pages 197–207.

30 D. Florescu, A. Levy, A. Mendelzon. Database techniques for the World-Wide-Web: A survey. In *SIGMOD Record*, 27(3):59–74, 1998.

31 M. Friedman, A. Y. Levy, T. D. Millstein. Navigational plans For data integration. In *AAAI/IAAI* 1999, pages 67–73.

32 T. Imielinski, W. Lipski Jr. Incomplete information in relational databases. *J. of the ACM* 31(4), pages 761–791, 1984.

33 E. Kostylev, J. L. Reutter, D. Vrgoc. Containment of Data Graph Queries. In *ICDT* 2014, pages 131–142.

34 Z. Lacroix, H. Murthy, F. Naumann, L. Raschid. Links and paths through life sciences data Sources. In *DILS* 2004, pages 203–211.

35 L. Libkin, W. Martens, D. Vrgoc. Querying graph databases with XPath. In *ICDT* 2013, pages 129–140.

36 L. Libkin, J. L. Reutter, D. Vrgoc. Trial for RDF: adapting graph query languages for RDF data. In *PODS* 2013, pages 201–212.

37 J. Perez, M. Arenas, C. Gutierrez. nSPARQL: A navigational language for RDF. In *J. of Web Semantics* 8, 255–270 (2010).

38 S. Rudolph, M. Krotzsch. Flag & check: data access with monadically defined queries. In *PODS* 2013, pages 151–162.

39 Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operator. In *J. of the ACM* 27(4), 1980, pages 633–655.

Complexity and Expressiveness of ShEx for RDF

Sławek Staworko*[1], Iovka Boneva[1], Jose E. Labra Gayo[2], Samuel Hym[3], Eric G. Prud'hommeaux[4], and Harold Solbrig[5]

1 **LINKS, INRIA & CNRS, University of Lille, France**
2 **University of Oviedo, Spain**
3 **LIFL, University of Lille & CNRS, France**
4 **W3C, Stata Center, MIT**
5 **Mayo Clinic College of Medicine, Rochester, MN, USA**

Abstract

We study the expressiveness and complexity of Shape Expression Schema (ShEx), a novel schema formalism for RDF currently under development by W3C. A ShEx assigns types to the nodes of an RDF graph and allows to constrain the admissible neighborhoods of nodes of a given type with regular bag expressions (RBEs). We formalize and investigate two alternative semantics, multi- and single-type, depending on whether or not a node may have more than one type. We study the expressive power of ShEx and study the complexity of the validation problem. We show that the single-type semantics is strictly more expressive than the multi-type semantics, single-type validation is generally intractable and multi-type validation is feasible for a small (yet practical) subclass of RBEs. To curb the high computational complexity of validation, we propose a natural notion of determinism and show that multi-type validation for the class of deterministic schemas using single-occurrence regular bag expressions (SORBEs) is tractable.

1998 ACM Subject Classification H.2.2 Schema and subschema

Keywords and phrases RDF, Schema, Graph topology, Validation, Complexity, Expressiveness

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.195

1 Introduction

Schemas have a number of important functions in databases. They describe the structure of the database, and its knowledge is essential to any user trying to formulate and execute a query or an update over a database instance. Typically, schemas allow for efficient algorithms for validating the conformance of a given database instance. Schemas also capture the intended meaning of the data stored in database instances and are important for static analysis tasks such as query optimization.

Relational and XML databases have a number of well-established and widely accepted schema formalisms e.g., the SQL Data Definition Language for relational databases and W3C XML Schema and RELAX NG for XML databases. One of the reasons why the RDF data model at its conception has been schema-free was to promote its use and ensure its wide-spread adoption. Indeed, a number of existing RDF applications, such as the linked open data initiative[1], could not have had the same success if the published data had to comply with a rigid schema. However, RDF is slowly but surely becoming an independent database model [1], with applications that were previously considered only in the context of

* Contact author: slawomir.staworko@inria.fr
[1] http://linkeddata.org/

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 195–211

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
<BugReport> {
  descr xsd:string,
  reportedBy @<User>,
  reportedOn xsd:dateTime,
  (reproducedBy @<Employee>,
   reproducedOn xsd:dateTime)?,
  related @<BugReport>*
}

<User> {
  name xsd:string,
  email xsd:string?
}

<Employee> {
  (name xsd:string |
   (first-name xsd:string,
    last-name xsd:string)),
  email xsd:string,
}
```

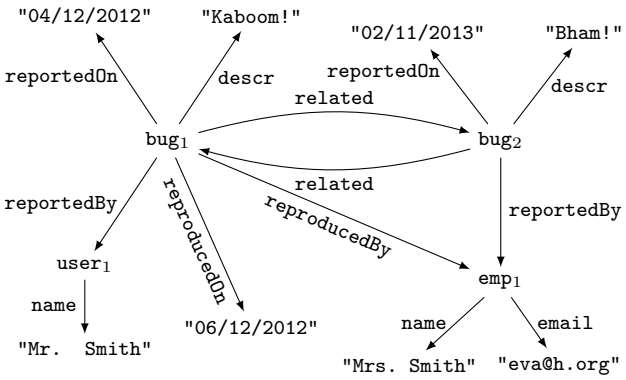


Figure 1 An example of a Shape Expression Schema and a valid RDF graph.

relational and XML databases, for instance data exchange [15, 2]. Such classical applications often rely on the conformance of the data with a set of constraints. Not only has the 'declarative definition of the structure of a graph for validation and description' been clearly identified [40] but also we currently see an emergence of approaches to address this need: apart from the existing but somewhat inadequate RDF Schemas (RDFS) [8], we have OSLC Resource Shapes (ResSh) [32], integrity constraints expressed in OWL [35], and SPARQL queries generated with SPIN templates [5]. In this paper, we investigate Shape Expression Schemas (ShEx) [30, 24], a novel schema formalism for RDF currently under development by W3C [41].

A ShEx allows to define a set of types that impose structural constraints on nodes and their immediate neighborhood. Figure 1 presents a simple example of a Shape Expression Schema for an RDF database storing bug reports.

Essentially, the schema above says that a bug report has a description, a user who reported it, and on what date. Optionally, a bug report may also have an employee who successfully reproduced the bug, and on what date. Finally, a bug report can have a number of related bug reports. For every user we wish to store his/her name and optionally email. For an employee we wish to store his/her name, either as one string or split into the first and last name, and email address.

A Shape Expression Schema defines a set of types to be associated to graph nodes. Each type defines the admissible collection of outgoing edges and the types of the nodes they lead to. Naturally, such a schema bears a strong resemblance to RELAX NG and DTDs which also use regular expressions to describe the allowed collections of children of an XML node. The most important difference comes from the fact that in XML, the children of a node are ordered, and the regular expressions in DTDs and RELAX NG schemas define admissible sequences of children, whereas for RDF graphs, no order on the neighborhood of a given node can be assumed. As the regular expressions used in ShEx define bags (multisets) of symbols rather than sequences, we call them *regular bag expressions* (RBEs).

The semantics of Shape Expression Schemas is quite natural. An RDF graph is valid if it is possible to assign types to the nodes of the graph in a manner that satisfies all shape expressions of the schema. A natural question arises: can a node be assigned more than

one type? In most applications the *multi-type semantics*, which permits assigning multiple types to a node, seems to be more natural. For instance, the RDF graph in Figure 1 requires assigning both the type `User` and the type `Employee` to the node $\texttt{emp}_1$ because $\texttt{emp}_1$ has reported $\texttt{bug}_2$ and reproduced $\texttt{bug}_1$. However, there are applications where the single-type semantics may be more suitable e.g., when modeling graph transformations that visit every node exactly once.

We first study the complexity of the validation problem i.e., checking if a given graph has a valid typing w.r.t. the given schema. Naturally, this problem comes in two flavors, depending on the chosen semantics, and we show significant computational differences between them. While validation for both semantics is generally intractable, the multi-type semantics admits tractable validation for a subclass RBE_0 of disjunction-free expressions that use the Kleene closure on symbols only. This fragment of RBEs is quite practical as it can, for instance, very easily capture the topology of RDF graphs obtained by exported relational databases in virtually any of the proposed approaches for this task (for survey, see [34]). More interestingly, however, we show that the complexity of multi-type validation for ShEx using a class $\mathcal{C}$ of RBEs is closely related (Turing reducible) to the complexity of the satisfiability problem for $\mathcal{C}$ with intersection. We show that in general this problem is NP-complete, which stands in contrast with its analogue for regular word expressions known to be PSPACE-complete [23].

To lower the complexity of validation, we introduce the notion of determinism. Essentially, determinism requires that every shape expression uses at most one type with every label. The shape expressions in Figure 1 are deterministic but the following shape expression is not.

```
<BugReport> {
   descr xsd:string,
   (reportedBy @<User> | reportedBy @<Employee>),
   reportedOn xsd:dateTime,
   (reproducedBy @<Employee>,
    reproducedOn xsd:dateTime)?
   related @<BugReport>*
}
```

This shape expression is not deterministic because `reportedBy` is used with two types: `User` and `Employee`. For deterministic shape expression schemas, we are able to relate the complexity of multi-type validation to the problem of checking membership of a bag of symbols to the language of RBEs. While this problem is known to be NP-complete [22], it is generally simpler than the satisfiability problem, and a number of tractable subclasses has already been identified [6, 22]. All known tractable classes of RBEs require the expressions to be single-occurrence i.e., every symbol of the alphabet is used at most once in an RBE. In the present paper, we show that the full class of *single-occurrence regular bag expressions* (SORBE) has in fact tractable membership. Finally, we consider the problem of validating only a fragment of a graph with preassigned types for its root nodes and argue that for deterministic ShEx using SORBEs, multi-type validation can be performed efficiently, and show that single-type validation can be performed with a single pass over the graph.

Regarding expressiveness of ShEx, the requirement of exactly one type per node makes the single-type semantics more restrictive, and therefore, capable of defining more refined families of graphs than the multi-type semantics. We show that the single-type semantics is in fact strictly more powerful than the multi-type semantics. We also show that both semantics are closed under intersection but neither is closed under union or complement. We then compare the expressive power of ShEx with two standard yardstick logics for graphs: first-order logic (FO_G) and existential monadic second-order logic ($\exists\mathsf{MSO}_\mathsf{G}$). ShEx are not comparable with FO_G but if the RBEs use the Kleene closure on symbols only (e.g. a^*), then

ShEx using such expressions are captured by $\exists MSO_G$. Finally, in our study we compare the expressive power of ShEx with graph grammars, which are generally incomparable, and graph acceptors/automata, which are typically less expressive.

While checking that the data values satisfy constraints is an important part of database validation, in our study we focus only on the core capacity of Shape Expression Schemas to define the graph topology, and therefore, ignore data values. While the exact array of types of value constraints of ShEx is yet to be elaborated, our results identify important computational obstacles that arise from the topology shaping properties of ShEx alone, and furthermore, we present tractable algorithms that can serve as a basis for RDF validation with a number of data value constraints. Our methodology can be compared to using automata to model schema languages for XML databases: while virtually all schema languages for XML allow to define constraints on data values, from simple domain checks in DTDs to key constraints in XML Schema, the results on automata often translate directly to results for schema languages for XML.

The main contributions of the present paper are:

1. We formalize two alternative semantics for ShEx, multi-type and single-type, depending on whether or not a node may have more than one type.
2. We provide a comprehensive understanding of the complexity of validation and show a very close relationship to the complexity of the problem of satisfiability of regular bag expressions with intersection. We show that multi-type validation is tractable for a practical subclass of RBE_0.
3. We propose a notion of determinism for ShEx that allows to curb the complexity of validation and (Turing) reduces validation to checking membership of a bag to the language of RBE. Additionally, we show that single-occurrence regular bag expressions (SORBE) enjoy a tractable membership problem which makes them an attractive candidate for use in deterministic shape expression schemas.
4. We study the expressive power and basic properties of ShEx.

Related work. A number of approaches for validation of RDF has been previously proposed. RDF Schema (RDFS) in essence allows to define a light ontology consisting of types of objects (classes), inclusion dependencies between types (a hierarchy), and specification of the domain and the range of the graph edges of a given label. However, the W3C recommendation does not fix a semantics but only suggest two possible usages: 1) as an ontology allowing to infer types of RDF objects and 2) as a constraint language. It should be noted that only the first use is formalized [18] and it is a common belief that, despite its name, RDF Schema is a basic ontology language rather than a schema language. We point out that the constraints definable with RDFS can be easily captured with ShEx but the converse is not the case. OSLC Resource Shapes (ResSh) [32] essentially extend RDFS by allowing to specify cardinality constraints ?, $*$, $+$, and 1 on types which renders it equivalent to ShEx using single-occurrence RBE_0.

While OWL [13] is an ontology language geared towards inference, it can enforce certain constraints by capturing constraint violations with rules that trigger an inconsistency. However, the class of constraints that can be enforced by OWL is limited due to the fact that OWL is interpreted under open world assumption (OWA) and without unique name assumption (UNA). Consequently, a modified semantics for OWL has been proposed [35] that uses OWA when inferring new facts while employing closed word assumption (CWA) and UNA when enforcing constraints. Since OWL with its standard semantics is widely accepted, concerns have been raised [32] about the potential confusion arising from the mixed

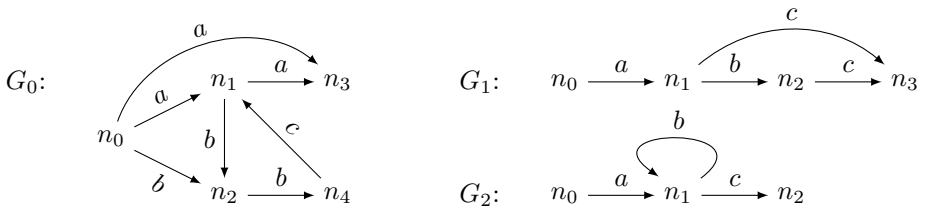


Figure 2 Edge-labeled oriented graphs.

semantics. Such formalisms is, however, very powerful and expressive, easily captures a rich fragment of ShEx (equivalent to $\exists\mathsf{MSO}_\mathsf{G}$), but its computational properties are yet to be characterized. While [37] outlines a method of translating OWL constraints into SPARQL queries, the size of the resulting SPARQL queries seems to depend on the size of the OWL constrains. Currently, we do not know if it is reasonable to assume the size of schema for RDF to be fixed as it may involve large vocabularies of types used by ontologies. This gives a PSPACE upper bound while ShEx enjoys much lower (combined) complexity. Similar criticism applies to other solutions based on SPARQL queries, such as SPIN [5], while they are very powerful and expressive, their use may require significant computational resources.

Finally, we point out an important difference in semantics: while we investigate the existence and construction of a valid typing, all approaches above assume the typing to be given (with `rdf:type` edges) and only verify that the typing is valid. Our approach is more general, we show how to verify the validity of a given typing and that it is the main source of complexity. In particular, we propose a method constructing a maximal (multi-type) valid typing and a method extending the given (possibly invalid) typing to one that is valid.

Organization. In Section 2 we present basic notions. In Section 3 we introduce Shape Expression Schemas (ShEx) and define the single- and multi-type semantics. In Section 4 we study the complexity of the validation problem for ShEx. In Section 5 we introduce a natural notion of determinism for ShEx and identify a rich class of single-occurrence RBEs that together render multi-type validation tractable. In Section 6 we analyze the expressive power of ShEx. Finally, we conclude and discuss related and future work in Section 7. Because of space restriction we omit the proofs: they can be found in the technical report [7].

2 Preliminaries

Because we wish to investigate only the capacity of ShEx to shape the graph topology, we model RDF databases with standard graphs whose edges are labeled by elements of a finite set. In [7] we show how a more general model can be employed without affecting the results.

2.1 Graphs

We assume a finite set Σ of edge labels. An *edge-labeled graph* (or simply a *graph*) is a pair $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is the set of edges. In Figure 2 we present a number of examples of edge-labeled graphs.

In our approach, we shape the topology of a graph based on the immediate outbound neighborhood of nodes. The *labeled outbound neighbourhood* of the node n in the graph $G = (V, E)$ is essentially the set of edges outgoing from n, and is defined as $\mathit{out\text{-}lab\text{-}node}_G(n) = \{(a, m) \in \Sigma \times V \mid (n, a, m) \in E\}$. For instance, $\mathit{out\text{-}lab\text{-}node}_{G_0}(n_0) = \{(a, n_1), (b, n_2), (a, n_3)\}$. On occasions, we use only the collection of outgoing labels, ignoring their target nodes. Note, however, that this collection needs not be representable as neither a set nor by a list of labels

because a node may have multiple outgoing edges with the same label and its neighborhood is not ordered. Take for instance node n_0 in the graph G_0: it has two outgoing a-edges and one outgoing b-edge. Consequently, we employ bags, also known as multisets, which essentially specify the number of occurrences of every symbol.

2.2 Bags of symbols

Let Δ be a finite set of symbols (which is not necessarily Σ). A *bag* over Δ is a function $w : \Delta \to \mathbb{N}$ that maps a symbol to the number of its occurrences. The empty bag ε has 0 occurrences of every symbol i.e., $\varepsilon(a) = 0$ for every $a \in \Delta$. We write $a \in w$ as a short for $w(a) \neq 0$.

We present bags using the notation $\{\!|a, \ldots|\!\}$ with elements possibly being repeated. For example, when $\Delta = \{a, b, c\}$, $w_0 = \{\!|a, a, a, c, c|\!\}$ represents the function $w_0(a) = 3$, $w_0(b) = 0$, and $w_0(c) = 2$. Now, for a given graph $G = (V, E)$ and its node $n \in V$, we define the *bag of outbound labels* of n in G as the bag $\mathit{out\text{-}lab}_G(n) = \{\!|a \mid (n, a, m) \in E|\!\}$. For instance, for the graph G_0 in Figure 2 and the node n_0 we have $\mathit{out\text{-}lab}_{G_0}(n_0) = \{\!|a, a, b|\!\}$.

The *bag union* $w_1 \uplus w_2$ of two bags w_1 and w_2 is defined as $[w_1 \uplus w_2](a) = w_1(a) + w_2(a)$ for all $a \in \Delta$. For instance, $\{\!|a, c, c|\!\} \uplus \{\!|a, b|\!\} = \{\!|a, a, b, c, c|\!\}$. A *bag language* is a set of bags. The bag union of two languages L_1 and L_2 is the language $L_1 \uplus L_2 = \{w_1 \uplus w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. Also, for a given bag language L, we define $L^0 = \{\varepsilon\}$ and $L^i = L \uplus L^{i-1}$ for $i \geq 0$.

2.3 Regular bag expressions

A number of XML schema languages, including DTD, XML Schema, and RelaxNG, uses regular expressions to define the content model (local structure) of types. The popularity of using regular expressions (for words) in validation comes from the fact that they are easy to grasp and to use by a wide range of potential users. Because in the context of RDF graphs the nodes are not ordered, we employ regular expressions for bags, which replace the ordered concatenation operator with its unordered version $\|$, and implicitly the Kleene star by the unordered Kleene star. This family of expressions have been successfully employed to model XML with unordered and mixed content model [6, 11].

A *regular bag expression* (RBE) defines bags by using disjunction "|", unordered concatenation "$\|$", and unordered Kleene star "$*$". Formally, RBEs over Δ are defined with the following grammar $E ::= \epsilon \mid a \mid E^* \mid (E \text{“|”} E) \mid (E \text{“}\|\text{”} E)$, where $a \in \Delta$. Their semantics is defined as follows: $L(\epsilon) = \{\varepsilon\}$, $L(a) = \{\{\!|a|\!\}\}$, $L(E_1 \mid E_2) = L(E_1) \cup L(E_2)$, $L(E_1 \,\|\, E_2) = L(E_1) \uplus L(E_2)$, and $L(E^*) = \bigcup_{i \geq 0} L(E)^i$. We use two standard macros: $E^? := (\epsilon \mid E)$ and $E^+ := (E \,\|\, E^*)$. We also use *intervals* on symbols $a^{[n;m]}$, where $n \in \mathbb{N}$ and $m \in \mathbb{N} \cup \{\infty\}$, with the natural semantics: $L(a^{[n;m]}) = \bigcup_{n \leq i \leq m} L(a)^i$.

We define next different syntactic restrictions on RBEs. We denote $\mathsf{RBE_0}$ the class of expressions constructed using only the $\|$ operator and arbitrary intervals on symbols. By $\mathsf{RBE_1}$ we denote the RBEs of the form $(a_{1,1} \mid \cdots \mid a_{1,k_1}) \| \cdots \| (a_{n,1} \mid \ldots \mid a_{n,k_n})$, with $a_{i,j} \in \Delta$. In the sequel, we also use RBE to denote the family of bag languages definable with regular bag expressions, and it should be clear from the context whether "RBE" stands for the class of expressions, or for the class of languages.

A number of important facts are known about RBE: it is closed under intersection, union, and complement [27], testing membership $w \in L(E)$ is NP-complete [22], and so is testing the emptiness of $L(E_1) \cap L(E_2)$ [11]. Also, when a bag of symbols is viewed as vector of natural numbers (obtained by fixing some total order on Δ), RBE is equivalent to the class of semilinear sets and the class of vectors definable with Presburger arithmetic [17, 29].

3 Shape Expression Schemas

In this section we formally introduce shape expressions schemas and propose two semantics that we study in the remainder of the paper. We assume a finite set of edge labels Σ and a finite set of types Γ. A *shape expression* is an RBE over $\Sigma \times \Gamma$. In the sequel we write $(a, t) \in \Sigma \times \Gamma$ simply as $a :: t$. A *shape expression schema* (ShEx), or simply *schema*, is a tuple $S = (\Sigma, \Gamma, \delta)$, where Σ is a finite set of edge labels, Γ is a finite set of types, and δ is a *type definition* function that maps elements of Γ to bag languages over $\Sigma \times \Gamma$. We only use shape expressions for defining bag languages of δ. Typically, we present a ShEx as a collection of rules of the form $t \to E$ to indicate that $\delta(t) = L(E)$, where $t \in \Gamma$ and E is a shape expression over $\Sigma \times \Gamma$ (naturally, no two rules shall have the same left-hand side). If for some type t a rule is missing, the default rule is $t \to \epsilon$. For a class of RBEs $\mathcal{C}$, by ShEx($\mathcal{C}$) we denote the class of shape expression schemas using only shape expressions in $\mathcal{C}$. Two example schemas follow:

$$S_0 : t_0 \to a :: t_1 \parallel b :: t_2 \qquad t_1 \to (a :: t_1 \mid b :: t_2)^* \qquad t_2 \to b :: t_2 \mid c :: t_1$$

$$S_1 : t_0 \to a :: t_1 \qquad t_1 \to b :: t_2 \parallel c :: t_3 \qquad t_2 \to (b :: t_2)^? \parallel c :: t_3 \qquad t_3 \to \epsilon$$

The semantics of ShEx is natural: a graph is valid if it is possible to assign types to the nodes of the graph in a manner that satisfies the type definitions of the schema. Two variants of semantics can be envisioned depending on whether or not more than one type can be assigned to a node.

3.1 Single-type semantics

We fix a graph $G = (V, E)$ and a schema $S = (\Sigma, \Gamma, \delta)$. A *single-type typing* (or simply an *s-typing*) of G w.r.t. S is a function $\lambda : V \to \Gamma$ that associates with every node $n \in V$ its type $\lambda(n)$. An example of an s-typing of G_0 (Figure 2) w.r.t. S_0 is

$$\lambda_0(n_0) = t_0, \qquad \lambda_0(n_1) = t_1, \qquad \lambda_0(n_2) = t_2, \qquad \lambda_0(n_3) = t_1, \qquad \lambda_0(n_4) = t_2.$$

Next, we identify the conditions that an s-typing needs to satisfy. Given a typing λ and a node $n \in V$ we define the *labeled and typed out-neighborhood* of n w.r.t. λ as the bag over $\Sigma \times \Gamma$

$$\textit{out-lab-type}^{\lambda}_{G}(n) = \{\!\{a :: \lambda(m) \mid (n, a, m) \in E\}\!\}.$$

For instance, for the graph G_0 and the typing λ_0 we have $\textit{out-lab-type}^{\lambda_0}_{G_0}(n_1) = \{\!\{a :: t_1, b :: t_2\}\!\}$ and $\textit{out-lab-type}^{\lambda_0}_{G_0}(n_4) = \{\!\{c :: t_1\}\!\}$.

Now, λ is a *valid* s-typing of S on G if and only if every node satisfies the type definition of its associated type i.e., for every $n \in V$, $\textit{out-lab-type}^{\lambda}_{G}(n) \in \delta(\lambda(n))$. By $L_s(S)$ we denote the set of all graphs that have a valid s-typing w.r.t. the shape expression schema S. For a class $\mathcal{C}$ of bag languages by $\text{ShEx}_s(\mathcal{C})$ we denote the class of graph languages definable under the single-type semantics with shape expression schemas using shape expressions from $\mathcal{C}$ only. Naturally, λ_0 is a valid typing of G_0 w.r.t. S_0. G_1 also has a valid s-typing of S_1:

$$\lambda_1(n_0) = t_0, \qquad \lambda_1(n_1) = t_1, \qquad \lambda_1(n_2) = t_2, \qquad \lambda_1(n_3) = t_3.$$

G_2, however, does not have a valid s-typing w.r.t. S_1.

Note that the notion of single-type semantics defined here is not to be confused with single-type regular tree grammars introduced for XML [26]. In the latter, the term *single-type* designates a syntactical restriction on regular tree grammars that guarantees an efficient top-down validation.

3.2 Multi-type semantics

Again, we assume a fixed graph $G = (V, E)$ and a fixed schema $S = (\Sigma, \Gamma, \delta)$. A *multi-type typing* (or simply an *m-typing*) of G w.r.t. S is a function $\lambda : V \to 2^\Gamma$ that associates with every node of G a set of types. For instance, an m-typing of G_2 w.r.t. S_1 is

$$\lambda_2(n_0) = \{t_0\}, \qquad \lambda_2(n_1) = \{t_1, t_2\}, \qquad \lambda_2(n_2) = \{t_3\}.$$

The labeled and typed out-neighborhood of a node is defined in the same way but note that this time it is a bag over $\Sigma \times 2^\Gamma$. For instance, $\mathit{out\text{-}lab\text{-}type}^{\lambda_2}_{G_2}(n_1) = \{\!\{b :: \{t_1, t_2\}, c :: \{t_3\}\}\!\}$.

Now, a flattening of a bag over $\Sigma \times 2^\Gamma$ is any bag over $\Sigma \times \Gamma$ obtained by choosing one type from every occurrence of every set. For instance, $\mathit{out\text{-}lab\text{-}type}^{\lambda_2}_{G_2}(n_1)$ has two flattenings: $\{\!\{b :: t_1, c :: t_3\}\!\}$ and $\{\!\{b :: t_2, c :: t_3\}\!\}$. Formally, a *flattening* of a bag w over $\Sigma \times 2^\Gamma$ is any bag in the language of the following RBE_1 expression $\mathit{Flatten}(w) = \|_{a::T \in\!\!\!\in w}(|_{t \in T}\ a :: t)$, where $a :: T \in\!\!\!\in w$ indicates that the symbol $a :: T$ is to be considered $w(a :: T)$ times. For instance,

$$\mathit{Flatten}(\{\!\{a :: \{t_0, t_1\}, a :: \{t_0, t_1\}, b :: t_1\}\!\}) = (a :: t_0 \mid a :: t_1) \,\|\, (a :: t_0 \mid a :: t_1) \,\|\, (b :: t_1).$$

By $\mathit{fl\text{-}out\text{-}lab\text{-}type}^\lambda_G(n)$ we denote the set of all flattenings of $\mathit{out\text{-}lab\text{-}type}^\lambda_G(n)$ i.e.

$$\mathit{fl\text{-}out\text{-}lab\text{-}type}^\lambda_G(n) = L(\mathit{Flatten}(\mathit{out\text{-}lab\text{-}type}^\lambda_G(n))).$$

For instance, $\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda_2}_{G_2}(n_1) = \{\{\!\{b :: t_1, c :: t_3\}\!\}, \{\!\{b :: t_2, c :: t_3\}\!\}\}$. Note that while $\mathit{Flatten}(\mathit{out\text{-}lab\text{-}type}^\lambda_G(n))$ is an expression of size polynomial in the size of G and S, the cardinality of the set $\mathit{fl\text{-}out\text{-}lab\text{-}type}^\lambda_G(n)$ may be exponential in the size of G and S. Now, λ is a *valid* m-typing of G w.r.t. S if and only if:

1. it assigns at least one type to every node, $\lambda(n) \neq \emptyset$ for $n \in V$,
2. every node satisfies the type definition of every type assigned to the node i.e., for every $n \in V$ and every $t \in \lambda(n)$, $\mathit{fl\text{-}out\text{-}lab\text{-}type}^\lambda_G(n) \cap \delta(t) \neq \emptyset$.

For instance, λ_2 is a valid multi-type typing of G_2 w.r.t. S_1. By $L_{\mathrm{m}}(S)$ we denote the set of all graphs that have a valid m-typing w.r.t. S. For a class $\mathcal{C}$ of bag languages by $\mathsf{ShEx}_{\mathrm{m}}(\mathcal{C})$ we denote the class of graph languages definable under the multi-type semantics with shape expressions schemas using shape expressions in $\mathcal{C}$ only.

4 Validation

In this section we consider the problem of *validation*: checking whether a given graph has a valid typing w.r.t. a given ShEx. This problem has two parameters: 1) the kind of typing, either single-type or multi-type and 2) the class of regular bag expressions used for type definitions in the schema.

We first point out that the complexity of single-type validation for ShEx(RBE) is NP-complete. The NP upper bound follows from the fact that the membership problem for RBE is in NP. The lower bound is shown with a reduction from 3-colorability of graphs.

▶ **Theorem 1.** *Single-type validation for* ShEx(RBE) *is NP-complete.*

Proof Sketch. Under the single-type semantics, the following schema defines the set of graphs with homomorphism into K_3 i.e., all 3-colorable graphs:

$$t_r \to \_ :: t_b^* \,\|\, \_ :: t_g^* \qquad t_g \to \_ :: t_r^* \,\|\, \_ :: t_b^* \qquad t_b \to \_ :: t_g^* \,\|\, \_ :: t_r^* \qquad \blacktriangleleft$$

For the remaining of this section, we focus on multi-type validation and return briefly to the single-type semantics in the next section.

4.1 Semi-lattice of m-typings

We begin by presenting a downward refinement method that allows to construct a valid m-typing. Take a graph $G = (V, E)$ and a ShEx S, and let $mTyping(G, S)$ be the set of all valid m-typings of the graph G w.r.t. the schema S. $mTyping(G, S)$ is a semi-lattice with the meet operation $\sqcup$ and the (induced) partial order defined as follows:

$$(\lambda_1 \sqcup \lambda_2)(n) = \lambda_1(n) \cup \lambda_2(n) \quad \text{for } n \in V, \qquad \text{and} \qquad \lambda_1 \sqsubseteq \lambda_2 \quad \text{iff} \quad \forall n \in V.\ \lambda_1(n) \subseteq \lambda_2(n).$$

The refinement method works as follows. We begin with the typing λ° that assigns to every node the set of all types, i.e. $\lambda^\circ(n) = \Gamma$ for all $n \in V$, and then we iteratively remove the types that are not satisfied. Every iteration is an application of the *one-step refinement operator* on m-typings defined as follows (with $n \in V$):

$$[Refine(\lambda)](n) = \{t \in \lambda(n) \mid \mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n) \cap \delta(t) \neq \emptyset\}.$$

Clearly, $Refine(\lambda) \sqsubseteq \lambda$, and therefore, the fix-point $Refine^*(\lambda)$ is well-defined. We claim that the procedure outlined above indeed constructs the maximal valid m-typing if one exists.

▶ **Lemma 2.** *For any $\lambda \in mTyping(G, S)$, $\lambda \sqsubseteq Refine^*(\lambda^\circ)$, where $\lambda^\circ(n) = \Gamma$ for all $n \in V$.*

In particular, G satisfies S if and only if $Refine^*(\lambda^\circ)$ is valid, and then, $Refine^*(\lambda^\circ)$ is the $\sqsubseteq$-maximal valid m-typing of G on S. We point out that there does not necessarily exist a unique $\sqsubseteq$-minimal valid m-typing.

4.2 Complexity of Validation

Using the above refinement procedure, we show that multi-type validation is NP-complete for arbitrary regular bag expressions and later identify a tractable fragment.

In essence, performing the refinement procedure requires testing the nonemptyness of the intersection $\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n) \cap \delta(t)$. Recall that $\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ is defined by an RBE_1 expression, i.e. an expression of the form $(a_{1,1} \mid \cdots \mid a_{1,k_1}) \parallel \cdots \parallel (a_{n,1} \mid \ldots \mid a_{n,k_n})$. Therefore, for a class of RBEs $\mathcal{C}$ we identify the following decision problem:

$$\mathsf{INTER}_1(\mathcal{C}) = \{(E_0, E) \in \mathsf{RBE}_1 \times \mathcal{C} \mid L(E_0) \cap L(E) \neq \emptyset\}.$$

Tractability of INTER_1 is a necessary and sufficient condition for tractability of multi-type validation for $\mathsf{ShEx}(\mathcal{C})$. On the one hand, we show that for any class $\mathcal{C}$ of RBEs there exists a polynomial-time reduction from $\mathsf{INTER}_1(\mathcal{C})$ to validation for $\mathsf{ShEx}_{\mathrm{m}}(\mathcal{C})$. On the other hand, the refinement procedure performs a polynomial number of INTER_1 tests (with polynomially-sized inputs). This observation allows us to characterize precisely the complexity of multi-type validation for ShEx(RBE): the lower bound follows from an existing complexity results [11] on testing emptiness of RBEs with intersection and the upper bound follows from a more general result we prove, namely testing satisfiability of RBEs with intersection can be reduced to finding integer solutions to a system of linear equations known to be in NP [28].

▶ **Theorem 3.** *Multi-type validation for* ShEx(RBE) *is NP-complete.*

4.3 The tractable subclass RBE_0

When ShEx use only expressions in RBE_0, the multi-type validation is tractable, which we show by providing a polynomial algorithm for $\mathsf{INTER}_1(\mathsf{RBE}_0)$. We point out that ShEx using

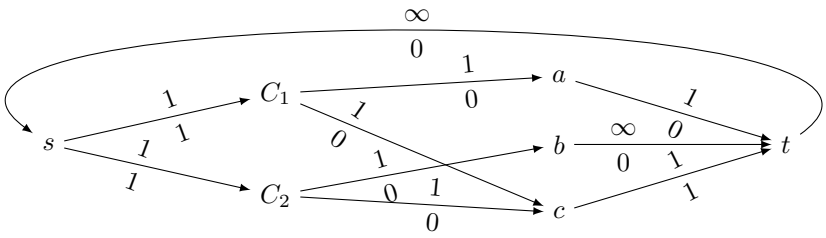


Figure 3 A flow network with a valid flow.

RBE_0 are capable, for instance, of capturing the topology of RDF graphs obtained from exporting relational databases to RDF [34].

We reduce $\mathsf{INTER}_1(\mathsf{RBE}_0)$ to the circulation problem in flow networks. Recall that a flow network is a directed graph with arcs having additionally assigned the amount of flow they require (minimum flow) and the amount of flow they can accept (maximum flow). The *circulation problem* is to find a valid flow i.e., an assignment of flow values to arcs of the flow network so that the flow constraints of every arc are satisfied and at every node the sum of incoming flow is equal to the sum of outgoing flow. This problem has been well-studied and a number of efficient polynomial algorithms exist (cf. [19]).

We illustrate the reduction on the example of $E_0 = (a \mid c) \parallel (b \mid c)$ and $E = a^? \parallel b^* \parallel c$. The corresponding network $N_{E_0,E}$ is presented in Figure 3 where the maximum flow (the minimum flow) of an arc is indicated above (below respectively).

An example of a valid flow f in $N_{E_0,E}$, that corresponds to $\{\!|a, c|\!\} \in L(E_0) \cap L(E)$, is $f(s, C_1) = 1$, $f(C_1, a) = 1$, $f(C_2, b) = 0$, $f(s, C_2) = 1$, $f(C_1, c) = 0$, $f(C_2, c) = 1$, $f(a, t) = 1$, $f(b, t) = 0$, $f(c, t) = 1$, and $f(t, s) = 2$.

▶ **Theorem 4.** *Multi-type validation for* $\mathsf{ShEx}(\mathsf{RBE}_0)$ *is in PTIME.*

5 Determinism

Determinism is a classical tool for decreasing the complexity of validation [20], which we explore next. We propose a suitable and natural notion of determinism for ShEx and show that multi-type validation for deterministic ShEx is not harder than membership of a bag to the language of an RBE. This allows us to identify a large and practical class of single-occurrence regular bag expressions (SORBE) that render validation tractable (Section 5.2). We then investigate the problem of partial validation, where the conformance of only a fragment of the input graph is to be checked (Section 5.3), and which we believe to be an important practical use case of ShEx. We present an optimal algorithm for partial validation which is tractable for classes of deterministic ShEx with tractable membership, for both multi-type and single-type semantics.

5.1 Deterministic Shape Expressions

Essentially, the idea of determinism for a ShEx $S = (\Gamma, \delta)$ is that, knowing the type t of a node $n \in V$ and the label a of an outgoing edge $(n, a, m) \in E$ we should know the type t' that must be satisfied by m if n is to satisfy the type t.

Formally, a shape expression E is *deterministic* if every label $a \in \Sigma$ is used with at most one type $t \in \Gamma$ in E. For instance, $E_1 = a :: t_1 \parallel b :: t_2^* \parallel a :: t_1 \parallel c :: t_2$ is deterministic but $E_2 = a :: t_1 \parallel b :: t_2^* \parallel a :: t_3 \parallel c :: t_2$ is not because the symbol a is used with two different types t_1 and t_3. Now, a shape expression schema $S = (\Sigma, \Gamma, \delta)$ is *deterministic* if it uses only

deterministic shape expressions, and then, by $\delta(t, a)$ we denote the unique type used with the symbol a in the expression used to define $\delta(t)$ (if a is used in this expression).

Recall that the tractability of the refinement method for multi-type validation presented in Section 4.1 depends on the tractability of testing that $\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n) \cap \delta(t)$ is nonempty. When the schema is deterministic, $\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n) \cap \delta(t)$ is nonempty if and only if 1) the bag $\mathit{out\text{-}lab\text{-}type}^{\delta}_{G}(n,t) = \{\!\{a :: \delta(t,a) \mid (n,a,m) \in E\}\!\}$ belongs to $\delta(t)$ and 2) for every $(n,a,m) \in E$, $\delta(t,a)$ belongs to $\lambda(m)$. Using this argument, we show that the tractability of testing membership is a necessary and sufficient condition for the tractability of multi-type validation for deterministic schemas. Formally, for a class $\mathcal{C}$ of RBEs, define the decision problem

$$\mathsf{MEMB}(\mathcal{C}) = \{(w, E) \mid E \in \mathcal{C},\ w \in L(E)\}.$$

Then multi-type validation against a deterministic ShEx using RBEs from the class $\mathcal{C}$ is tractable if and only if $\mathsf{MEMB}(\mathcal{C})$ is tractable.

5.2 Single-occurrence RBE (SORBE)

While the problem of membership of a bag to a language defined by an RBE is in general intractable [22], we identify a rich and practical class of RBEs with tractable membership. This class is obtained by disallowing repeating symbols, while allowing arbitrary intervals on symbols, a restriction on regular expressions commonly imposed in the context of document content models with evidence justifying its use [3, 10, 11, 16, 25]. Formally, a *single-occurrence regular bag expression* (SORBE) over Δ is an RBE that allows at most one occurrence of every symbol of Δ and allows the use of the Kleene's plus on expressions E^+ as well as arbitrary intervals on symbols $a^{[n;m]}$. Note that this also enables the use of the wildcard $E^?$ since it can be defined using ϵ and the union operator without repeating any symbol of Δ.

▶ **Theorem 5.** MEMB(SORBE) *is in PTIME.*

Proof. We fix a bag of symbols w over Δ. For a regular bag expression E, by $\Delta(E)$ we denote the subset of Δ containing exactly the symbols used in E. For a subset $X \subseteq \Delta$ by w_X we denote the bag over X obtained from w by removing all occurrences of symbols outside of X. W.l.o.g. we assume that the Kleene's plus E^+ is used only if $\varepsilon \notin L(E)$ (otherwise E^+ can be replaced by E^*).

The algorithm recursively constructs for an expression E a set of integers $I(E)$ such that $i \in I(E)$ iff $w_{\Delta(E)} \in L(E)^i$. This set is represented by an interval. Recall that an *interval* $[n; m]$ is a finite representation of the set $\{i \mid n \leq i \leq m\}$. It is *empty* if $m < n$ and we use $\emptyset$ to denote (the equivalence class of) all empty intervals. Also, the intersection of two intervals can be obtained easily $[n_1; m_1] \cap [n_2; m_2] = [\max\{n_1, n_2\}; \min\{m_1, m_2\}]$ and the *component-wise addition* $A \oplus B = \{a + b \mid a \in A,\ b \in B\}$ can be implemented as $[n_1; m_1] \oplus [n_2; m_2] = [n_1 + n_2; m_1 + m_2]$ with $m + \infty = \infty + m = \infty$ for any $m \in \mathbb{N} \cup \{\infty\}$. The algorithm is presented on Figure 4 (with $0/\infty = 0$ and $i/\infty = 1$ for $i \geq 1$).

Note that assigning $I(E^+) = [0; 0]$ when $w_{\Delta(E)} = \varepsilon$ is valid since we assume $\varepsilon \notin L(E)$. Naturally, $w \in L(E)$ if and only if $1 \in I(E)$ and w uses only symbols present in E. ◀

We, therefore, immediately get

▶ **Corollary 6.** *Multi-type validation for deterministic* ShEx *using* SORBE *is in PTIME.*

We employ the single type requirement to reduce the NP-complete problem of exact set cover to single-type validation against a deterministic ShEx using only single-occurrence RBE_0 expressions.

$$
\begin{aligned}
&I(\epsilon) = [0; \infty],\\
&I(a^{[n,m]}) = \left[\lceil w(a)/m \rceil; \lfloor w(a)/n \rfloor\right],\\
&I(E_1 \mid E_2) \;=\; I(E_1) \oplus I(E_2),\\
&I(E_1 \parallel E_2) = I(E_1) \cap I(E_2),\\
&I(E^*) = \begin{cases} [0;\infty] & \text{if } w_{\Delta(E)} = \varepsilon,\\ [1;\infty] & \text{if } w_{\Delta(E)} \neq \varepsilon \text{ and } I(E) \neq \emptyset,\\ \emptyset & \text{otherwise,} \end{cases}\\
&I(E^+) = \begin{cases} [0;0] & \text{if } w_{\Delta(E)} = \varepsilon,\\ [1;\max I(E)] & \text{if } w_{\Delta(E)} \neq \varepsilon \text{ and } I(E) \neq \emptyset,\\ \emptyset & \text{otherwise.} \end{cases}
\end{aligned}
$$

Figure 4 Computing the interval $I(E)$ for a SORBE E (Theorem 5).

▶ **Theorem 7.** *Single-type validation for deterministic* ShEx *using* SORBE *is NP-complete.*

5.3 Optimal validation algorithm

Some applications might not require testing validity of the whole graph, but rather checking the validity of only a fragment that will be accessed by the application. Such a fragment can be identified by a set of root nodes, entry points for navigating the graph, and typically, the application will require the entry points to satisfy certain types. In this section, we show how this scenario can be modeled with ShEx and present an efficient algorithm that works with deterministic shape expressions.

For this, take a schema $S = (\Sigma, \Gamma, \delta)$ such that Γ contains a special *universal type* $t_\top$ with the definition $\delta(t_\top) = (\Sigma \times \Gamma)^*$. The language of S is the universal graph language, as any node of any graph can be typed with $t_\top$. In essence, the universal type allows to forgo validation of a node because all nodes implicitly satisfy $t_\top$. For instance, the rule $t_0 \to a :: t_1^* \parallel (\|_{b \in \Sigma, b \neq a} b :: t_\top)$ indicates that a node is valid for type t_0 if all its neighbour nodes reachable by an a-labelled edge are valid for type t_1, but no constraint is imposed to nodes reachable by labels other than a. Therefore, all such (non-a-label neighbour) nodes do not need to be visited by the validation algorithm.

To carry out validation on a fragment of a graph identified by the entry points, we introduce the notion of pre-typing, an assignment of required types to a selected set of nodes. Formally, a *pre-typing* of a graph G (w.r.t. S) is a partial mapping $\lambda\_ : V \to 2^\Gamma$. Now, the objective is to find a *valid extension* of $\lambda\_$ i.e., a valid m-typing λ of G w.r.t. S such that $\lambda\_ \sqsubseteq \lambda$. Since we are not interested in typing the whole graph G, we focus on the smallest possible valid extension of a given pre-typing λ. Interestingly, we can show the following.

▶ **Lemma 8.** *For a deterministic* ShEx $S = (\Sigma, \Gamma, \delta)$ *with universal type, a graph* $G = (V, E)$*, and a pre-typing* $\lambda\_ : V \to 2^\Gamma$*, if* $\lambda\_$ *admits a valid extension, then it admits a unique* $\sqsubseteq$*-minimal valid extension.*

We present an algorithm that constructs the minimal valid extension of a given pre-typing $\lambda\_$ of a given graph G w.r.t. a given deterministic Shape Expression Schema S with universal type. For technical reasons, we represent a typing as binary relation between the set of

Algorithm 1 MinValidExt$(S, G, \lambda_)$

Input: $S = (\Sigma, \Gamma, \delta)$ a deterministic ShEx,
$G = (V, E)$,
$\lambda\_ \subseteq V \times \Gamma$ a pre-typing;
Output: $\lambda \subseteq V \times \Gamma$ the minimal valid extension of $\lambda\_$.

```
1:  let F := λ_
2:  let λ := ∅
3:  while F ≠ ∅ do
4:      choose (n, t) ∈ F and remove it from F
5:      let out-lab-type^δ_G(n, t) := {|(a, δ(t, a)) | (a, m) ∈ out-lab-node_G(n)|}
6:      if out-lab-type^δ_G(n, t) ∉ δ(t) then
7:          fail
8:      λ := λ ∪ {(n, t)}
9:      for (a, m) ∈ out-lab-node_G(n) do
10:         if δ(t, a) ≠ t_⊤ and (m, δ(t, a)) ∉ λ then
11:             F := F ∪ {(m, δ(t, a))}
12: return λ
```

nodes and the set of types, and deliberately omit the universal type. More precisely, we use a relation $R_\lambda \subseteq V \times (\Gamma \setminus \{t_\top\})$ to represent the typing $\lambda(n) = \{t \mid (n, t) \in R_f\}$ if $(n, t) \in R_f$ for some $t \in \Gamma$, and $\lambda(n) = \{t_\top\}$ otherwise. Furthermore, we abuse notation and use λ instead of R_λ. Recall that $\delta(t, a)$ is the unique type used together with the symbol a in $\delta(t)$.

This algorithm is a modified graph flooding algorithm that maintains a frontier set F of pairs (n, t) for which it remains to be verified that the node n satisfies type t. Initially, this set contains only the pairs specified by the pre-typing (line 1). The algorithm fails whenever for some $(n, t) \in F$ the outgoing edges of n do not satisfy the structural constraints given by $\delta(t)$ (lines 5–7). If, however, the constraints are satisfied, any node m reachable from n is added to F with an appropriate type unless the type is universal.

Note that a run of the algorithm considers the pair (n, t) at most once, and therefore the main loop is executed at most $|V| \times |\Gamma|$ times. Once F is empty, the constructed λ represents the *minimal valid extension* of $\lambda\_$. This algorithm is optimal in the sense that it constructs the minimal representation of the minimal valid extension and considers assigning a type to a node only if it is required to construct the extension. Naturally, for single occurrence RBEs the algorithm works in polynomial time.

▶ **Theorem 9.** *Given a deterministic* ShEx(SORBE) S*, a graph* G*, and a pre-typing* $\lambda\_ : V \to \Gamma$*, the algorithm* MinValidExt$(S, G, \lambda_)$ *constructs in polynomial time the minimal valid extension of* $\lambda\_$ *if it exists, or fails otherwise.*

A slight modification of this algorithm works for the single-type semantics too: in the inner loop (lines 9–11) it suffices to add a check that neither F nor λ contain (m, t') for some $t' \neq \delta(t, a)$, which prevents assigning two different types to the same node. As a result such modified algorithm constructs an s-typing λ (with universal type omitted). Also, note that with the single-type modification the while loop is executed at most $|V|$ times, and the algorithm considers each edge of the graph at most once, i.e. the algorithm makes a single pass over the graph.

Figure 5 Fork and diamond graphs.

6 Expressive power

This section outlines the results of our study of expressive power of ShEx. First, we consider the first-order logic on graphs ($\mathsf{FO_G}$) over the standard signature consisting of relation names $(E_a)_{a\in\Sigma}$, and the existential monadic second-order logic on graphs ($\exists\mathsf{MSO_G}$) allowing only formulas of the form $\exists X_1, \ldots, X_n \varphi$, where $X_1, \ldots, X_n$ are monadic second order variables and φ is an FO formula using additional atomic formulae of the form $x \in X_i$.

We say that a class of graph languages $\mathcal{C}$ *separates* a graph H from a graph G if there is $L \in \mathcal{C}$ such that $G \in L$ and $H \notin L$. Consider the fork $G_<$ and the diamond $G_\diamond$ graphs in Figure 5.

We observe that single-type semantics can easily separate $G_\diamond$ from $G_<$ while it can be easily shown that the multi-type semantics cannot. However, even the single-type semantics cannot separate $G_<$ from $G_\diamond$ while this separation is trivial for $\mathsf{FO_G}$ and $\exists\mathsf{MSO_G}$. Also, let L_{cycle} be the set of graphs labeled with $\Sigma = \{a, b\}$ such that for every node n with an incoming b-edge, there is a cycle reachable from n. It is a classic result that $\mathsf{FO_G}$ cannot define sets of graphs which contain cycles of unbounded size [14]. However, L_{cycle} can be defined in both semantics with the following schema:

$$S_{cycle} : t_0 \to (a :: t_\bullet \mid a :: t_0)^* \parallel (b :: t_\bullet)^* \qquad t_\bullet \to (a :: t_\bullet^+ \mid b :: t_\bullet) \parallel (a :: t_0)^* \parallel (b :: t_\bullet)^*$$

The following table present a comparison of expressive power, indicating whether a language L satisfying the constraints given in the first column can be expressed by each of the formalisms.

	$\mathsf{FO_G}$	$\mathsf{ShEx_m}$	$\mathsf{ShEx_s}$	$\exists\mathsf{MSO_G}$
$L : G_\diamond \in L,\ G_< \in L$	✓	✓	✓	✓
$L : G_\diamond \notin L,\ G_< \in L$	✓	×	✓	✓
$L : G_\diamond \in L,\ G_< \notin L$	✓	×	×	✓
L_{cycle}	×	✓	✓	✓

It should also be noted that RBEs can express cardinality constraints e.g., $(a \parallel b)^*$ means that the number of a must be equal to the number of b, that cannot be captured by $\exists\mathsf{MSO_G}$. However, if we limit the expressive power of the bag languages used in schemas to those that can be captured by $\exists\mathsf{MSO_G}$, then the expressive power of schemas is captured by $\exists\mathsf{MSO_G}$, a result easily proven with a simple adaptation of the standard translation of an automaton to an existential monadic second-order formula [38]. For instance, the class $\mathsf{RBE}(a^{[n;m]}, \|, |)$ of bag languages definable by RBE without the Kleene star operator but allowing arbitrary intervals of symbols, can be captured by $\exists\mathsf{MSO_G}$. The restriction on the use of the Kleene closure in defining unordered content models has been previously advocated for complexity reasons in the context of XML [11], and we provide yet another reason.

The single-type semantics is in fact very powerful and can easily capture graph languages defined by homomorphism into a fixed graph, as illustrated in the proof of Theorem 3. It seems unlikely that the multi-type semantics is as powerful as suggested by complexity arguments in the present paper. Both semantics have the same closure properties:

Table 1 Summary of main complexity results for the validation problem.

	RBE_0	RBE	SORBE	SORBE det.	SORBE det. + $\lambda\_$ + $t_\top$
multi-type	PTIME (Thm. 3)	NP-c. (Thm. 4)		PTIME (Cor. 6 and Thm. 9)	
single-type	NP-c. (Thm. 1)		NP-c. (Thm. 7)	PTIME (Section 5.3)	

▶ **Theorem 10.** ShEx_s *and* ShEx_m *are not closed under union and complement. Both* ShEx_s *and* ShEx_m *are closed under intersection.*

The closure under intersection can be extended to a powerset technique, similar to the determinisation technique of finite automata [21], that allows to show that the single-type semantics is in fact more powerful than the multi-type semantics.

▶ **Theorem 11.** ShEx_s *properly contains* ShEx_m.

ShEx can be viewed as an automaton on edge-labeled (possibly infinite) trees obtained by unraveling the input graph and we believe that such a model would correspond closely to Presburger automata [33] if the latter were extended to infinite trees, taking a universal acceptance condition. Interestingly, in this analogy the single-type semantics corresponds to deterministic automata while multi-type semantics corresponds to nondeterministic ones. More recently, k-Pebble automata on graphs have been proposed [31] but they are not comparable with ShEx because the are capable of expressing arbitrary FO properties. Recognizable sets of graphs as defined in [12] go beyond MSO_G, and therefore, capture $\mathsf{ShEx}(\mathsf{RBE}(a^{[n;m]}, \|, |))$.

Finally, ShEx are incomparable with both the node replacement (NR) graph grammars, and the hyperedge replacement (HR) graph grammars. On the one hand, the language $\{G_\diamond\}$ is definable by both HR and NR graph grammars with single initial graph and no rules. On the other hand, ShEx can define languages that are not definable by neither HR nor NR grammars: HR grammars can define only languages of graphs of bounded tree-width while NR grammars cannot define a language containing infinitely many square grids.

7 Conclusions

We have investigated Shape Expressions Schemas (ShEx), a novel formalism of schemas for RDF graphs currently under development by W3C. We have proposed two alternative semantics, single-type and multi-type, studied their expressive power and the complexity of the problem of validation. We have also proposed a notion of determinism in order to curb down the complexity of validation. While the single-type semantics is in general intractable, for multi-type validation we have identified two essential bottleneck complexity problems on RBE, membership and satisfiability of RBEs with intersection, depending on whether or not deterministic expressions are used. Summary of complexity results can be found in Table 1.

Our results on expressive power suggest that an unrestricted use of the Kleene closure may render the proposed formalism too powerful and so far there exists little evidence of its practical usability in the context of unordered content model [11, 6]. Complexity results suggest that the single-type semantics may be too expensive for practical application unless we wish to validate only a fragment of graph with a given pretyping. As for the multi-type semantics, validation is tractable for a small yet practical fragment RBE_0 and if we use determinism, a richer class of SORBEs can be handled efficiently.

Future work. In the future, we plan to investigate the impact of data value constraints on complexity of validation. Our preliminary study shows that adding *local* data value constraints, such as domain check, does not affect our results. However, the impact of value constraints of *global* nature, such as key dependencies, remains to be investigated. We also plan to thoroughly evaluate experimentally the proposed algorithms and compare with existing validation approaches (SPIN, ICV) on both real-life and synthetically generated data e.g., RDF export of TPC-H benchmark data [39]. Our preliminary experiments [7] are very promising. Also, we would like to study the complexity of classical static analysis problems such as schema containment and query validity in the presence of schema. Finally, we would like devise inference algorithms for ShEx drawing inspiration from learning XML twig queries [36] and schemas for XML [4, 3, 9].

References

1 M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In *Reasoning Web*, Int'l Summer School on Semantic Technologies for Information Systems, pages 158–204, 2009. Invited Tutorial.

2 M. Arenas, J. Pérez, Reutter J., C. Riveros, and J. Sequeda. Data exchange in the relational and RDF worlds. Invited talk at the Int'l Workshop on Semantic Web Information Management (SWIM), June 2011.

3 G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems*, 35(2), 2010.

4 G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *Int'l Conf. on Very Large Data Bases (VLDB)*, pages 998–1009, 2007.

5 J. Bolleman, S. Gehant, and N. Redaschi. Catching inconsistencies with the semantic web: A biocuration case study. In *Int'l Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS)*, 2012.

6 I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theory of Computing Systems*, 2014. To appear. Available at `http://arxiv.org/abs/1311.7307`.

7 I. Boneva, J. Emilio Labra Gayo, S. Hym, E. G. Prud'hommeau, H. Solbrig, and S. Staworko. Validating RDF with shape expressions, April 2014. Available at `http://arxiv.org/abs/1404.1270`.

8 D. Brickley and R. V. Guha. RDF Schema 1.1. `http://www.w3.org/TR/rdf-schema`, February 2004.

9 R. Ciucanu and S. Staworko. Learning schemas for unordered XML. In *Int'l Symp. on Database Programming Languages (DBPL)*, 2013.

10 D. Colazzo, G. Ghelli, L. Pardini, and C. Sartiani. Linear inclusion for XML regular expression types. In *Int'l Conf. on Information and Knowledge Management (CIKM)*, pages 137–146, 2009.

11 D. Colazzo, G. Ghelli, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. *Information Systems*, 34(7):643–656, 2009.

12 B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

13 M. Dean and M. Schreiber. OWL Web Ontology Language Reference. `http://www.w3.org/TR/owl-ref`, February 2004.

14 H.-D. Ebbinghaus and J. Flum. *Finite model theory*. Springer, 1995.

15 J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and A. Arias. Binary RDF representation for publication and exchange (HDT). *J. Web Semantics*, 19:22–41, 2013.

16 G. Ghelli, D. Colazzo, and C. Sartiani. Linear time membership in a class of regular expressions with interleaving and counting. In *Int'l Conf. on Information and Knowledge Management (CIKM)*, pages 389–398, 2008.

17 S. Ginsburg and Spanier E. H. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, December 1966.

18 B. Glimm and O. Chimezie. SPARQL 1.1 Entailment Regimes. `http://www.w3.org/TR/sparql11-entailment/`, 2012.

19 A. V. Goldberg, E. Tardos, and R. E. Tarjan. Network flow algorithms. In *Algorithms and Complexity*, Volume 9, *Paths, Flows, and VLSI-Layout*, 1990.

20 B. Groz, S. Maneth, and S. Staworko. Deterministic regular expressions in linear time. In *ACM Symp. on Principles of Database Systems (PODS)*, May 2012.

21 J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2nd edition, 2001.

22 E. Kopczynski and A. To. Parikh images of grammars: Complexity and applications. In *LICS*, pages 80–89, 2010.

23 D. Kozen. Lower bounds for natural proof systems. In *IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 254–266, 1977.

24 J. E. Labra Gayo, E. Prud'hommeaux, H. Solbrig, and J. M. Álvarez Rodríguez. Validating and describing linked data portals using RDF Shape Expressions. In *Workshop on Linked Data Quality*, September 2015.

25 M. Montazerian, P. T. Wood, and S. R. Mousavi. XPath query satisfiability is in PTIME for real-world DTDs. In *Int'l XML Database Symp. (Xsym)*, pages 17–30, 2007.

26 M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.

27 D. C. O. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.

28 C. H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, October 1981.

29 R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

30 E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig. Shape Expressions: An RDF validation and transformation language. In *Int'l Conf. on Semantic Systems*, Sep. 2015.

31 J. L. Reutter and T. Tan. A formalism for graph databases and its model of computation. In *AMW*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

32 A. Ryman, A. Le Hors, and S. Speicher. Oslc resource shape: A language for defining constraints on linked data. In *Proc. of the WWW2013 Workshop on Linked Data on the Web (LDOW)*. CEUR-WS.org, 2013.

33 H. Seidl, T. Schwentick, and A. Muscholl. Counting in trees. *Logic and Automata*, pages 575–612, 2008.

34 J. Sequeda, H. Tirmizi, S, Ó. Corcho, and D. P. Miranker. Survey of directly mapping SQL databases to the Semantic Web. *Knowledge Engineering Review*, 26(4):445–486, 2011.

35 E. Sirin. Data validation with OWL integrity constraints. In *Int'l Conf. on Web Reasoning and Rule Systems (RR)*, pages 18–22, 2010.

36 S. Staworko and P. Wieczorek. Learning twig and path queries. In *Int'l Conf. on Database Theory (ICDT)*, March 2012.

37 J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in OWL. In *Int'l Conf. on Artificial Intelligence (AAAI)*, 2010.

38 J. W. Thatcher and Wright J. B. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.

39 TPC. TPC benchmarks, `http://www.tpc.org/`.

40 W3C. RDF validation workshop report: Practical assurances for quality RDF data. `http://www.w3.org/2012/12/rdf-val/report`, September 2013.

41 W3C. Shape expressions schemas, 2013. `http://www.w3.org/2013/ShEx/Primer`.

CONSTRUCT Queries in SPARQL

Egor V. Kostylev[1], Juan L. Reutter[2], and Martín Ugarte[2]

1 **University of Oxford**
egor.kostylev@cs.ox.ac.uk
2 **PUC Chile**
jreutter@ing.puc.cl, martinugarte@puc.cl

Abstract

SPARQL has become the most popular language for querying RDF datasets, the standard data model for representing information in the Web. This query language has received a good deal of attention in the last few years: two versions of W3C standards have been issued, several SPARQL query engines have been deployed, and important theoretical foundations have been laid. However, many fundamental aspects of SPARQL queries are not yet fully understood. To this end, it is crucial to understand the correspondence between SPARQL and well-developed frameworks like relational algebra or first order logic. But one of the main obstacles on the way to such understanding is the fact that the well-studied fragments of SPARQL do not produce RDF as output.

In this paper we embarrk on the study of SPARQL CONSTRUCT queries, that is, queries which output RDF graphs. This class of queries takes rightful place in the standards and implementations, but contrary to SELECT queries, it has not yet attracted a worth-while theoretical research. Under this framework we are able to establish a strong connection between SPARQL and well-known logical and database formalisms. In particular, the fragment which does not allow for blank nodes in output templates corresponds to first order queries, its well-designed sub-fragment corresponds to positive first order queries, and the general language can be restated as a data exchange setting. These correspondences allow us to conclude that the general language is not composable, but the aforementioned blank-free fragments are. Finally, we enrich SPARQL with a recursion operator and establish fundamental properties of this extension.

1998 ACM Subject Classification H.2.3 Languages – Query languages

Keywords and phrases RDF, SPARQL, Query Languages

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.212

1 Introduction

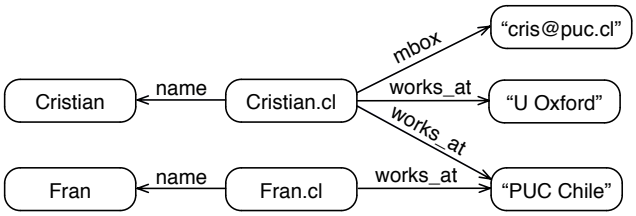
The Resource Description Framework (RDF) [25] is the World Wide Web consortium (W3C) standard for representing linked data on the Web. Intuitively, an RDF graph is a set of triples of internationalized resource identifiers (IRIs), where the first and last IRI in the triples represent entity resources, and the middle one relates these resources.

SPARQL is a language for querying RDF datasets. Originally introduced in 2006 [33], SPARQL was officially made the recommended language to query RDF data by W3C in 2008 [32]. A recent version of the standard, denoted SPARQL 1.1, was issued in 2013 [39]. Nowadays this language is recognised as one of the key standards of the Semantic Web initiative and there are several SPARQL engines available to industry (e.g., [12, 18, 37]).

The theoretical foundations of SPARQL were laid by Pérez et al. in their seminal work [27], and a body of research has followed covering a variety of issues such as complexity of query evaluation [4, 24, 29, 36], query optimisation [8, 9, 22, 30], federation [7], expressive power

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 212–229

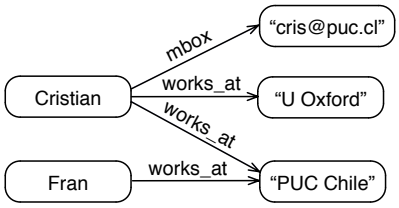
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a)

	?n	?w	?e
μ_1	Fran	PUC Chile	
μ_2	Cristian	U Oxford	cris@puc.cl
μ_3	Cristian	PUC Chile	cris@puc.cl

(b)



(c)

Figure 1 (a) RDF graph G_{ex}; (b) answer of $\boldsymbol{q}_{\text{sel}}$ over G_{ex} is the set of mappings $\{\mu_1, \mu_2, \mu_3\}$; (c) answer of $\boldsymbol{q}_{\text{cons}}$ over G_{ex} is RDF graph.

[2, 31], and provenance tracking [15, 17]. The impact of these studies in the Semantic Web community has been astonishing, even influencing in the definition of the SPARQL standards.

Despite the key importance of SPARQL, the fundamental aspects of this language are still not fully understood. Compared to the knowledge we have on other query languages such as SQL, Datalog or even XPath, very little is known about SPARQL queries. To this end, it is of particular importance to understand the correspondence between SPARQL and other well-developed formalisms such as first order logic or relational algebra. One of the main obstacles on the way to such understanding is the fact that the queries from well-studied fragments of SPARQL produce not RDF graphs as answers, but sets of mappings (partial evaluations), which is a different form for representing data.

▶ **Example 1.** As a classical example of SPARQL, let us consider the following query $\boldsymbol{q}_{\text{sel}}$:[1]

```
SELECT ?n, ?w, ?e
WHERE (
((?p, name, ?n) AND (?p, works_at, ?w))
         OPT   (?p, mbox, ?e)).
```

This query is intended to extract all names and affiliations of people for which a working place is known, appending their emails when available in the RDF graph. Thus, when evaluated on the RDF graph G_{ex} from Figure 1(a), it gives as result a set of partial mappings from the variables of $\boldsymbol{q}_{\text{sel}}$ to IRIs in the RDF graph, as depicted in Figure 1(b), where each row represents a mapping.

Returning mappings instead of tuples might appear just as a slight difference between SPARQL and other query languages such as SQL, but it is known to lead to several complications (see, e.g., [27, 31]). For example, when studying the expressive power of SPARQL in [2, 31], the authors need some rather technical machinery to be able to even compare SPARQL with relational query languages. The result is that, even if we now know

[1] In this paper we follow the SPARQL syntax of [27], in particular, we shorten `OPTIONAL` to OPT.

that the SELECT fragment of SPARQL is equivalent in expressive power to relational algebra, this is shown using proofs that are much more complicated than other similar results in database theory, and it has been difficult to build upon this proofs to produce new results.

There are also practical consequences: while recursive queries have been part of SQL for more than twenty years, we are still left without a comprehensive operator to define recursive queries in SPARQL (SPARQL 1.1 includes the property paths primitive [39], but this additional feature is very restrictive in expressing recursive queries [23]).

However, this complication is relevant only to the SELECT queries of SPARQL, which have been considered in the theoretical literature almost exclusively. But there is also a class of queries that output RDF graphs, namely the class of CONSTRUCT queries. The following example illustrates how a user can specify such a query.

▶ **Example 2.** Let $\boldsymbol{q}_{\text{cons}}$ be the following SPARQL CONSTRUCT query:

```
CONSTRUCT {(?n, works_at, ?w), (?n, mbox, ?e)}
WHERE (
((?p, name, ?n) AND (?p, works_at, ?w))
         OPT   (?p, mbox, ?e)).
```

This query has the same WHERE clause as $\boldsymbol{q}_{\text{sel}}$, but the form of the output is different. The RDF graph resulting from the evaluation of this query over the dataset G_{ex} is depicted in Figure 1(c).

CONSTRUCT queries in SPARQL shape the class of effective queries whose inputs and answers are RDF graphs, so it is conceivable that much more insight can be obtained by comparing them to well-established query languages. But rather surprisingly, and despite being an important part of the SPARQL standard, these queries have received almost no theoretical attention. This can be partially explained by the fact that, as the examples above suggest, the difference between these classes of queries might seem negligible. However, as we show in this paper, this resemblance is often deceptive, and in many cases the properties of these queries are different. For example, CONSTRUCT queries allow for blank nodes in the templates specifying the answer triples, which is a feature unavailable in SELECT queries. Trying to fill this gap, we conduct a thorough study of CONSTRUCT queries. We concentrate on the AND-UNION-OPT-FILTER fragment, which is the core of SPARQL [27].

The first question studied in the paper is the expressive power of CONSTRUCT queries. In particular, we show that if blank nodes are not allowed in the templates, then this language is equivalent in expressive power to first order logic. Furthermore, if the underlying graph patterns are enforced to belong to the class of well designed patterns (see [27]) then we obtain a correspondence with positive first order logic. If, in turn, blank nodes in templates are allowed, we establish that the expressive power of these queries is equivalent to that of a well known class of mappings in data exchange.

These expressivity results lead to important conclusions on the composability of the aforementioned classes of queries, that is, whether the composition of two queries can always be expressed by another query in the same class. We show that the fragments without blank nodes are composable, but if blank nodes are allowed in construct templates then this important property is lost.

We also obtain results on the computational complexity of the evaluation of such queries: for the blank-free language it is the same as for SELECT queries (PSPACE-complete), but for the well-designed sublanguage there is a difference – it is Σ_2^p-complete for the SELECT case ([22]), but drops to NP-complete in CONSTRUCT case.

Finally, the properties of CONSTRUCT queries allow us to develop an extension of SPARQL with a form of recursion that resembles that of SQL. This proposal unifies several formalisms for querying RDF data such as SPARQL 1.1 property paths [39], c-query answering over OWL 2 RL entailment regime [16, 20], navigational SPARQL [28], GraphLog [11], and TriAL [23]. We are also able to pinpoint the expressivity of this extension to SPARQL by comparing it with a fragment of Datalog.

Due to the space limitations, only ideas of most important proofs are exposed in the main body of this paper. Complete proofs shall be given in the full version of this paper.

2 Preliminaries

RDF Graphs and Datasets

RDF graphs can be seen as edge-labeled graphs where edge labels can be node themselves, and an RDF dataset is a collection of RDF graphs. Formally, let $\mathbf{I}$ and $\mathbf{B}$ be infinite pairwise disjoint sets of *IRIs* and *blank nodes*,[2] respectively, and $\mathbf{T} = \mathbf{I} \cup \mathbf{B}$ be the set of *terms*. Then an *RDF triple* is a tuple (s, p, o) from $\mathbf{T} \times \mathbf{I} \times \mathbf{T}$, where s is called the *subject*, p the *predicate*, and o the *object*. An *RDF graph* is a finite set of RDF triples, and an *RDF dataset* is a set $\{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$, where $G_0, \ldots, G_n$ are RDF graphs and $u_1, \ldots, u_n$ are distinct IRIs, such that the graphs G_i use pairwise disjoint sets of blank nodes. The graph G_0 is called *default graph*, and $G_1, \ldots, G_n$ are called *named graphs* with *names* $u_1, \ldots, u_n$, respectively. For a dataset D and IRI u we define $\mathsf{gr}_D(u) = G$ if $\langle u, G\rangle \in D$ and $\mathsf{gr}_D(u) = \emptyset$ otherwise. We also use $\mathcal{G}$ and $\mathcal{D}$ to denote the sets of all RDF graphs and datasets, correspondingly, as well as $\mathsf{blank}(S)$ to denote the set of blank nodes appearing in S, which can be a triple, a graph, etc.

SPARQL Syntax

SPARQL is the standard pattern-matching language for querying RDF datasets. Let $\mathbf{V}$ be an infinite set $\{?x, ?y, \ldots\}$ of *variables*, disjoint from $\mathbf{T}$. Similarly to $\mathsf{blank}(S)$, let $\mathsf{var}(S)$ denote the set of variables appearing in S. SPARQL *graph patterns* are recursively defined as follows:

1. a triple in $(\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V})$ is a graph pattern, called a *triple pattern*;
2. if P_1 and P_2 are graph patterns then $(P_1 \,\mathsf{AND}\, P_2)$, $(P_1 \,\mathsf{OPT}\, P_2)$, and $(P_1 \,\mathsf{UNION}\, P_2)$ are graph patterns, called AND-, OPT-, and UNION-*patterns*, correspondingly;
3. if P is a graph pattern and $g \in \mathbf{I} \cup \mathbf{V}$ then $(g \,\mathsf{GRAPH}\, P)$ is a graph pattern, called a GRAPH-*pattern*;
4. if P is a graph pattern and R is a filter condition then $(P \,\mathsf{FILTER}\, R)$ is a graph pattern, called a FILTER-*pattern*, where SPARQL *filter conditions* are constraints of the form:
   - $?x = u$, $?x = ?y$, $isBlank(?x)$ or $bound(?x)$ for $?x, ?y \in \mathbf{V}$ and $u \in \mathbf{I}$ (called *atomic constraints*[3]),
   - $\neg R$, $R_1 \wedge R_2$, or $R_1 \vee R_2$, for filter conditions R, R_1 and R_2.

The fragment of SPARQL graph patterns, as well as its generalisation to SELECT queries, has drawn most of the attention in the Semantic Web community. In this paper we concentrate on another class of queries, formalized next.

[2] For the sake of simplicity we do not consider literals, but all the results in this paper hold if we introduce them explicitly.

[3] We use a simplified list of SPARQL atomic constraints, for the complete one see [39].

A SPARQL CONSTRUCT query, or *c-query* for short, is an expression

$$\mathsf{CONSTRUCT}\ H\ \mathsf{WHERE}\ P,$$

where H is a set of triples from $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$, called a *template*, and P is a graph pattern. We also distinguish c-queries without blank nodes in templates, called *blank-free*, and c-queries without GRAPH-subpatterns in their patterns, called *graph-free*. We use c-SPARQL to denote the class of all c-queries, and specify these restrictions with subscripts bf and gf for the blank- and graph-free subclasses. For instance, c-SPARQL$_{\mathsf{bf,gf}}$ denotes the class of blank-free and graph-free c-queries.

SPARQL Semantics

The semantics of graph patterns is defined in terms of *mappings*; that is, partial functions from variables $\mathbf{V}$ to terms $\mathbf{T}$. The *domain* $\mathsf{dom}(\mu)$ of a mapping μ is the set of variables on which μ is defined. Two mappings μ_1 and μ_2 are *compatible* (written as $\mu_1 \sim \mu_2$) if $\mu_1(?x) = \mu_2(?x)$ for all variables $?x$ in $\mathsf{dom}(\mu_1) \cap \mathsf{dom}(\mu_2)$. If $\mu_1 \sim \mu_2$, then we write $\mu_1 \cup \mu_2$ for the mapping obtained by extending μ_1 according to μ_2 on all the variables in $\mathsf{dom}(\mu_2) \setminus \mathsf{dom}(\mu_1)$.

Given two sets of mappings M_1 and M_2, the *join*, *union* and *difference* between M_1 and M_2 are defined respectively as follows:

$$\begin{array}{lcl} M_1 \bowtie M_2 & = & \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1,\ \mu_2 \in M_2 \text{ and } \mu_1 \sim \mu_2\}, \\ M_1 \cup M_2 & = & \{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\}, \\ M_1 \setminus M_2 & = & \{\mu_1 \mid \mu_1 \in M_1 \text{ and there is no } \mu_2 \in M_2 \text{ such that } \mu_1 \sim \mu_2\}. \end{array}$$

Based on these, the *left outer join* operation is defined as

$$M_1 \mathbin{⟕} M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2).$$

Given a dataset $D = \{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$, and a graph G among $G_0, \ldots, G_n$, the *evaluation* $[\![P]\!]^D_G$ of a graph pattern P over D with respect to G is defined as follows:

1. if P is a triple pattern, then $[\![P]\!]^D_G = \{\mu : \mathsf{var}(P) \to \mathbf{T} \mid \mu(P) \in G\}$;
2. if $P = (P_1\ \mathsf{AND}\ P_2)$, then $[\![P]\!]^D_G = [\![P_1]\!]^D_G \bowtie [\![P_2]\!]^D_G$;
3. if $P = (P_1\ \mathsf{OPT}\ P_2)$, then $[\![P]\!]^D_G = [\![P_1]\!]^D_G \mathbin{⟕} [\![P_2]\!]^D_G$;
4. if $P = (P_1\ \mathsf{UNION}\ P_2)$, then $[\![P]\!]^D_G = [\![P_1]\!]^D_G \cup [\![P_2]\!]^D_G$;
5. if $P = (g\ \mathsf{GRAPH}\ P')$, then

$$[\![P]\!]^D_G = \begin{cases} [\![P']\!]^D_{\mathsf{gr}_D(g)} & \text{if } g \in \mathbf{I} \\ \bigcup_{u \in \mathbf{I}} \left([\![P']\!]^D_{\mathsf{gr}_D(u)} \bowtie \{\mu_{g \mapsto u}\} \right) & \text{if } g \in \mathbf{V} \end{cases}$$

   where $\mu_{g \mapsto u}$ is the mapping with domain $\{g\}$ and where $\mu_{g \mapsto u}(g) = u$;
6. if $P = (P'\ \mathsf{FILTER}\ R)$, then $[\![P]\!]^D_G = \{\mu \mid \mu \in [\![P']\!]^D_G \text{ and } \mu \models R\}$, where a mapping μ *satisfies* a built-in condition R, denoted by $\mu \models R$, if one of the following holds:
   - R is $?x = u$, $?x \in \mathsf{dom}(\mu)$ and $\mu(?x) = u$; or
   - R is $?x = ?y$, $?x \in \mathsf{dom}(\mu)$, $?y \in \mathsf{dom}(\mu)$ and $\mu(?x) = \mu(?y)$; or
   - R is $isBlank(?x)$ and $?x \in \mathsf{dom}(\mu)$ and $\mu(?x) \in \mathbf{B}$; or
   - R is $bound(?x)$ and $?x \in \mathsf{dom}(\mu)$; or
   - R is a Boolean combination of other filter conditions and this combination is satisfied according to the usual notions of $\{\neg, \vee, \wedge\}$.

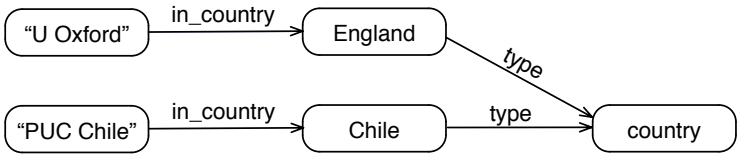


Figure 2 RDF graph containing information about location of universities.

The evaluation $[\![P]\!]^D$ of a pattern P over a dataset D with default graph G_0 is $[\![P]\!]^D_{G_0}$.

Next we define the semantics of c-queries. We concentrate for now on the class c-SPARQL$_{\mathsf{bf}}$ of queries, and discuss the semantics for full c-SPARQL in Section 5. The answer $\mathsf{ans}(\boldsymbol{q}, D)$ of a c-query $\boldsymbol{q} = \mathsf{CONSTRUCT}\ H\ \mathsf{WHERE}\ P$ in c-SPARQL$_{\mathsf{bf}}$ over an input dataset D is defined as

$$\mathsf{ans}(\boldsymbol{q}, D) = \{\mu(t) \mid \mu \in [\![P]\!]^D, t \text{ is a triple in } H \text{ and } \mu(t) \text{ is well-formed}\},$$

Note that the well-formedness condition disallows triples with blank nodes in predicate positions. Next we provide an example to illustrate the use of the operators GRAPH and CONSTRUCT. See [5] for examples on the rest of the operators.

▶ **Example 3.** Let G and G_1 be the graphs depicted in Figure 1(a) and Figure 2, respectively. Suppose we want to query the dataset $D = \{G, \langle \text{country}, G_1 \rangle\}$ to obtain a new graph with information about where workers live. This would be achieved by the next SPARQL CONSTRUCT query:

```
CONSTRUCT {(?name, lives_in, ?country)} WHERE (
(?worker, name, ?name) AND (?worker, works_at, ?university) AND
(country GRAPH (?university, in_country, ?country)) ).
```

3 Blank-free c-Queries

We start our study with c-SPARQL$_{\mathsf{bf}}$, the language of c-queries without blank nodes in their construct templates. This fragment has simple syntax and clear semantics, and it is of fundamental importance in our study. In particular, it resembles SPARQL SELECT queries in the sense that all the blank nodes in the answer graph of a c-SPARQL$_{\mathsf{bf}}$ query already appear in the input dataset.

The first problem we consider is the expressive power of c-SPARQL$_{\mathsf{bf}}$. As usual in databases our yardstick is first order logic (FO) with safe negation. However, since we are dealing with c-queries that input RDF graphs and datasets, it is only fair to compare them with FO over a signature that corresponds to these entities. Formally, we specify the following query language. Consider relational predicates $Default$, $Named$ and $IsBlank$, of arities 3, 4 and 1, respectively. Then the language FO$_{\mathsf{rdf}}$ consists of all well-formed ternary FO formulas over this signature. We always assume that the domain of FO structures is the set $\mathbf{T}$ of terms, and that for all structures we have that $IsBlank(b)$ holds for some b if and only if $b \in \mathbf{B}$, and $IsBlank(b)$ implies that none of $Default(a, b, c)$, $Named(b, a, c, d)$ and $Named(d, a, b, c)$ hold for any a, c, and d. Thus the answers for this language are sets of triples from $\mathbf{T} \times \mathbf{I} \times \mathbf{T}$, essentially RDF graphs. Finally, the evaluation function for FO$_{\mathsf{rdf}}$ is the usual FO entailment $\models_{\text{adom}}$ over active domain semantics. This means that quantification

is realised over the finite set of all the terms from $\mathbf{T}$ appearing in the input database and query (see [1] for formal definitions).

Note that the set of input databases of $\mathrm{FO}_{\mathsf{rdf}}$ have a straightforward one-to-one correspondence with the set of input datasets of c-SPARQL$_{\mathsf{bf}}$ queries, and the same holds for answers of queries in these languages. This allows us to compare their expressive power, for which we need the following definitions. A query language $\mathcal{Q}_1$ is *contained* in a language $\mathcal{Q}_2$ if and only if there are bijections $\mathsf{trans}_{\mathcal{I}} : \mathcal{I}_1 \to \mathcal{I}_2$, $\mathsf{trans}_{\mathcal{O}} : \mathcal{O}_1 \to \mathcal{O}_2$ between their input sets $\mathcal{I}_i$ and answer sets $\mathcal{O}_i$, and a function $\mathsf{trans}_{\mathcal{Q}} : \mathcal{Q}_1 \to \mathcal{Q}_2$ such that $\mathsf{trans}_{\mathcal{O}}(\mathsf{eval}_1(\boldsymbol{q}, I)) = \mathsf{eval}_2(\mathsf{trans}_{\mathcal{Q}}(\boldsymbol{q}), \mathsf{trans}_{\mathcal{I}}(I))$ holds for any $\boldsymbol{q} \in \mathcal{Q}_1$ and $I \in \mathcal{I}_1$, where eval_i are the evaluation functions of the languages. Two languages are *equivalent* if and only if they contain each other.

We are ready to present our first result, claiming that the language of blank-free construct queries is subsumed by first order logic.

▶ **Lemma 4.** *The language* c-SPARQL$_{\mathsf{bf}}$ *is contained in* $\mathrm{FO}_{\mathsf{rdf}}$.

To show this lemma one can use ideas similar to the ones presented in the reductions from the language of SPARQL SELECT queries to non-recursive Datalog with safe negation developed in [2] and [31]. Starting with a query Q, the idea of these reductions is to assemble an extensional predicate for each subpattern of Q in a way such that the evaluation of that predicate contains all the tuples that correspond to a mappings in the evaluation of the subpattern. Since some of the variables of these mappings may not be assigned, the undefined value is modelled by a special constant $Null$. We present a simpler reduction where $Null$ is not used, but instead we create a predicate for each subset of the set of variables of Q. Avoiding predicate $Null$ makes our proof much more simple and intuitive, and we make use of this proof to obtain several results in the following section.

Proof (idea). First we establish an equivalence between graph patterns and $\mathrm{FO}_{\mathsf{rdf}}$. Once this is done we just need to project out those variables that are not on the construct template and generate the corresponding triple. We do it as follows.

Given a graph pattern P, for every $X \subseteq \mathsf{var}(P)$ we construct a formula φ^P_X with X as free variables, such that a mapping μ is in $[\![P]\!]^D$ for a dataset D if and only if the variable assignment defined by μ satisfies $\varphi^P_{\mathsf{dom}(\mu)}$ in the $\mathrm{FO}_{\mathsf{rdf}}$ structure corresponding to D. Having such a formula for each set of variables makes it easier to define an inductive construction. We illustrate this construction with the translation of patterns P of the form $(P_1 \ \mathsf{AND} \ P_2)$. Consider, for every subset X of $\mathsf{var}(P)$, the formula

$$\varphi^P_X = \bigvee_{X_1 \subseteq \mathsf{var}(P_1), X_2 \subseteq \mathsf{var}(P_2), X_1 \cup X_2 = X} \varphi^{P_1}_{X_1} \wedge \varphi^{P_2}_{X_2},$$

where $\varphi^{P_i}_{X_i}$ are the formulas constructed on the previous inductive step.

Finally, the ternary formula $\varphi_{\boldsymbol{q}}$ producing, for every dataset D, the set of triples which correspond to the answer graph to the c-query $\boldsymbol{q} = \mathsf{CONSTRUCT}\ H\ \mathsf{WHERE}\ P$ over D can be simply obtained from all φ^P_X by means of disjunction, existential quantification and checking that all the second arguments are not blank nodes. ◀

We illustrate this proof by means of the following example.

▶ **Example 5.** Recall the query $\boldsymbol{q}_{\mathrm{cons}}$ from Example 2. By simple inspection we see that the domain of every mapping in the evaluation of the graph pattern is either $\{?p, ?n, ?w\}$ or $\{?p, ?n, ?w, ?e\}$. Hence, we only need to construct a formula for each of these sets, as the

formulas corresponding to other subsets of $\mathsf{var}(\boldsymbol{q}_{\text{cons}})$ will be unsatisfiable. Following the construction process, we obtain

$$\begin{aligned}\varphi_{\{?p,?n,?w\}}(p,n,w) &= Default(p,\text{name},n) \wedge Default(p,\text{works\_at},w) \wedge \\ &\qquad \neg\exists e\ Default(p,\text{mbox},e),\\ \varphi_{\{?p,?n,?w,?e\}}(p,n,w,e) &= Default(p,\text{name},n) \wedge Default(p,\text{works\_at},w) \wedge \\ &\qquad Default(p,\text{mbox},e),\end{aligned}$$

where '?' is omitted before variables to resemble the conventional FO notation. Having these, we need to create the formula $\varphi_{\boldsymbol{q}_{\text{cons}}}$ that always outputs exactly the same graph as $\boldsymbol{q}_{\text{cons}}$. As discussed above, this formula can be constructed by projecting out the non-relevant variables and checking that the triples are well-formed. In particular, we obtain

$$\begin{aligned}\varphi_{\boldsymbol{q}_{\text{cons}}}(x,y,z) &= \neg IsBlank(y) \wedge \\ &\quad \big(\exists p,n,w\,\big[\varphi_{\{?p,?n,?w\}}(p,n,w) \wedge (x=n \wedge y=\text{works\_at} \wedge z=w)\big] \vee \\ &\quad\ \exists p,n,w,e\,\big[\varphi_{\{?p,?n,?w,?e\}}(p,n,w,e) \wedge (x=n \wedge y=\text{mbox} \wedge z=e)\big] \big).\end{aligned}$$

Our next result is the inclusion in the other direction.

▶ **Lemma 6.** *The language* FO_{rdf} *is contained in* $\text{c-SPARQL}_{\mathsf{bf}}$.

Proof (idea). The proof of this lemma is an inductive construction that exploits the idea that the difference operation on mappings can be expressed in SPARQL by means of the following application of optional matching [27]. Let

$$P_1\ \mathsf{MINUS}\ P_2 = (P_1\ \mathsf{OPT}\ (P_2\ \mathsf{AND}\ (?x_1,?x_2,?x_3)))\ \mathsf{FILTER}\ \neg bound(?x_1),$$

where $?x_1, ?x_2$ and $?x_3$ are mentioned neither in P_1 nor in P_2. It is readily verified that $[\![P_1\ \mathsf{MINUS}\ P_2]\!]^D_G = [\![P_1]\!]^D_G \setminus [\![P_2]\!]^D_G$ for any dataset D and its graph G. Our construction is again similar to the one in [2], where a reduction from non-recursive Datalog with safe negation to SPARQL graph patterns is provided. ◀

Having these lemmas at hand we conclude the following theorem.

▶ **Theorem 7.** *The languages* $\text{c-SPARQL}_{\mathsf{bf}}$ *and* FO_{rdf} *are equivalent in expressive power.*

This result and its proof have a couple of immediate important consequences. First of them is that the language $\text{c-SPARQL}_{\mathsf{bf,gf}}$ of graph-free and blank-free c-queries is equivalent to the fragment of FO_{rdf} which does not allow for the quaternary predicate $Named$ (we use $\text{FO}^{\text{ternary}}_{\mathsf{rdf}}$ to denote this fragment). This comes from a straightforward inspection of the proofs of the previous lemmas.

▶ **Corollary 8.** *The languages* $\text{c-SPARQL}_{\mathsf{bf,gf}}$ *and* $\text{FO}^{\text{ternary}}_{\mathsf{rdf}}$ *are equivalent in expressive power.*

As a side remark we note that even if the syntax of manipulating graph names in SPARQL is very different from the syntax for manipulating subjects, predicates and objects, semantically the values are treated very similarly, and they are equivalent in terms of expressivity.

The second important consequence is that the blank-free fragment of c-SPARQL is composable. Formally, a query language $\mathcal{Q}$ with the same input and answer sets $\mathcal{I}$, and evaluation function eval, is *composable* if and only if for every pair of queries $\boldsymbol{q}_1, \boldsymbol{q}_2 \in \mathcal{Q}$ there is another query $\boldsymbol{q} \in \mathcal{Q}$ such that $\mathsf{eval}(\boldsymbol{q}_1, \mathsf{eval}(\boldsymbol{q}_2, I)) = \mathsf{eval}(\boldsymbol{q}, I)$ for any input database

$I \in \mathcal{I}$. According to this definition, it makes no sense to talk about composability of the language c-SPARQL$_{\mathsf{bf}}$ of blank-free c-queries, because it does not satisfies the condition that the sets of inputs and answers coincide. But queries in c-SPARQL$_{\mathsf{bf,gf}}$ enjoy such a property, and we can obtain composability of c-SPARQL$_{\mathsf{bf,gf}}$ from composability of $\mathrm{FO}^{\mathrm{ternary}}_{\mathsf{rdf}}$.

▶ **Corollary 9.** *The language* c-SPARQL$_{\mathsf{bf,gf}}$ *is composable.*

We conclude this section with the complexity of the evaluation of blank-free c-queries. The lower bounds of the following result carries almost verbatim from the lower bounds in [27]. The upper bound is also very similar, the only additional step is guessing the values of all the variables which are not mentioned in the template.

▶ **Proposition 10.** *The problem of checking whether a triple is in the answer to a c-query from* c-SPARQL$_{\mathsf{bf}}$ *over a dataset is* PSPACE*-complete in general and in* NLOGSPACE *if the c-query is fixed.*[4] *The bounds hold also for c-queries from* c-SPARQL$_{\mathsf{bf,gf}}$.

Hence the complexity of evaluating blank-free c-queries is the same as that of SPARQL graph patterns, as well as of SPARQL SELECT queries.

4 OPT-free and Well-designed CONSTRUCT queries

The Semantic Web community has adopted the fragment of unions of well-designed graph patterns as a good practice for writing SPARQL queries. This is mainly because enforcing this property prevents users from writing graph patterns that do not agree with the open-world nature of the Semantic Web (see [27] for a more detailed discussion). Furthermore, restricting to unions of well-designed graph patterns drops the (combined) complexity of evaluation from PSPACE-complete to coNP-complete, and to Σ^p_2-complete if projection is allowed. Also, several optimization techniques have been developed for the evaluation of well-designed queries (see [22, 27]). In this section we study the properties of c-queries whose graph patterns are unions of well-designed patterns. We concentrate on the sublanguages of c-SPARQL$_{\mathsf{bf,gf}}$, leaving c-queries with blank nodes in templates for the next section, and restricting to graph-free c-queries for brevity. Note, however, that all the relevant results in this section hold also for c-queries with GRAPH-patterns.

We start with the definition of *well-designed* graph patterns, which are patterns using:

1. no UNION-subpatterns,
2. only FILTER-subpatterns (P FILTER R) such that all variables in R are mentioned in P,
3. only OPT-subpatterns (P_1 OPT P_2) such that all variables in P_2 which appear outside this subpattern are mentioned in P_1.

In this section we consider c-queries with graph patterns that are unions of well-designed patterns. We also consider c-queries without OPT-subpatterns, called *opt-free*. We will use superscripts uwd and of to specify sublanguages satisfying these restrictions, such as c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$. Note that c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ contains c-SPARQL$^{\mathsf{of}}_{\mathsf{bf,gf}}$, since although graph patterns in opt-free c-queries are not a union of well-designed patterns per se, they can be easily transformed into such by applying distributivity rules to push UNION outside and techniques of [2] to enforce the condition on FILTER subpatterns. Somewhat surprisingly, the next lemma shows that this containment holds in other direction as well, which means that adding well-designed OPT to patterns does not increase the expressive power of c-queries.

[4] The latter setting is known as *data complexity* of the problem (see [38]).

▶ **Lemma 11.** *The language* c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ *is contained in* c-SPARQL$^{\mathsf{of}}_{\mathsf{bf,gf}}$.

This result relies on the following fact: Consider a c-query CONSTRUCT H WHERE P in c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$. Then no triple in the answer to this query is generated by those mappings obtained from the evaluation of P in which some of the variables from H are not defined. This obviously does not hold for graph patterns themselves nor for SPARQL SELECT queries, which explains the importance of well-designed OPT in those classes of queries.

An important corollary from the proof of the previous lemma is that a c-query in c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ can be efficiently transformed into an opt-free c-query. In other words, in the context of c-queries well-designed OPT is not just dispensable, but also syntactic sugar.

▶ **Corollary 12.** *Every c-query from* c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ *can be transformed to an equivalent c-query from* c-SPARQL$^{\mathsf{of}}_{\mathsf{bf,gf}}$ *in* LOGSPACE.

We also relate the described languages to a fragment of first order logic, defined next. A formula $\varphi \in \mathrm{FO}^{\mathrm{ternary}}_{\mathsf{rdf}}$ is $\exists$*-positive* if it is in the $\{\exists, \neg, \wedge, \vee\}$ fragment of FO where negation is atomic and only appear over equalities or $IsBlank$ atomic predicates. The language of all such $\exists$-positive formulas is denoted $\exists\mathrm{pos\text{-}FO}^{\mathrm{ternary}}_{\mathsf{rdf}}$.

The following result comes from a straightforward inspection of the reduction from c-SPARQL$_{\mathsf{bf}}$ to FO$_{\mathsf{rdf}}$ (Lemma 4), as the only way to generate negation over the $Default$ predicate is by means of OPT patterns.

▶ **Lemma 13.** *The language* c-SPARQL$^{\mathsf{of}}_{\mathsf{bf,gf}}$ *is contained in* $\exists\mathrm{pos\text{-}FO}^{\mathrm{ternary}}_{\mathsf{rdf}}$.

Quite similarly, an inspection of the proof of Lemma 6 shows that a transformation of an existential formula in which the predicate $Default$ only appears positively gives us a c-query which does not use the OPT operator. Note, however, that this c-query can have negations in the filter expressions.

▶ **Lemma 14.** *The language* $\exists\mathrm{pos\text{-}FO}^{\mathrm{ternary}}_{\mathsf{rdf}}$ *is contained in* c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$.

We can now state the main theorem of this section.

▶ **Theorem 15.** *The languages* c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$, c-SPARQL$^{\mathsf{of}}_{\mathsf{bf,gf}}$ *and* $\exists\mathrm{pos\text{-}FO}^{\mathrm{ternary}}_{\mathsf{rdf}}$ *are equivalent in expressive power.*

We obtain the composability of c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ as a corollary.

▶ **Corollary 16.** *The language* c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ *is composable.*

We conclude this section with the complexity of evaluation of c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$. As it is with expressive power, the complexity of evaluation for this fragment is lower than the complexity of evaluation of SELECT queries with well-designed patterns, which is, as already mentioned, Σ^p_2-complete.

▶ **Proposition 17.** *The problem of checking whether a triple is in the answer to a c-query from* c-SPARQL$^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ *over a dataset is* NP*-complete.*

5 c-Queries with Blank Nodes in Templates

In this section we study the properties of c-queries with blank nodes in templates. Like in the previous section we concentrate on c-SPARQL$_{\mathsf{gf}}$ queries, that is, c-queries that do not use GRAPH operator, and work with RDF graphs but not datasets. However, all relevant results of this section transfer easily to the full class of c-SPARQL queries.

In order to define the semantics of c-queries with blank nodes, for every template H and dataset D we fix a family $F(H, D)$ of *renaming functions*. This family contains, for every mapping μ from $\mathrm{var}(H)$ to the domain of D, an injective function $f_\mu : \mathsf{blank}(H) \to \mathbf{B} \setminus \mathsf{blank}(D)$. These functions must have pairwise disjoint ranges.

Then the *answer* $\mathsf{ans}(\boldsymbol{q}, D)$ to a c-query $\boldsymbol{q} = \mathsf{CONSTRUCT}\ H\ \mathsf{WHERE}\ P$ over an input dataset D is the RDF graph

$$\mathsf{ans}(\boldsymbol{q}, D) = \{\mu(f_\mu(t)) \mid \mu \in [\![P]\!]^D, t \text{ is a triple in } H \text{ and } \mu(f_\mu(t)) \text{ is well formed}\},$$

where f_μ is the corresponding renaming function for μ in $F(H, D)$.

▶ **Example 18.** Recall again the dataset from Figure 1 and consider the c-query

```
CONSTRUCT {(_:b, manages, ?n), (?n, mbox, ?e)}
WHERE (
((?p, name, ?n) AND (?p, works_at, ?w))
         OPT   (?p, mbox, ?e)),
```

where `_:b` is a blank node. This blank node is intended to create a new blank node for each person, representing his manager. However, one must be cautious: the semantics of blank nodes in c-queries creates one blank node per each of the mappings in the evaluation of the inner pattern, and thus two blank nodes are created for Cristian, since there are two different mappings that assign Cristian to $?w$. Recall that the evaluation of the inner pattern of this query over the graph G_{ex} of Figure 1 is the set of mappings $\{\mu_1, \mu_2, \mu_3\}$, according to Figure 1. If we set f_{μ_1}(`_:b`) =`_:b1`, f_{μ_2}(`_:b`) =`_:b2` and f_{μ_3}(`_:b`) =`_:b3` then the result of evaluating the query above over G_{ex} contains the triples `(_:b1,manages,Fran)`, `(_:b2,manages,Cristian)`,`(_:b3,manages,Cristian)` and `(Cristian,mbox,cris@puc.cl)`.

In order to understand the properties of c-SPARQL$_{\mathsf{gf}}$, we start with the study of its expressive power. Since queries from this class can create values from scratch, it does not make much sense to compare them with FO queries. Instead, we focus on the resemblance between the semantics of blank nodes in c-SPARQL$_{\mathsf{gf}}$ queries and the one of nulls in universal solutions for data exchange problems (see [3] for a good introduction to the topic). In the following we show that this resemblance is not a coincidence, since all c-queries in c-SPARQL$_{\mathsf{gf}}$ can be simulated by source-to-target dependencies in the context of data exchange, in the following sense: Given a c-query q in c-SPARQL$_{\mathsf{gf}}$, one can construct a data exchange setting such that the graph created by posing q over an RDF graph corresponds to the result of exchanging this graph according to the data exchange setting (up to renaming of blank nodes). This establishes that these two formalisms are, in a way, equivalent in expressive power, and one of the most important consequences of this result is the non-composability of queries in c-SPARQL$_{\mathsf{gf}}$, in contrast to the blank-free c-queries from the previous sections.

To state these results we recall some terminology on data-exchange. We begin by adapting the definitions of [3, 13] to our context[5]. A *dependency* is an expression of the form

$$\forall \bar{x}\ \forall \bar{y}\ (\varphi(\bar{x}, \bar{y}) \to \exists \bar{z}\ \psi(\bar{x}, \bar{z}))\,, \tag{1}$$

where $\bar{x}, \bar{y}$, and $\bar{z}$ are disjoint tuples of variables and φ and ψ are first-order formulas. We concentrate on a restricted class of dependencies, that we call *source-to-target dependencies*

[5] Our definition differs slightly from the chase for RDF used in [10], but it follows the same spirit.

(or *st-dependencies*), which are those in which φ belongs to $\mathrm{FO}^{\mathrm{ternary}}_{\mathsf{rdf}}$ (i.e., formulas over $Default$ and $IsBlank$ relations), and ψ is a conjunction of atoms over another ternary relation $OTriple$. In data exchange terminology, sets of st-dependencies are usually known as "mappings", but since we already use this term for solutions of graph patterns, we call them *de-mappings*, and denote $\mathrm{DE}_{\mathsf{rdf}}$ the language of de-mappings.

The semantics of a de-mapping Σ in our context can be defined as follows. A first-order structure over $OTriple$ (called a *target instance*) is a *solution under* Σ for a structure over $Default$ and $IsBlank$ (called a *source instance*), if the set Σ of dependencies holds in the union of the source and target instances (recall that the domain of our structures is always the set $\mathbf{T}$ of terms).

In data exchange one is usually interested in computing *universal solutions* for a source instance and a de-mapping Σ. These are solutions that have homomorphic images to all solutions. A typical way to compute universal solutions is by means of the chase procedure. In traditional data exchange settings, this procedure repeatedly tests and enforces the satisfaction of all dependencies that are not satisfied, instantiating each existential variable in the right hand side of dependencies with a fresh *null* value. These nulls have very similar semantics to the semantics of blank nodes in SPARQL settings, so we define the chase using blanks.

Next we define the *chase* of a source instance S under a de-mapping Σ as a target instance constructed by sequentially adding triples to $OTriple$. To compute this instance, for every st-dependency of the form (1) in Σ proceed as follows. Take every assignment $\pi : \bar{x} \cup \bar{y} \to \mathbf{T}$ such that $S \models_{\mathrm{adom}} \varphi(\pi(\bar{x}), \pi(\bar{y}))$ and extend π by assigning a distinct fresh blank node from $\mathbf{B}$ to each variable in $\bar{z}$. Then add to the target instance the fact $OTriple(\pi(v_1), \pi(v_2), \pi(v_3))$ for each conjunct $OTriple(v_1, v_2, v_3)$ in ψ, as long as $\pi(v_2)$ is not a blank node.

The result of the chase is deterministic up to renaming of the introduced blank nodes, so we can consider $\mathrm{DE}_{\mathsf{rdf}}$ as a query language with answers being the results of the chase. This enables us to compare the expressive power of c-queries with that of data exchange settings.

▶ **Theorem 19.** *The languages* c-SPARQL$_{\mathsf{gf}}$ *and* $\mathrm{DE}_{\mathsf{rdf}}$ *are equivalent in expressive power.*

It is known that de-mappings are not composable in the data exchange scenario [14]. We can adapt this argument into our context to obtain the following important negative result.

▶ **Proposition 20.** *The language* c-SPARQL$_{\mathsf{gf}}$ *is not composable.*

Next we refine the results above for the language c-SPARQL$^{\mathsf{uwd}}_{\mathsf{gf}}$. Since we have shown that such queries are equivalent to positive FO, it would be reasonable to guess that c-SPARQL$^{\mathsf{uwd}}_{\mathsf{gf}}$ is equivalent to the query language given by de-mappings where every dependency (1) is such that the formula φ is a conjunction of atoms. This last language is, in fact, a very well studied class of de-mappings, called *GLAV-mappings* (see, e.g., [13, 21]), and are denoted by $\mathrm{GLAV}_{\mathsf{rdf}}$ in this paper.

Unfortunately, the following example shows that the previous intuition is not correct.

▶ **Example 21.** Consider the c-query

```
CONSTRUCT {(_:b, p, ?x), (_:b, p, ?y)}
WHERE ((?x, p, a) OPT (?x, p, ?y)).
```

Note that here the same blank needs to be added to both of the triples in the template whenever a mapping that bounds both $?x$ and $?y$ exists. However, we also need to account

for mappings that bind only $?x$. Hence, this c-query is not equivalent to the de-mapping

$$\begin{aligned} \forall x \forall y \left(Default(x,p,a) \wedge Default(x,p,y) \right. &\rightarrow \left. \exists z \left(OTriple(z,p,x) \wedge OTriple(z,p,y) \right) \right), \\ \forall x \left(Default(x,p,a) \right. &\rightarrow \left. \exists z \, OTriple(z,p,x) \right), \end{aligned}$$

because it creates additional blank nodes whenever the same pair of IRIs witnesses both dependencies. In fact, one can show that this c-query is not equivalent to any query in $\text{GLAV}_{\mathsf{rdf}}$.

In the above example both the chase of the de-mapping and the answer to the CONSTRUCT query are *homomorphically equivalent*, in the sense of [19]. This correspondence is not accidental, and an equivalence between $\text{GLAV}_{\mathsf{rdf}}$ and $\text{c-SPARQL}^{\mathsf{uwd}}_{\mathsf{gf}}$ can be shown under this relaxed notion of equivalence between RDF graphs. We omit these results for space reasons, but we can show the following containment without introducing any additional notation.

▶ **Proposition 22.** *The language* $\text{GLAV}_{\mathsf{rdf}}$ *is contained in* $\text{c-SPARQL}^{\mathsf{uwd}}_{\mathsf{gf}}$.

Regardless, the following corollary is a consequence of the proof of Proposition 20. This again goes in line with results for data exchange, since *GLAV-mappings* are not composable in general.

▶ **Corollary 23.** *The language* $\text{c-SPARQL}^{\mathsf{uwd}}_{\mathsf{gf}}$ *is not composable.*

We conclude this section with the following observation. Example 21 is problematic as we use the same blank nodes in two triples in the CONSTRUCT template. If we disallow blank nodes we regain equivalence between c-queries and data exchange settings, since $\text{c-SPARQL}^{\mathsf{uwd}}_{\mathsf{bf,gf}}$ is equivalent to the setting given by *GAV-mappings*, that is, GLAV-mappings of the form (1) but without existential variables.

6 Adding Recursion to SPARQL

Recursion is an integral part of most practical query languages such as SQL:1999 [34]. The recent version 1.1 of SPARQL also allows for some form of recursion, namely *property paths* [39], a binary primitive based on two way regular path queries, a well-known query language for graph databases [6]. However, as shown in e.g. [23,28], this recursion is limited and cannot express several interesting and relevant queries. It is possible to simulate more recursion by exploiting the power of *entailment regimes* like OWL 2 RL [16]. But to put it simple, this formalism is also quite limited and, more important, not part of SPARQL 1.1 itself. The aim of this section is to develop syntax and semantics for a full recursive operator in SPARQL and study its properties.

Before starting the formal development we discuss what are the difficulties of introducing recursion in SPARQL. The semantics in the majority of query languages that allow for recursion is defined in terms of a fixed point operator. But to have such operator one needs to be able to pose a query over the result of another query, that is, the query language must have the same input and answer domains. Hence it is not possible to introduce recursion based on SPARQL SELECT queries: they evaluate over a dataset, but answer a set of mappings. On the contrary, we can do it for c-queries, since the output and input of these queries are RDF graphs.

In this section we show how to apply our study of c-queries in the development of a recursive operator for SPARQL. Our proposal resembles the syntax and semantics of such an operator in the SQL:1999 standard. Let us explain our proposal by means of an example.

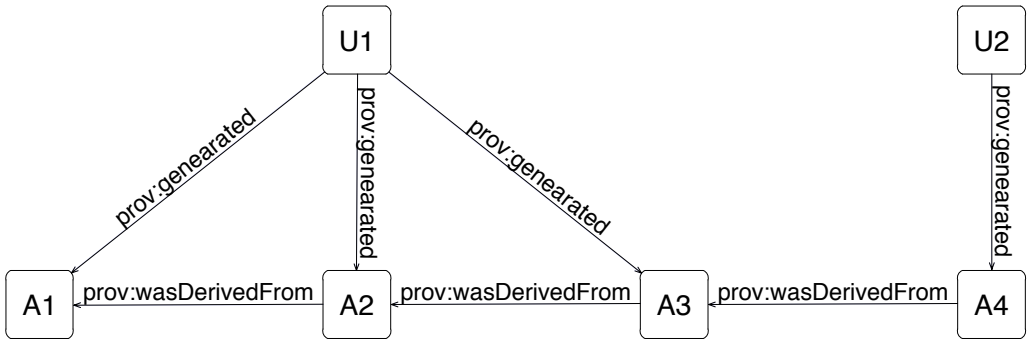


Figure 3 RDF graph storing provenance history of Wikipedia articles A1, A2, A3 and A4.

▶ **Example 24.** Consider the RDF graph in Figure 3, where a piece of the provenance information about the history of Wikipedia pages is depicted, according to PROV data model (see [26]). New articles are derived from their previous versions, and each version is linked to the user that was responsible for its generation. When inspecting this graph, one of the things we may be interested in is to obtain all triples of the form $(A, \text{same:user}, A')$ such that A' is an article derived from A by a path of revisions generated by the same user. For example, in the graph of Figure 3, we would want to obtain triple $(A3, \text{same:user}, A1)$, among others. In SPARQL we propose to obtain all such triples by means of the following query:

```
WITH RECURSIVE http://db.ing.puc.cl/temp AS
{
  CONSTRUCT {(?x, ?u, ?y)}
  WHERE
   ((?x, prov:wasDerivedFrom, ?y) AND
       (?u,prov:generated,?x) AND (?u,prov:generated,?y))
   UNION
   ((?x, prov:wasDerivedFrom, ?z) AND (?u,prov:generated,?x) AND
       (http://db.ing.puc.cl/temp GRAPH (?z, ?u, ?y)))
}
CONSTRUCT (?x, same:user, ?y)
WHERE (http://db.ing.puc.cl/temp GRAPH (?x, ?u, ?y))
```

The intention of this query is as follows. The first line is the actual fixed point operator: it specifies that the RDF graph `http://db.ing.puc.cl/temp` is a temporal graph, which is iteratively computed until the least fixed point of the subsequent query is reached. In this example, the iterated query in braces states that all the triples in `http://db.ing.puc.cl/temp` are of the form (X, U, Y), where Y is either a revision of X or is linked to X via a chain of revisions of arbitrary length, but in which all revisions involved were generated by user U. Finally, the CONSTRUCT part of the query in the end extracts the desired information from the computed temporal graph `http://db.ing.puc.cl/temp` into the output graph.

In this example, as in the rest of the paper, we deal with CONSTRUCT queries, but of course nothing prevents the main subquery to be of any other form (for example, one could retrieve mappings using SELECT in the main subquery) We also concentrate on blank-free

c-queries and leave the study of recursive c-queries with blank nodes in the templates for future work.

▶ **Definition 25.** A *recursive c-query* is either a blank-free c-query from c-SPARQL$_{\mathsf{bf}}$ or an expression of the form

$$\mathsf{WITH\ RECURSIVE}\ t\ \mathsf{AS}\ \{\boldsymbol{q}_1\}\ \boldsymbol{q}_2, \tag{2}$$

where t is an IRI from $\mathbf{I}$, $\boldsymbol{q}_1$ is a c-query from c-SPARQL$_{\mathsf{bf}}$, and $\boldsymbol{q}_2$ is a recursive c-query. The set of all recursive c-queries is denoted rec-c-SPARQL$_{\mathsf{bf}}$.

We reinforce the idea that in this definition $\boldsymbol{q}_1$ is non-recursive, but $\boldsymbol{q}_2$ could be recursive by itself, which allows us to compose recursive definitions.

Having the syntax at hand we define the semantic of recursive c-queries. Given datasets D, D' with default graphs G_0 and G'_0, we define $D \cup D'$ as the dataset with the default graph $G_0 \cup G'_0$ and $\mathsf{gr}_{D\cup D'}(u) = \mathsf{gr}_D(u) \cup \mathsf{gr}_{D'}(u)$ for any URI u.

▶ **Definition 26.** If a recursive query $\boldsymbol{q}$ from rec-c-SPARQL$_{\mathsf{bf}}$ is a simple c-query, then its answer $\mathsf{ans}(\boldsymbol{q}, D)$ over a dataset D is defined according to the semantics of c-queries. Otherwise, that is, if $\boldsymbol{q}$ is of the form (2), then the *answer* $\mathsf{ans}(\boldsymbol{q}, D)$ is equal to $\mathsf{ans}(\boldsymbol{q}_2, D_{\mathrm{lfp}})$, where D_{lfp} is the least fixed point of the sequence $D_0, D_1, \ldots$ with $D_0 = D$ and

$$D_{i+1} = D \cup \{\langle t, \mathsf{ans}(\boldsymbol{q}_1, D_i)\rangle\}, \text{ for } i \geq 0.$$

Naturally, the above definition makes sense only when the sequence $D_0, D_1, \ldots$ has a (finite) fixed point. In this case, we say that the answer $\mathsf{ans}(\boldsymbol{q}, D)$ is *well-defined*. By our results on expressive power, one way to guarantee this is to require graph pattern of the c-query $\boldsymbol{q}_1$ to be a union of well-designed patterns, since this implies that the sequence is monotone. However, we can partially relax this condition and concentrate on the following fragment of rec-c-SPARQL$_{\mathsf{bf}}$. A recursive c-query $\boldsymbol{q}$ is *semi-positive* iff it is either a simple c-query, or it is of the form (2) with $\boldsymbol{q}_2$ semi-positive and every subpattern P in $\boldsymbol{q}_1$ satisfying the following conditions:

1. if P is $(g\ \mathsf{GRAPH}\ P')$ with $g \in \mathbf{V} \cup \{t\}$ then P' is well-designed, and
2. if P is $(P_1\ \mathsf{OPT}\ P_2)$ then all subpatterns $(g\ \mathsf{GRAPH}\ P')$ of P_2 are such that $g \in \mathbf{I} \setminus \{t\}$.

The language of all semi-positive recursive c-queries is denoted by rec-c-SPARQL$^{\mathsf{semi}}_{\mathsf{bf}}$. They always have fixed points, as desired.

▶ **Proposition 27.** *For every recursive c-query $\boldsymbol{q}$ in* rec-c-SPARQL$^{\mathsf{semi}}_{\mathsf{bf}}$ *and dataset D the answer* $\mathsf{ans}(\boldsymbol{q}, D)$ *is well-defined.*

Next we study its expressive power of our language and show that it is equivalent to a particular class of Datalog programs (see [1] for a good introduction on Datalog).

Let V be a vocabulary of relational predicates. A *rule* is an expression of the form

$$Pr(\bar{x}) \leftarrow \varphi(\bar{x}, \bar{y}),$$

where $\bar{x}$ and $\bar{y}$ are tuples of variables, Pr is a predicate from $V \cup \{OTriple\}$, and φ is a conjunction of positive and negated atoms (including equalities) over $\mathbf{I} \cup V \cup \{Default, Named\}$, such that every variable from $\bar{x}$ appears in φ. In such a rule, $Pr(\bar{x})$ is the *head* and $\varphi(\bar{x}, \bar{y})$ is the *body*.

A *Datalog program with rule-by-rule stratification* is a sequence $\Pi_1, \ldots, \Pi_n$ of sets of rules for which there exist a set $V = \{Pr_1, \ldots, Pr_n\}$ such that the following holds:

1. the head of each rule in Π_i is the predicate Pr_i;
2. each Π_i does not mention any Pr_j with $j > i$;
3. each Π_i does not mention Pr_i in negated atoms.

Without loss of generality we assume that $Pr_n = OTriple$. The language of all Datalog programs with rule-by-rule stratification is denoted by $\text{Datalog}^{\mathsf{rbr}}_{\mathsf{rdf}}$. The semantics of these programs over structures in the signature of FO_{rdf} is the standard fixed point semantics (see, e.g., [1] for a formal definition).

▶ **Theorem 28.** *The languages* $\text{rec-c-SPARQL}^{\mathsf{semi}}_{\mathsf{bf}}$ *and* $\text{Datalog}^{\mathsf{rbr}}_{\mathsf{rdf}}$ *are equivalent in expressive power.*

We conclude this section with some discussion on the relationship of the semi-positive recursive SPARQL with other known formalisms. First, from the last theorem and the results of [11] we may conclude that $\text{rec-c-SPARQL}^{\mathsf{semi}}_{\mathsf{bf}}$ contains first order logic with transitive closure. Second, it is a technicality to check that this query formalism contains SPARQL 1.1 property paths [39], c-query answering over OWL 2 RL entailment regime [16], the algebra defined in [35], navigational SPARQL [28], as well as GraphLog [11] and TriAL [23] query languages. Also, it is possible to show that none of these formalisms can express all $\text{rec-c-SPARQL}^{\mathsf{semi}}_{\mathsf{bf}}$ queries, that is, the containment is strict in all the cases. Hence, we may conclude that $\text{rec-c-SPARQL}^{\mathsf{semi}}_{\mathsf{bf}}$ is a clean unification of all these languages, and, as we believe, it deserves a further dedicated studies, both theoretical and applied.

7 Conclusions and Future Work

We presented a thorough study of the expressive power and complexity of evaluation of SPARQL CONSTRUCT queries. By studying these queries we provided a strong bridge between SPARQL and well-developed frameworks such as first-order logic and Datalog. In particular, we gave a clean proof of the equivalence between CONSTRUCT queries without blank nodes and first-order logic, characterized well-designed CONSTRUCT queries by a reduction into a positive fragment of first-order logic, and presented a translation between the full fragment of CONSTRUCT queries and a specific setting for data exchange. Finally, having a good understanding of these queries we were able to present a proposal for extending SPARQL with recursion, which we proved to be equivalent in expressive power to Datalog with rule-by-rule stratification.

CONSTRUCT queries are an important fragment of SPARQL since they provide the standard language for querying RDF to produce RDF as output. Query languages with this property have several advantages, such as allowing for composability and recursion. The results in this paper present a first formal study of this fragment, and we believe the Semantic Web community will take good advantage of them. As future work we would like to extend our results to advance in the understanding of more expressive versions of SPARQL. There is still a good deal of research to be done in characterizing CONSTRUCT queries allowing for blank nodes in the template, as well as studying c-queries allowing for advanced SPARQL 1.1 operators. It is also left as future work to implement the recursive fragment, and to develop and apply techniques for its optimization.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

2 Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In *ISWC*, pages 114–129, 2008.

3 Marcelo Arenas, Pablo Barcelo, Leonid Libkin, and Filip Murlak. Relational and XML data exchange. *Synthesis Lectures on Data Management*, 2(1):1–112, 2010.

4 Marcelo Arenas, Sebastián Conca, and Jorge Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *Proceedings of the 21st international conference on World Wide Web*, pages 629–638. ACM, 2012.

5 Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *PODS*, pages 305–316, 2011.

6 Pablo Barceló Baeza. Querying graph databases. In *Proceedings of the 32nd symposium on Principles of database systems*, pages 175–188. ACM, 2013.

7 Carlos Buil-Aranda, Marcelo Arenas, and Oscar Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *The Semanic Web: Research and Applications*, pages 1–15. Springer, 2011.

8 Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. SPARQL query containment under $\mathcal{SHI}$ axioms. In *AAAI*, 2012.

9 Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. SPARQL query containment under RDFS entailment regime. In *IJCAR*, pages 134–148, 2012.

10 Rada Chirkova and George HL Fletcher. Towards well-behaved schema evolution. In *WebDB*, 2009.

11 Mariano P. Consens and Alberto O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 404–416. ACM, 1990.

12 Orri Erling and Ivan Mikhailov. RDF support in the virtuoso DBMS. In *Networked Knowledge-Networked Media*, pages 7–24. Springer, 2009.

13 Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

14 Ronald Fagin, Phokion G Kolaitis, Lucian Popa, and Wang-Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems (TODS)*, 30(4):994–1055, 2005.

15 Floris Geerts, Grigoris Karvounarakis, Vassilis Christophides, and Irini Fundulaki. Algebraic structures for capturing the provenance of SPARQL queries. In *ICDT*, pages 153–164, 2013.

16 Birte Glimm and Chimezie Ogbuji. SPARQL 1.1 Entailment Regimes. W3C Recommendation, 2013. Available at `http://www.w3.org/TR/sparql11-entailment/`.

17 Harry Halpin and James Cheney. Dynamic provenance for SPARQL updates. In *ISWC*, 2014.

18 Steve Harris, Nick Lamb, and Nigel Shadbolt. 4store: The design and implementation of a clustered rdf store. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pages 94–109, 2009.

19 Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. Everything you always wanted to know about blank nodes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2014.

20 Egor V. Kostylev and Bernardo Cuenca Grau. On the semantics of SPARQL queries with optional matching under entailment regimes. In *ISWC*, 2014.

21 Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

22 Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25, 2013.

23 Leonid Libkin, Juan Reutter, and Domagoj Vrgoč. TriAL for RDF: adapting graph query languages for RDF data. In *Proceedings of the 32nd symposium on Principles of database systems*, pages 201–212. ACM, 2013.

24 Katja Losemann and Wim Martens. The complexity of evaluating path expressions in SPARQL. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 101–112. ACM, 2012.

25 Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

26 Paolo Missier, Khalid Belhajjame, and James Cheney. The w3c prov family of specifications for modelling provenance metadata. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 773–776. ACM, 2013.

27 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

28 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nSPARQL: A navigational language for RDF. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):255–270, 2010.

29 François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *SWIM*, 2011.

30 Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–50. ACM, 2014.

31 Axel Polleres and Johannes Peter Wallner. On the relation between SPARQL1.1 and answer set programming. *Journal of Applied Non-Classical Logics*, 23(1-2):159–212, 2013.

32 Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation, 2008. Available at `http://www.w3.org/TR/rdf-sparql-query/`.

33 Eric Prud'hommeaux, Andy Seaborne, et al. SPARQL query language for RDF, 2006.

34 Raghu Ramakrishnan, Johannes Gehrke, and Johannes Gehrke. *Database management systems*, volume 3. McGraw-Hill New York, 2003.

35 Edward L Robertson. Triadic relations: An algebra for the semantic web. In *Semantic Web and Databases*, pages 91–108. Springer, 2005.

36 Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *ICDT*, pages 4–33, 2010.

37 Andy Seaborne. ARQ-A SPARQL processor for Jena. *Obtained through the Internet: http://jena. sourceforge. net/ARQ/*, 2010.

38 Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

39 W3C SPARQL Working Group. SPARQL 1.1 Query language. W3C Recommendation, 21 March 2013. Available at `http://www.w3.org/TR/sparql11-query/`.

Separability by Short Subsequences and Subwords

Piotr Hofman and Wim Martens

Department of Computer Science, University of Bayreuth, Germany

Abstract

The separability problem for regular languages asks, given two regular languages I and E, whether there exists a language S that separates the two, that is, includes I but contains nothing from E. Typically, S comes from a simple, less expressive class of languages than I and E. In general, a simple separator S can be seen as an approximation of I or as an explanation of how I and E are different. In a database context, separators can be used for explaining the result of regular path queries or for finding explanations for the difference between paths in a graph database, that is, how paths from given nodes u_1 to v_1 are different from those from u_2 to v_2. We study the complexity of separability of regular languages by combinations of subsequences or subwords of a given length k. The rationale is that the parameter k can be used to influence the size and simplicity of the separator. The emphasis of our study is on tracing the tractability of the problem.

1998 ACM Subject Classification H.1.0 Models and Principles – General, H.2 Database Management, F.4.3 Formal Languages – Decision problems

Keywords and phrases Separability, complexity, graph data, debugging

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.230

1 Introduction

More and more people today are being confronted with systems that are becoming increasingly complex. Computers are becoming more and more powerful and, in the current boom of *big data*, are making decisions based on rapidly growing data sets. When we use such systems, or when we develop them, it is crucial that we have a sufficient understanding of why they do what they do. For example, when a robot performs an unexpected action, a developer should understand why the robot did the action before she can fix the error. Similarly, when a query on a database returns an answer that should not be there, we need some understanding of the data and the query before we can make a correction.

This motivates a need to search for explanations of the behavior of complex systems. We want to make a first step towards investigating to which extent *separation problems* can be useful for explaining the result of queries. Separation problems come from language theory and study differences between languages. Assume that we have two regular word languages I and E, given by their non-deterministic finite automata. A language S *separates* I from E if it *includes* I and *excludes* E, that is $I \subseteq S$ and $S \cap E = \emptyset$. If S comes from a class of languages $\mathcal{S}$, it could be seen as an approximation of I within $\mathcal{S}$. The language E can be used to tune how closely S should approximate I. For example, if E is the complement of I, then no approximation is possible and finding a separator reduces to finding an $S \in \mathcal{S}$ that is equal to I.[1] In this paper, we are mostly interested in separators that come from classes of very simple languages that only express properties about subsequences and subwords.

[1] In this case, separation corresponds to a *rewritability* or *definability* problem.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 230–246

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We claim that separation problems are rather general and seem to be helpful in a wide range of scenarios. For example, separators can give users a description of why tuples are not selected by a regular path query. Say that a regular path query r on a graph database G does not return an answer (u, v) that we expected. In the usual semantics of regular path queries, this means that there is no path from u to v that matches r in G. If we consider the graph G as a finite automaton A_G with initial state u and accepting state v, then a separator for r and A_G that is simple enough to be understandable by a human could provide a description of why the expected tuple (u, v) was not in the answer. For example, if $S = \Sigma^* a \Sigma^* b \Sigma^*$ would be a separator, it could mean that r requires paths to have an a-edge before a b-edge (since $r \subseteq S$) but no such path exists from u to v. Notice that, in this scenario, A_G is expected to be much larger than r.

Similarly, separators may be useful to understand differences between subgraphs. More precisely, consider a system that is abstracted as a (large) finite, edge-labeled graph G in which each edge-label represents an action of the system. Assume that the system, after having started in some initial state i, arrives, after a long up-time, in a certain undesirable state s^- whereas, according to common sense, it should have arrived in s^+. As an aid to understand why the system arrived in s^- instead of s^+ it may be helpful to compute a separator between between the systems (G, i, s^-) and (G, i, s^+), consisting of all paths in G that lead from i to s^- and from i to s^+, respectively. For example, if $S = \Sigma^* a \Sigma^* b \Sigma^*$ would be a separator, it would mean that all labels of paths from i to s^- have a subsequence ab, whereas this is not the case for any of the paths from i to s^+. In this sense, S has the potential to pinpoint a difference between (G, i, s^-) and (G, i, s^+) in simple terms.

In this paper we want to make a first step in this direction. More precisely, we study the complexity of separability. That is, given two languages I and E and a class of separators $\mathcal{S}$, we study the complexity of deciding if there exists an $S \in \mathcal{S}$ that separates I from E. Here, I and E are given as non-deterministic finite automata (or, equivalently, edge-labeled graphs) and for $\mathcal{S}$ we consider languages that reason about the presence and/or absence of certain subsequences or subwords.

Previous work [5, 15] considered separability with respect to *piecewise testable languages*. A language is piecewise testable if it can be defined as a boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where the a_i are symbols from Σ. So, piecewise testable languages reason about subsequences of words. It can be decided in PTIME if the language of two given NFAs I and E can be separated by a piecewise testable language [5, 15]. Tractability of this problem may come as a slight surprise in the light that many basic static analysis questions concerning NFAs (such as containment and universality, for example) are already PSPACE-complete. However, in the case that a separator exists, it is not yet clear from [5, 15] how to construct a separator that would be useful for explaining the behavior of a system to a user. Indeed, the work shows that *non-existence* of a separator for I and E can be witnessed by a polynomial-size common pattern but, if a separator exists, it can be a complex boolean combination that reasons about long subsequences.

We want to come closer to simple separators by limiting the boolean combinations and the length of the subsequences involved. That is, we look at unions, intersections, and positive combinations of languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$ where n is bounded from above by a given parameter k. We also investigate similar combinations of subword languages, that is, languages of the form $\Sigma^* a_1 a_2 \cdots a_n \Sigma^*$. Our motivation to look at subsequence languages and subword languages comes from our belief that these may be helpful in generating understandable explanations: one does not have to be an expert in the internal design of a system to understand that one can avoid the error state by "not performing action b after action a".

Our results focus on finding which combinations of separators and languages I, E may lead to favorable complexities for separability. Apart from the most general cases we consider, the complexity results range from PTIME to Π_2^P for separability by combinations of subsequence languages and from PTIME to PSPACE for combinations of subword languages. For the most general cases, our best current upper bound is NEXPTIME. Some of our results are reductions to simpler problems, such as separability of a language from a word. These simpler problems are interesting in their own respect. For example, it is also interesting to generate a simple reason why, e.g., a regular expression cannot be matched in a very long text.

Finally, we stress that we think that a system for generating simple separators or explanations should explore many different classes of separators. In this paper we only focus on subwords or subsequences, but a simple explanation for, say, the behavior of a query may also consist of completely different concepts. Just to mention one example, we intend to investigate separability by Parikh images in the future as well. Kopczynsky and Widjaja Lin [7] show that this approach could be feasible from a complexity point of view. A system could then search in parallel for separators in a wide array of classes and offer the user the simplest ones it can find.

The motivation in this paper mainly comes from explaining the results of queries, which is also a significant motivation for *provenance in databases*, a very successful line of research (see, e.g., [3, 23] for an overview). Storing and handling provenance in databases for explaining the results of queries is an approach that is orthogonal to ours, since we do not rely on the availability of provenance data. Another approach on explaining the results of queries, on relational data, was recently taken by Roy and Suciu [18]. Their approach also does not depend on the presence of provenance data and is based on *intervention*, which means that they investigate which tuples significantly affect the result of the answer.

Related Work

This paper is directly motivated by [5, 15], where it was shown that it can be decided in PTIME if two regular languages given by their NFA can be separated by a piecewise testable language. Recently, this problem has also been shown to be PTIME-complete [24]. Separability by locally testable and locally threshold testable languages, which are closely related to piecewise testable languages, was investigated by Place et al. [14]. They provide algorithms to solve both problems in co-NEXPTIME and 2EXPSPACE respectively. In addition they proved that both problems are coNP-hard. Place and Zeitoun recently proved that deciding separability of regular languages by first-order-logic is in EXPTIME [16] if the languages are given by their semigroups. For given NFAs, their techniques imply a 2EXPTIME upper bound. We also refer to [17] for an overview of these results.

Our approaches have roots in separability by piecewise testable and locally testable languages. Piecewise testable languages were defined and studied by Simon [19, 20], who showed that a regular language is piecewise testable iff its syntactic monoid is J-trivial and iff both the minimal DFA for the language and the minimal DFA for the reversal are partially ordered. Stern [21] proved that it is PTIME-decidable if a language, given by its DFA, is piecewise testable. A language is *locally testable* if membership of a word can be tested by inspecting its prefixes, suffixes and infixes, up to some length that depends on the language. The problem if a given regular language is locally testable was posed by McNaughton and Papert [12] and independently solved by McNaughton and Zalcstein [11, 26] and by Brzozowski and Simon [2].

Separation is closely related to Craig interpolation [4]. Craig interpolants are defined with respect to a given implication $\varphi \Rightarrow \psi$ and are formulas ρ such that $\varphi \Rightarrow \rho$ and $\rho \Rightarrow \psi$.

Moreover, ρ only contains atoms that occur in *both* φ and ψ. Hence, ρ can be seen as a separator between φ and $\neg\psi$. Craig interpolants have been used for verifying safety in a system in the context of model checking [6, 10]. The classical results on Craig interpolation say that, in first-order logic, every valid implication has an interpolant. (So, for valid implications, it is trivial to decide if an interpolant exists.) Lutz and Wolter investigated complexity questions in the context of interpolants for the description logic $\mathcal{ALC}$ [8]. In particular, they showed that deciding whether there is a uniform interpolant (over a given signature) of a given TBox is 2EXPTIME-complete.

The concept of *query inseparability* was recently investigated by Botoeva et al. [1]. This problem asks, for two knowledge bases K_1 and K_2 and a class C of queries, whether every query in C has the same answer over K_1 and K_2.

Very recently, Masopust and Thomazo investigated the complexity of the *characterization problem* for k-piecewise testable languages, that is, boolean combinations of $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$ with $n \leq k$ [9]. The characterisation problem asks whether a given language *is* a k-piecewise testable language.

2 Preliminaries

For a finite set S, we denote its cardinality by $|S|$. By Σ we always denote an alphabet, that is, a finite set of symbols. A (Σ-)*word* w is a finite concatenation of symbols $a_1 \cdots a_n$, where $n \geq 0$ and $a_i \in \Sigma$ for all $i = 1, \ldots, n$. The *length of* w, denoted by $|w|$, is n. The empty word is denoted by ε and has length zero. The set of all Σ-words is denoted by Σ^*. A *language* is a set of words.

We assume familiarity with finite automata and regular expressions. In regular expressions, we also use sets to denote disjunctions of symbols. For example, $\Sigma^* a \Sigma^*$ denotes all words that have an a.

We denote a *(nondeterministic) finite automaton* or *NFA* $\mathcal{A}$ as a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is its finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. We sometimes use $q_1 \xrightarrow{a} q_2 \in \delta$ to denote that $q_2 \in \delta(q_1, a)$. The *size* of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is defined as $|Q| + \sum_{q,a} |\delta(q, a)|$, which is the total number of transitions and states. An NFA is *deterministic* (or a *DFA*) when every $\delta(q, a)$ consists of at most one element.

For an automaton A or regular expression r we write $L(A)$, resp., $L(r)$ for their language. We sometimes identify a regular expression r or automaton A with its language and write, for example, $w \in r$ instead of $w \in L(r)$.

2.1 Subsequences and Subwords

Let $v = a_1 \cdots a_n$. If $w \in \Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, we say that v is a *subsequence* of w and we denote it by $v \preccurlyeq w$. Moreover v is a k-subsequence if it is a subsequence and has length at most k. If $w \in \Sigma^* v \Sigma^*$, then v is a *subword* of w, denoted $v \trianglelefteq w$. It is a k-subword if, additionally, it has length at most k. For $k \in \mathbb{N}$, a *k-subsequence language* is a language of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_\ell \Sigma^*$ with $\ell \leq k$ and $a_1, \ldots, a_\ell \in \Sigma$. Similarly, a *$k$-subword language* is a language of the form $\Sigma^* a_1 a_2 \cdots a_\ell \Sigma^*$ with $\ell \leq k$ and $a_1, \ldots, a_\ell \in \Sigma$. A language is a *subsequence language*, resp., *subword language* if there exists a k such that it is a k-subsequence, resp., k-subword language.

We use the following notation on automata, expressions, and languages L:

- k-Subseqs(L) (resp., k-Subwords(L)) is the set of k-subsequences (resp., k-subwords) of words from the language L, (sometimes we identify a word w with a language $\{w\}$ and we use notation k-Subwords(w) instead of k-Subwords($\{w\}$)),

- closure$^{\preccurlyeq}(L)$ (resp., closure$^{\sqsubseteq}(L)$) is the set of words that contain a word from L as a subsequence (resp., subword).

▶ **Observation 1.** *For a given word w, one can construct in polynomial time a DFA of size $O(|w|^2)$ which recognizes all words that are a subsequence of w.*

As a corollary, notice that we can also construct a DFA of size $O(k \cdot |w|^2)$ for all k-subsequences of a given word in polynomial time. This can be obtained from the DFA of Observation 1 by taking a product with the state DFA that accepts all words of length at most k. (This DFA has $k+1$ states.)

▶ **Observation 2.** *For a given word w and alphabet Σ one can construct in polynomial time a DFA which accepts all Σ-words that are supersequences of w.*

2.2 Separability

For two languages I and E, we say that language S *separates I from E* if S contains I and is disjoint from E. Notation-wise, we will consistently use I for the language to be *included* in S and E for the language to be *excluded* from S. We say that I *is separable from E* by a class of languages $\mathcal{S}$ if there exists an $S \in \mathcal{S}$ that separates I from E.

We are interested in cases where the separating language S come from classes of *simple* languages $\mathcal{S}$, such as:

- subsequence languages;
- subword languages;
- finite unions, intersections, positive combinations, or boolean combinations of subsequence- or subword languages.

By *boolean combinations* we mean finite combinations of unions, intersections, and complements. *Positive* combinations are boolean combinations that do not use complements. We note that boolean combinations of subsequence languages are also known as *piecewise testable languages* [19, 20].

We parametrize the families of separators by the length of the subsequences or subwords that we allow. For example, if $\mathcal{S}$ is the class of unions of subsequence languages, then $\mathcal{S}_k$ denotes the class of unions of k-subsequence languages. We study the following problem.

Problem:	Separability of $\mathcal{I}$ from $\mathcal{E}$ by $\mathcal{S}$	($\mathcal{I}, \mathcal{E}, \mathcal{S}$: classes of languages)
Input:	Languages $I \in \mathcal{I}$, $E \in \mathcal{E}$ given by NFAs, and a parameter $k \in \mathbb{N}$ in unary.	
Question:	Is there an $S \in \mathcal{S}_k$ that separates I from E?	

We consider variants of the separability problem where $\mathcal{S}$ comes from the aforementioned simple classes of separator languages. The classes $\mathcal{I}$ and $\mathcal{E}$ are usually either regular languages or singleton words.

The input parameter k serves as a measure for how complex we allow the separating language to be, i.e., combinations of k-subsequence languages or k-subword languages. Of course, since a separator can still be relatively complex even if k is small, we also want to explore other parameters in future work.

When we speak about the complexity of separability, we always assume that I and E are given as NFAs. We denote the sizes of these NFAs by $|I|$ and $|E|$ respectively. We assume that k is provided in unary (or, alternatively, k should not be larger than the given NFAs) to simplify some of the proofs.

2.3 Inclusion and Exclusion Equivalence

In this section we provide tools that allow us to simplify some cases of separability. We say that language I is *inclusion-equivalent* to I' with respect to a class of separators $\mathcal{S}$ if, for every language E, we have that

$$I \text{ is separable from } E \text{ by } \mathcal{S} \iff I' \text{ is separable from } E \text{ by } \mathcal{S}.$$

Similarly, E is *exclusion-equivalent* to E' with respect to $\mathcal{S}$ if, for every language I,

$$I \text{ is separable from } E \text{ by } \mathcal{S} \iff I \text{ is separable from } E' \text{ by } \mathcal{S}.$$

We extend this terminology to automata, so that we can also say, for example, that A and A' are inclusion-equivalent or exclusion-equivalent if their languages are.

In the remainder of this section, we prove basic properties about inclusion- and exclusion equivalent languages. The properties hold for general languages, so they do not require regularity of I or E.

▶ **Lemma 3.** *Let $\mathcal{S}$ be a class of separators. Let language I be inclusion-equivalent to I' w.r.t. $\mathcal{S}$ and E be exclusion-equivalent to E' w.r.t. $\mathcal{S}$. Then, for each $S \in \mathcal{S}$,*

$$S \text{ separates } I \text{ from } E \text{ iff } S \text{ separates } I' \text{ from } E'.$$

We are not aware of any work that defines equivalences between languages up to separability, but we note that a similar notion appears in Place et al. [15]. They defined an equivalence for separability of single words by a k-piecewise testable language, i.e., a boolean combination of k-subsequences. Additionally they mentioned that, for a given k, there exists a smallest k-piecewise testable language that contains a given regular language L. However, without a restriction on k, a smallest piecewise testable language containing L, i.e., a canonical piecewise testable approximation for L, does not exist.

We believe that it is useful to think about equivalences between languages. Due to non-existence of a smallest piecewise testable language equivalent to a given language L, we do not have any nice characterisation for equivalences with respect to boolean combinations. However, we provide characterizations for the weaker classes of separators we consider in this paper. The characterizations apply to more general notions than subsequences or subwords. More precisely, they hold for quasi-orders (preorder) on words.

▶ **Definition 4.** A *quasi-order* is a binary relation which is transitive and reflexive. A *well-quasi-order* is a quasi order $\lesssim$ such that any infinite sequence of elements $x_0, x_1, x_2, \ldots$ contains an increasing pair $x_i \lesssim x_j$ with $i < j$.

We present the lemmas for quasi-orders here, but readers only interested in subsequences or subwords can simply think about the subsequence (resp., subword) ordering $\preccurlyeq$ ($\trianglelefteq$) when reading them. There is one exception, however. For Lemma 6 we need a *well*-quasi-orders. It is well-known that $\preccurlyeq$ is a well-quasi-order (due to Higman's lemma) but $\trianglelefteq$ is not. Indeed, for the infinite sequence $x_n = 10^n1$ (for increasing values of n), there is no $i < j$ such that x_i is a subword of x_j. However, as we will see later, the lemmas do apply for k-subsequences and k-subwords.

Let $\lesssim$ be a quasi-order on words over alphabet Σ. For a word $w \in \Sigma^*$, the *$\lesssim$-language induced by w* is the language $\{u \in \Sigma^* \mid w \lesssim u\}$, that is, the set of all words u that are at least as large as w with respect to $\lesssim$. We usually leave the word w implicit and say that a language is a $\lesssim$-language if there exists such a word w. Subsequence and subword languages are examples of $\lesssim$-languages.

▶ **Lemma 5.** *For every quasi-order $\lesssim$ on words, the following are equivalent for languages E and E':*
(a) *E is exclusion-equivalent to E' w.r.t. $\lesssim$-languages;*
(b) *E is exclusion-equivalent to E' w.r.t. unions of $\lesssim$-languages;*
(c) *E is exclusion-equivalent to E' w.r.t. intersections of $\lesssim$-languages;*
(d) *E is exclusion-equivalent to E' w.r.t. positive combinations of $\lesssim$-languages; and*
(e) *$\forall_{w\in E}\exists_{w'\in E'}$ such that $w \lesssim w'$ and $\forall_{w'\in E'}\exists_{w\in E}$ such that $w' \lesssim w$.*

We now turn to a similar characterisation as Lemma 5 for inclusion-equivalence. For inclusion-equivalence, however, the characterisation is no longer the same for all positive combinations. For example, $\{ab, aa\}$ is inclusion-equivalent to $\{a\}$ with respect to subsequence languages and intersections thereof, but not with respect to unions or positive combinations of subsequence languages.

We characterize these two cases in the following two lemmas.

▶ **Lemma 6.** *Let $\lesssim$ be a well-quasi-order on words. The following are equivalent for languages I and I':*
(a) *I is inclusion-equivalent to I' w.r.t. unions of $\lesssim$-languages;*
(b) *I is inclusion-equivalent to I' w.r.t. positive combinations of $\lesssim$-languages;*
(c) *$\text{closure}^{\lesssim}(I) = \text{closure}^{\lesssim}(I')$; and*
(d) *For every $\lesssim$-minimal element $i \in I$ there is a $\lesssim$-minimal element in $i' \in I'$ such that $i \lesssim i'$ and $i' \lesssim i$.*
Where $\text{closure}^{\lesssim}(I)$ is the set of all words that are larger or equal with respect to $\lesssim$ than some word from I.

▶ **Lemma 7.** *For every quasi-order $\lesssim$ on words, the following are equivalent for languages I and I':*
(a) *I is inclusion-equivalent to I' w.r.t. $\lesssim$-languages;*
(b) *I is inclusion-equivalent to I' w.r.t. intersections of $\lesssim$-languages;*
(c) *$\cap_{w\in I}\{u \in \Sigma^* : u \lesssim w\} = \cap_{w'\in I'}\{u \in \Sigma^* : u \lesssim w'\}$*

We now argue that Lemmas 5–7 can be used in the context of k-subword and k-subsequence languages. To this end, for $k \in \mathbb{N}$ and words w_1, w_2, we define

$$w_1 \preccurlyeq_k w_2 \text{ if and only if } k\text{-Subseqs}(w_1) \subseteq k\text{-Subseqs}(w_2).$$

Similarly, we say that $w_1 \trianglelefteq_k w_2$ if k-Subwords$(w_1) \subseteq k$-Subwords(w_2). Since $\preccurlyeq_k$ and $\trianglelefteq_k$ are well-quasi orders for every $k \in \mathbb{N}$, we have that Lemmas 5–7 apply to k-subsequences and k-subwords as well.

2.4 Witnesses for Non-Separability

We provide simple characterizations of non-separability that state, in each of the cases, what kind of witness one should search for proving non-separability. We found these characterizations to be quite useful in proofs.

▶ **Lemma 8.** *Let I and E be two regular languages. Then I is not separable from E*
(a) *by a union of k-subsequence languages iff there exists a word $w_I \in I$ that can not be separated from E, i.e., such that every k-subsequence s of w_I appears in some word $w_s \in E$.*
(b) *by an intersection of k-subsequence languages iff there exists a word $w_E \in E$ that cannot be separated from I, i.e., such that every k-subsequence that appears in every word from I also appears in w_E.*

(c) *by a positive combination of k-subsequence languages iff there exists words $w_I \in I$ and $w_E \in E$ such that k-Subseqs$(w_I) \subseteq k$-Subseqs(w_E).*

(d) *by a boolean combination of k-subsequences iff there exist words w_I in I and w_E in E such that k-Subseqs$(w_I) = k$-Subseqs(w_E).*

Item (d) from the lemma is rather standard and follows almost immediately from the definition of k-piecewise testable languages by Simon [19, 20]; see also Lemma 4.1 in [17].

The corresponding lemma for k-subword languages is analogous. The different cases in the lemma give a good idea of the different flavors of the separation problem when one searches for a witness of non-separability. For example, in case (4), we are looking for words in I and E that have precisely the same sets of k-subsequences. It is usually much harder to argue that such witnesses are small than in, say, case (3) where only inclusion in one direction is required.

3 A Tractable Case

The main result of this section is a tractability result of separability of I from E by k-subsequences, by finite unions, intersections, and positive combinations thereof, if a certain condition holds on E. The main idea is that we reduce E to a small language E' which is exclusion-equivalent and solve the separation problem of I and E'. Afterwards, we generalize this to two more expressive form of separators, namely *k-subsequences of constant-length words*, and finite unions thereof. Here, for a constant c, a k-subsequence of c-length words is a language of the form

$$\Sigma^* w_1 \Sigma^* \cdots \Sigma^* w_\ell \Sigma^*$$

where $\ell \leq k$ and each w_i has length at most c. We think that such languages could be helpful to separate languages in practice, because they seem to be rather expressive, potentially simple to understand, yet have a tractable separation problem.

We need a little bit of terminology to get started. We call a set $X \subseteq Q$ of states of an automaton $A = (Q, \Sigma, \delta, q_0, F)$ a *strongly connected component*, or *SCC*, if it is a maximal strongly connected component in the usual graph representation of A. Let $X \subseteq Q$ be a set of states of an automaton $A = (Q, \Sigma, \delta, q_0, F)$. We say that A' is obtained from A by *collapsing* X if A' is the image of A under a homomorphism $g : Q \to (Q \setminus X) \uplus \{q_X\}$ which is the identity on $Q \setminus X$ and maps each $u \in X$ to q_X. Furthermore, q_X is accepting in A' if and only if X contains an accepting state.

▶ **Lemma 9.** *For a given automaton A let X be one of its SCCs. If A' is obtained from A by collapsing X, then A and A' are exclusion-equivalent with respect to subsequence languages, unions of subsequence languages, intersections of subsequence languages, and positive combinations.*

Notice that Lemma 9 is an easy corollary of Lemma 5. Furthermore, if A and A' are exclusion-equivalent with respect to subsequence languages then, for every fixed $k \in \mathbb{N}$, they are also exclusion-equivalent with respect to the (less expressive) k-subsequence languages. (Similarly for unions, intersections, and positive combinations thereof.) Therefore, Lemma 9 relativises to k-subsequence languages.

The following notion, a *core* of an NFA, will be central in our search for tractable separation. Basically, we will obtain tractability for separation of languages for which we can find a small approximation of its core. The overall idea is similar to the idea of *kernelization* in the context of finding fixed-parameter tractability, but, as far as we know, our approach does not necessarily lead to a proof of fixed-parameter tractability of the general problem.

▶ **Definition 10.** An NFA C is a *core* of a language E, if the following hold:

1. E and C are exclusion-equivalent w.r.t. positive combinations of subsequences,
2. C is minimal among all NFAs of exclusion-equivalent languages to E w.r.t. positive combinations of subsequences.

So, the rationale of a core C is that it captures the whole complexity of E when it comes to separation. When we want to decide whether I is separable from E, we can obtain the correct result by deciding separability of I from C instead. The challenge is to be able to compute a core (or sufficiently small approximation thereof) from the NFA of E. The following observation gives us a first step.

▶ **Observation 11.** *Each core of a regular language E contains no loops other than self-loops.*

Proof. If a core C is not a DAG with self-loops then it contains a non-trivial SCC. Thus, according to Lemma 9 we can collapse the non-trivial SCC and obtain a smaller core, which contradicts the minimality of C. ◀

Hence, an initial approximation of a core of E can be obtained by collapsing all its SCCs. We will see later that cores can be even smaller in general.

Observe that, in general, computing a core is at least NP-hard, because minimizing an automaton which is a DAG is NP-hard [22]. Therefore we will search for approximations of cores that we know how to compute efficiently.

3.1 Core-Approximations

Given an NFA $A = (Q, \Sigma, \delta, q_0, F)$, the following procedure computes an exclusion-equivalent NFA C' that may be much smaller than A and therefore can be seen as an approximation of a core.

For each SCC $X \subseteq Q$ we collapse X. As collapsing SCCs does not change the exclusion-equivalence class of an NFA, the obtained DAG D with self-loops is exclusion-equivalent to A (Lemma 9). In a next step we collapse further to obtain a possibly even smaller exclusion-equivalent NFA:

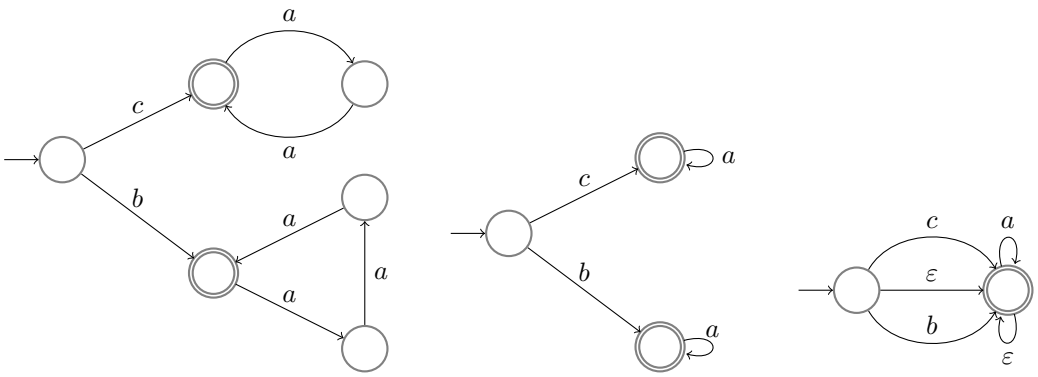
- First, for every transition $q_1 \xrightarrow{a} q_2$ in D, we add a transition $q_1 \xrightarrow{\varepsilon} q_2$, thereby obtaining an ε-NFA D_ε. Notice that, according to Lemma 5, languages D_ε and D are exclusion-equivalent w.r.t. (k-)subsequence languages, unions thereof, intersections thereof, and positive combinations thereof.
- Second, we take a *weak bisimulation quotient* of D_ε. This step does not change the language, so the exclusion-equivalence class trivially remains the same.

Bisimulation quotients are probably the simplest and best known heuristic to minimize an NFA [13]. *Weak* bisimulation is simply the version of the bisimulation quotient that takes ε-transitions into account. Weak bisimulation gives us a better refinement of the automaton D_ε than the ordinary bisimulation quotient. We illustrate a core-approximation in Figure 1.

We refer to the resulting automaton as a *core-approximation* of A. The subsequent results in this section imply tractability of the separation problems whenever we can compute a constant-size core-approximation.

3.2 Using Core-Approximations to Separate

We explain how an arbitrary language E' that is exclusion-equivalent to E can be used to separate I from E. In particular, the exposition applies to cores and core-approximations.



(a) Automaton $A(E)$ **(b)** After collapsing SCCs **(c)** After the second reduction

Figure 1 Illustration of a core-approximation.

In the cases of separability by k-subsequences and combinations thereof that have unions, the number of states of E, and therefore also the number of states of E's cores, gives an upper bound for the length of the subsequences that we need to consider for separation.

▶ **Lemma 12.** *If automata I and E are separable by a k-subsequence language (resp., union of k-subsequence languages, or positive combination of k-subsequence languages), then they are separable by an $|E|$-subsequence language (resp., union of $|E|$-subsequence languages, or positive combination of $|E|$-subsequence languages).*

These bounds can be used to obtain the following upper bounds on separability by k-subsequence languages, intersections, unions, and positive combinations thereof. In the Lemma, f denotes a function and *poly* denotes a polynomial function.

▶ **Lemma 13.** *For a given automata I, E and a number k we can decide if I is separable from E*

(a) *by a k-subsequence language in time $O(poly(|I|) \cdot \Sigma \cdot f(|E|))$;*
(b) *by an intersection of k-subsequence languages in time $O(poly(|I|) \cdot |\Sigma|^{|E|+1} \cdot f(|E|))$;*
(c) *by a union of k-subsequence languages in time $O(poly(|I|) \cdot f(|E|))$; and*
(d) *by a positive combination of k-subsequence languages in time $O(poly(|I|) \cdot f(|E|))$.*

Proof sketch. In the proof, f is an exponential function in cases (a) and (b) and double exponential in (c) and (d), but we stress that we have not yet attempted any optimization. All our algorithms are based on exhaustive checking of witnesses that fulfil certain constraints; we provide a sketch for case (a).

(a) By Lemma 12 we know that we can bound k by $|E|$. Let X be the set of words u that have length at most k and such that there is no $v \in E$ such that $u \preccurlyeq v$. We can separate I from E if and only if there is a word $s \in X$ that is a subsequence of every word in I. Without loss of generality we can restrict ourselves to minimal words in X with respect to the $\preccurlyeq$ order. Indeed, if $w \preccurlyeq w'$ and closure$^{\preccurlyeq}(w')$ separates I from E then closure$^{\preccurlyeq}(w)$ is also a separator. The number of such minimal words is bounded by $|\Sigma_E|^k + |\Sigma|$, where Σ_E is the alphabet used in E. So, a naive algorithm could simply enumerate all $|\Sigma_E|^k + |\Sigma|$ such minimal words and test the conditions. Testing if such a word w does not have a supersequence in E can be done in polynomial time by computing from w the DFA A_w of Observation 2 and testing if its intersection with E is empty. Testing if w is a subsequence of every word in I can be

answered by testing if I is included in $L(A_w)$. Since A_w is deterministic, this can be done in time polynomial in $|I|$ and the length of w too. Thus, we get that the overall complexity is bounded by $(p_1(|I|) + p_2(|E|)) \cdot (\Sigma + |\Sigma_E|^{|E|})$, where p_1 and p_2 are polynomials. ◀

From Lemma 13, we immediately get tractability of separability if we can find a core-approximation of E that has constant size. This is trivial, for example, if E has a constant number of SCCs.

▶ **Theorem 14.** *For a given automata I, E and a number k, if the core approximation of E has constant size, we can decide in PTIME if I is separable from E by*

(a) *a k-subsequence language,*
(b) *an intersection of k-subsequence languages,*
(c) *a union of k-subsequence languages, and*
(d) *a positive combination of k-subsequence languages.*

3.3 Sequences of Words

We now generalize the algorithm for Theorem 14 to deal with more expressive separators that combine subsequences and subwords. For k and c in $\mathbb{N}$, a language of *k-subsequences of c-words* is a language of the form $\Sigma^* w_1 \Sigma^* \cdots \Sigma^* w_k \Sigma^*$ where each w_i has length at most c. In the remainder of this section, k should be thought of as an input to the separability problem as before, and c should be thought of as a constant. Our aim is to show that, if c is constant, then we can extend Theorem 14 to languages of k-sequences of c-words.

The idea consists of doing a preprocessing step on the NFA for E and then performing an analogous construction as in the previous section. Essentially, the preprocessing step consists of extending E's alphabet such that it also reads c-tuples of Σ-symbols.

More formally, let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA. By $A^{\leq c} = (Q, \Sigma^{\leq c}, \delta^{\leq c}, q_0, F)$ we denote the NFA obtained from A as follows:

- $\Sigma^{\leq c} := \uplus_{1 \leq i \leq c} \Sigma^i$
- for every $a \in \Sigma$ and $q \in Q$, $\delta^{\leq c}(q, a) := \delta(q, a)$
- for every $(a_1, \ldots, a_i) \in \Sigma^{\leq c}$ and $q \in Q$, $\delta^{\leq c}(q, (a_1, \ldots, a_i))$ is the set of states that can be reached in A by reading the word $a_1 \cdots a_i$, that is, $\cup_{p \in \delta^{\leq c}(q, (a_1, \ldots, a_{i-1}))} \delta(p, a_i)$.

That is, $A^{\leq c}$ behaves exactly the same as A but, whenever it reads a tuple $(a_1, \ldots, a_i)$ it behaves as if A would read the word $a_1 \cdots a_i$.

When we have given automata I and E, we can use the core-approximation of $E^{\leq c}$ to separate I from E by languages of k-subsequences of c-words. Here, we construct the core-approximation of $E^{\leq c}$ (w.r.t. k-subsequences over $\Sigma^{\leq c}$) as explained in Section 3.1.

▶ **Theorem 15.** *For a given automata I, E, a number k, and a constant c, if the core approximation of $E^{\leq c}$ has constant size, it is decidable in PTIME if I is separable from E by*

- *a language of k-subsequence of c-words, or*
- *a union of languages of k-subsequences of c-words.*

The proof of the Theorem is obtained by minor adaptations in the proof of Theorem 14 (a) and (c) to deal with the different alphabet of $E^{\leq c}$. (To this end, we also need a slightly different version of Lemma 12, adapted to the new alphabet. Its proof is analogous.)

We conclude this section by remarking that the approach we used to show Theorem 15 does not naively generalize to separators that have intersection. Basically, when intersection is allowed, we cannot treat constant-length words as single symbols anymore. For example, when ab and ba are treated as single symbols in Σ, then $\Sigma^* ab \Sigma^* \cap \Sigma^* ba \Sigma^*$ does not contain

aba. When ab and ba are words of length two, then $aba \in \Sigma^*ab\Sigma^* \cap \Sigma^*ba\Sigma^*$. So, the naive application of the former algorithm may return the wrong result.

4 Separability by k-Subsequences

In Sections 4 and 5 we present the result of a systematical investigation of the complexity of separability in different constellations. More precisely, we consider all combinations of separating I from E where I and E can be a regular language or a word. As possible separators, in Section 4 we consider all combinations of k-subsequence languages that we also considered before and, in Section 5 we consider the same combinations of k-subword languages. We note that many complexity bounds are not yet tight.

The complexity landscape in this section shows separability by k-subsequences is often hard, even if one of the languages is a singleton word. More precisely, if we restrict either I or E (but not both) to be a single word, separation seems to remain NP- or coNP-hard in almost all cases.[2] Only when both I and E are words, we can prove that separability by k-subsequence languages and combinations thereof is in PTIME. On the positive side, almost all upper bounds lie within Π_2^P which is lower than the typical PSPACE bound which one expects for many static analysis problems for NFAs such as universality and containment.

▶ **Theorem 16.** *Given NFAs for I and E, and a number k, separability of I from E*
(a) *by k-subsequence languages is NP-complete;*
(b) *by unions of k-subsequence languages is NP-hard and in Π_2^P;*
(c) *by intersections of k-subsequence languages is coNP-hard and in Π_2^P; and*
(d) *by positive combinations of k-subsequence languages is coNP-complete.*
(e) *by a boolean combinations of k-subsequence languages is coNP-hard and in NEXPTIME.*

All hardness results already follow from Theorem 17, where I is a singleton. In cases (a) and (b), we have reductions from SAT that use an acyclic NFA for E. However, the proof requires non-determinism in the NFA. For (c), (d), and (e), we have several reductions. One is from the Hamilton path problem and shows the problem is hard even if the automaton for E acyclic, but it makes linearly many copies of the input graph for the Hamilton problem. The other is from a problem investigated by Wood [25], doesn't produce an acyclic automaton, but has a shorter proof, which we present here. Hardness already holds for $k = 1$, which is a sharp contrast with the PTIME upper bounds in Theorems 14 and 15.

4.1 Restricted Cases

We now consider severely restricted cases in which at least one of the two languages is restricted to be a single word. If we restrict I to be a single word, then we see that all separability problems become coNP-complete when the separator languages have intersection, and NP-complete otherwise.

▶ **Theorem 17.** *Given word w_I, an NFA for E, and a number k, separability of w_I from E*
(a) *by k-subsequence languages or by unions of k-subsequence languages is NP-complete;*
(b) *by intersections, positive combinations, or boolean combinations of k subsequence languages is coNP-complete.*

[2] There is one notable exception in which we do not yet know the precise complexity: Theorem 18(2).

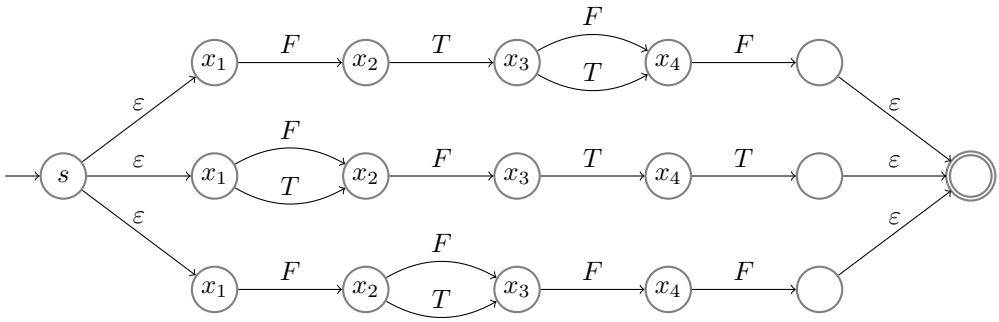


Figure 2 Construction of ε-NFA for E in the proof of Theorem 17(a).

Proof sketch. We sketch the lower bound proofs. The hardness proof for (a) is by reduction from SAT. For a given formula φ in conjunctive normal form, over the variables $\{x_1, \ldots, x_k\}$, we construct w_I and E such that φ is satisfiable if and only if w_I can be separated from E by a k-subsequence language.

More precisely, E is defined over the alphabet $\{T, F\}$ and contains those words of length k that correspond to valuations that do not satisfy the formula φ. Valuations are encoded as words in a straightforward manner: a word $X_1 \cdots X_k$ with each $X_i \in \{T, F\}$ encodes the assignment that assigns x_i true if and only if $X_i = T$. The construction of the automaton to recognize E is in paragraph below. The word w_I is defined as $(TF)^k$. Since w_I contains every k-subsequence over $\{T, F\}$, the word w_I can be separated from the language E by a k-subsequence if and only if $E \neq (T + F)^k$ by Lemma 8. This is equivalent to the fact that there is a valuation which satisfies the formula φ.

Clearly, k and w_I can be constructed in polynomial time from φ. It remains to show how to construct an NFA for E. Let $\varphi = C_1 \wedge \cdots \wedge C_m$. This NFA is a union of sub-automata that accept those words of length k that encode valuations that do *not* satisfy C_i. Figure 2 contains and ε-NFA E for the formula $\varphi = (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4)$. Notice that this ε-NFA for E and thus also an NFA for E can be constructed in polynomial time.

The coNP hardness for (b) is almost immediate from the NP-hardness of the following problem [25]: Given a DFA A over alphabet Σ, is there a word in $L(A)$ that contains every letter in Σ?

Indeed, given the DFA A over $\Sigma = \{a_1, \ldots, a_n\}$, we can define $w_I = a_1 \cdots a_n$ and $E = L(A)$ and ask if there is an intersection of languages of the form $\Sigma^* a_i \Sigma^*$ that separates w_I from E. Notice in this case, if there is a separator then the intersection $\cap_{i=1}^{n}(\Sigma^* a_i \Sigma^*)$ of *all* such languages is a separator as well. So, w_I is separable from E by an intersection of 1-subsequence languages if and only if it is separable by $\cap_{i=1}^{n}(\Sigma^* a_i \Sigma^*)$. This means that w_I is separable from E if and only if no word in $L(A)$ contains every letter from Σ.

Given the proof for intersections, observe that if there exists a separator that uses positive boolean combinations, namely of a form $X_1 \cup X_2 \cup \cdots \cup X_n$ then one of of X_i separates w_I from E as well. Thus the problem for positive boolean combination is equivalent to the problem for intersection and is coNP-hard as well. Finally, in the solution of the above problem, negation does not help, as w_E contains all letters from Σ. Thus solving the boolean combination problem is equivalent to solving the intersection problem as well. ◀

The converse case in which we restrict E to be a single word shows a similar picture, but note that we do not have a coNP lower bound or PTIME upper bound for separability

by unions or positive combinations. For unions, for example, we need to decide if there exists a word $w_I \in I$ such that all its k-subsequences appear in w_E (Lemma 8). One can easily construct a polynomial-size DFA for all k-subsequences that do not appear in w_E, but in general there is no small DFA for all words that *contain* a k-subsequence that does not appear in w_E. (The intuition is that such an automaton needs to guess which symbols to use for the subsequence.) Therefore, checking if all words in I contain some k-subsequence that does not appear in w_E seems to be difficult.[3] Conversely, for a hardness proof, the fact that w_E is a single word gives little freedom for encoding gadgets.

▶ **Theorem 18.** *Given word w_E, an NFA for I, and a number k, separability of I from w_E*
(a) *by k-subsequence languages or by intersections of k-subsequence languages is NP-complete;*
(b) *by unions or positive combinations of k-subsequence languages is in coNP; and*
(c) *by boolean combinations of k-subsequence languages is coNP-complete.*

Hardness in case (a) is by reduction from the longest common subsequence problem. For (c), the question is equivalent to Theorem 17(2) because the separators are closed under complement.

Finally, when both languages are restricted to be a single word, separability can be decided in PTIME by using standard automata techniques (Observation 1).

▶ **Theorem 19.** *Given words w_I, w_E, and number k, separability of w_I from w_E by k-subsequence languages or by unions, intersections, positive combinations, or boolean combinations thereof is in PTIME.*

5 Separability by k-Subwords

Analysis of subwords provides us a similar overview like in subsequence case, but the complexities are more diverse. We see more cases that are tractable, but the arguments why this is so are rather easy. The main reason why, in some cases, separability by k-subwords seems to be easier than by subsequences is because a given word w can only have $O(k|w|)$ many subwords, whereas it can have exponentially many subsequences. That said, subwords can also be used to reason about the distance between positions in a word. This can be exploited to encode corridor tiling and which makes separability by boolean combinations of k-sequences PSPACE-hard.

▶ **Theorem 20.** *Given NFAs for I and E, and a number k, separability I from E*
(a) *by k-subword languages is in PTIME;*
(b) *by unions or positive combinations of k-subword languages is PSPACE-complete;*
(c) *by intersections of k-subword languages is coNP-complete; and*
(d) *by boolean combinations of k-subword languages is in NEXPTIME and PSPACE-hard.*

5.1 Restricted Cases

When we restrict one of the languages to be a word, we see that separability becomes coNP-complete at worst.

[3] If I contains a polynomial number of shortest words, it can be done in polynomial time due to Theorem 19.

▶ **Theorem 21.** *Given word w_I, an NFA for E, and a number k, separability of w_I from E*
(a) *by k-subword languages or by unions of k-subword languages is in PTIME; and*
(b) *by intersections, positive combinations, or boolean combinations of k subword languages is coNP-complete.*

▶ **Theorem 22.** *Given word w_E, an NFA for I, and a number k, separability of I from w_E*
(a) *by k-subword languages or by unions, intersections, or positive combinations thereof is in PTIME; and*
(b) *by boolean combinations of k-subword languages is coNP-complete.*

Finally, for the sake of completeness, we mention that, when both languages are a word, separation is trivially in PTIME.

▶ **Theorem 23.** *Given words w_I, w_E, and number k, separability of w_I from w_E by k-subword languages or by unions, intersections, positive combinations, or boolean combinations thereof is in PTIME.*

6 Separability by k-Prefixes and k-Suffixes

For the sake of completeness, we mention that deciding separability by combinations of prefixes of length up to k is in polynomial time in all cases we consider. A *k-prefix language* is a language of the form $w\Sigma^*$ where $|w| \leq k$. All boolean combinations of k-prefix languages are defined similarly as before.

▶ **Observation 24.** *Given NFAs for I and E, and a number k, separation of I from E by k-prefix languages, or by unions, intersections, positive combinations, and boolean combinations thereof is in PTIME.*

The proofs are rather straightforward adaptations of the corresponding proofs in [5]. Naturally, the observation also holds for k-suffix languages by a reduction that simply reverses the NFAs for I and E.

7 Conclusions

Separation of regular languages seems to be a worthwhile approach to investigate for tackling several problems in graph databases, for example, approximating regular path queries (by specifying a query to be approximated and a second query of paths that should not be matched), computing explanations of the results of regular path queries, and for computing explanations of the differences between edge-labeled s-t-graphs in general.

When one searches for separators to provide explanations, it does not really matter to a user which class of separators is considered, as long as the separator is simple enough to understand and interpret. In fact, a system is likely to be perceived to be much more useful and intelligent when it is able to return simple specimens from many classes of separators, as opposed to complex specimens that come from a limited number of classes. Intuitively, the former case corresponds to a situation where explanations can be very diverse and, in the latter case explanations always have a similar flavor. In this paper we investigated to which extent subsequences, subwords, and combinations thereof can be used to describe the difference between regular languages (or graphs) I and E. Our main motivation for considering subsequences and subwords are that we feel that they are easy to understand and, at the same time, may be reasonably expressive. It was already shown in [5, 15] that, for regular languages, deciding the existence of an arbitrarily complex separator involving

subsequences, prefixes or suffixes is in PTIME. Here we made a step towards finding simpler separators in the sense that we considered a given bound k on the length of the subwords and subsequences involved. We showed that, if E can be reduced to a sufficiently small substructure, its core-approximation, there is good hope that the difference between I and E can be described in simple terms, if they can be separated.

Efficient construction of separators is clearly the goal of this line of research and is a challenging problem. We would like to understand when separators exist, when they are small, and when they are efficiently computable. This paper gives us better understanding of this case, which can then serve as a basis towards producing separators that are, ultimately, human readable.

Acknowledgments. We would like to thank the anonymous reviewers of ICDT 2015 for many helpful remarks. This work was supported by DFG grant MA4938/2-1.

References

1 E. Botoeva, R. Kontchakov, V. Ryzhikov, F. Wolter, and M. Zakharyaschev. Query inseparability for description logic knowledge bases. In *Principles of Knowledge Representation and Reasoning (KR)*, 2014.

2 J. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973.

3 P. Buneman and W. C. Tan. Provenance in databases. In *International Conference on Management of Data (SIGMOD)*, pages 1171–1173, 2007.

4 W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3), 1957.

5 W. Czerwinski, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *International Conference on Automata, Languages and Programming (ICALP)*, pages 150–161, 2013.

6 T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Principles of Programming Languages (POPL)*, pages 232–244, 2004.

7 E. Kopczynski and A. Widjaja To. Parikh images of grammars: Complexity and applications. In *Logic in Computer Science (LICS)*, pages 80–89, 2010.

8 C. Lutz and F. Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 989–995, 2011.

9 T. Masopust and M. Thomazo. On k-piecewise testability (preliminary report). *CoRR*, abs/1412.1641, 2014.

10 K. L. McMillan. Applications of craig interpolants in model checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 1–12, 2005.

11 R. McNaughton. Algebraic decision procedures for local testability. *Mathematical Systems Theory*, 8(1):60–76, 1974.

12 R. McNaughton and S. Papert. *Counter-free automata*. The M.I.T. Press, 1971.

13 R. Paige and R. Tarjan. Three parition refinement algorithms. *SIAM Journal on Computing*, 16:973–989, 1987.

14 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 363–375, 2013.

15 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Mathematical Foundations of Computer Science (MFCS)*, pages 729–740, 2013.

16 T. Place and M. Zeitoun. Separating regular languages with first-order logic. In *Computer Science Logic – Logic in Computer Science (CSL-LICS)*, 2014.

17 L. Van Rooijen. *Une approche combinatoire du problème de séparation pour les langages réguliers.* PhD thesis, Université de Bordeaux, 2014.

18 S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *International Conference on Management of Data (SIGMOD)*, pages 1579–1590, 2014.

19 I. Simon. *Hierarchies of Events with Dot-Depth One*. PhD thesis, Dept. of Applied Analysis and Computer Science, University of Waterloo, Canada, 1972.

20 I. Simon. Piecewise testable events. In *Proceedings of GI Conference on Automata Theory and Formal Languages*, pages 214–222. Springer, 1975.

21 J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.

22 L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *Symposium on Theory of Computing (STOC)*, pages 1–9, 1973.

23 W. C. Tan. Provenance in databases: Past, current, and future. *IEEE Data Engineering Bulletin*, 30(4):3–12, 2007.

24 Š. Holub, G. Jirśková, and T. Masopust. On upper and lower bounds on the length of alternating towers. In *Mathematical Foundations of Computer Science (MFCS), Part I*, pages 315–326, 2014.

25 P. T. Wood. Containment for XPath fragments under DTD constraints. In *International Conference on Database Theory (ICDT)*, 2003. Full version, obtained through personal communication.

26 Y. Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972.

Process-Centric Views of Data-Driven Business Artifacts*

Adrien Koutsos[1] **and Victor Vianu**[2]

1 **ENS Cachan, France**
adrien.koutsos@ens-cachan.fr
2 **UC San Diego & INRIA-Saclay**
vianu@cs.ucsd.edu

Abstract

Declarative, data-aware workflow models are becoming increasingly pervasive. While these have numerous benefits, classical process-centric specifications retain certain advantages. Workflow designers are used to development tools such as BPMN or UML diagrams, that focus on control flow. Views describing valid sequences of tasks are also useful to provide stake-holders with high-level descriptions of the workflow, stripped of the accompanying data. In this paper we study the problem of recovering process-centric views from declarative, data-aware workflow specifications in a variant of IBM's business artifact model. We focus on the simplest and most natural process-centric views, specified by finite-state transition systems, and describing regular languages. The results characterize when process-centric views of artifact systems are regular, using both linear and branching-time semantics. We also study the impact of data dependencies on regularity of the views.

1998 ACM Subject Classification H.2.3 Languages: Query languages, H.4.1 Office Automation: Workflow management, B.4.4 Performance Analysis and Design Aids: Formal models, Verification

Keywords and phrases Workflows, data-aware, process-centric, views

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.247

1 Introduction

Data-driven workflows have become ubiquitous in a variety of application domains, including business, government, science, health-care, social networks [37], crowdsourcing [7, 8], etc. Such workflows resulted from an evolution away from the traditional process-centric approach towards data-awareness. A notable exponent of this class is the *business artifact model* pioneered in [31, 25], deployed by IBM in professional services. Business artifacts (or simply "artifacts") model key business-relevant entities, which are updated by a set of services that implement business process tasks. This modeling approach has been successfully deployed in practice [4, 3, 9, 14, 39]. In particular, the Guard-Stage-Milestone (GSM) approach [12, 23] to artifact specification uses declarative services with pre- and post-conditions, parallelism, and hierarchy. The OMG standard for Case Management Model and Notation (CMMN) [5], announced last year, draws key foundational elements from GSM [28].

Declarative, high-level specification tools such as GSM allow to generate the application code from the high-level specification. This not only allows fast prototyping and improves programmer productivity but, as a side effect, provides new opportunities for automatic

* This work was partially supported by the National Science Foundation under award IIS-1422375.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 247–264

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

verification. Indeed, the high-level specification is a natural target for verification, as it addresses the most likely source of errors (the application's specification, as opposed to the less likely errors in the automatic generator's implementation). This has spawned an entire line of research seeking to trace the boundary of decidability of properties of such systems, expressed in variants of temporal logic (see [18]).

While declarative specifications of workflows have many benefits, they also come with some drawbacks. Workflow designers are used to traditional process-centric development tools, such as BPMN (Business Process Model and Notation), workflow nets, and UML activity diagrams, that focus on control flow while under-specifying or ignoring the underlying data. Such process-centric views of workflows are also useful to provide stakeholders with customized descriptions of the workflow, tailored to their role in the organization. For example, an executive may only require a high-level summary of the business process. Descriptions of the workflows as sequences of tasks stripped of data are intuitive and often sufficient for many users. Thus, recovering the sequencing of tasks from declarative specification is often desirable.

In this paper, we study views of business artifact runs consisting of the sequences of services applied in each run[1]. This comes in two flavors, depending on whether we are interested in linear runs alone, or in the more informative branching-time runs. We call these the linear-time, resp. branching-time (service) views of the artifact system. The simplest and most intuitive specification mechanism for such views consists of a finite-state transition system, which can describe both ω-regular languages (for linear-time views), and regular infinite trees of runs (for branching-time views).

▶ **Example 1.** To illustrate linear-time service views of declarative workflows, we consider a variant of the running example of [11]. In the example, the customer can choose a product, a shipment method and apply a coupon to the order. The order is filled in a sequential manner as is customary on e-commerce web-sites. After the order is filled, the system awaits for the customer to submit a payment. If the payment matches the amount owed, the system proceeds to shipping the product. At any time before submitting a valid payment, the customer may edit the order (select a different product) an unbounded number of times. The linear-time service view of this workflow consists of the sequences of service names that occur in all infinite runs of the system, and is specified by the finite-state transition system shown in Figure 1. A more informative view, that captures the services applied in branching-time runs of the system, is shown in Figure 2. Intuitively, the branching-time view captures *precisely* which services may be applied *at each point* in a run. To understand the difference with linear-time views, consider the states labeled *edit coupon* in Figures 1 and 2 (the latter highlighted in red). In the linear-time view specification, there is only one such state, in which two actions can be taken: *no coupon* and *apply coupon*. However, the two actions are not *always* applicable whenever the state *edit coupon* is reached. If no product has an applicable coupon, the only action possible is *no coupon*. If for all products and shipping method there is some applicable coupon, then both *no coupon* and *apply coupon* are applicable. Finally, if some products have applicable coupons and others do not, then both of the above cases may occur depending on the choice of product. The different cases are captured by distinct states in the branching-time specification, while the information is lost in the linear-time specification. Of course, the *sequences* of service names occurring in all runs of the system are the same in both specifications.

[1] In various formalizations of business artifacts, tasks are called *services*. As usual in program verification, we take runs to be infinite, because many business processes run forever and because infinite runs capture information lost by finite prefixes.

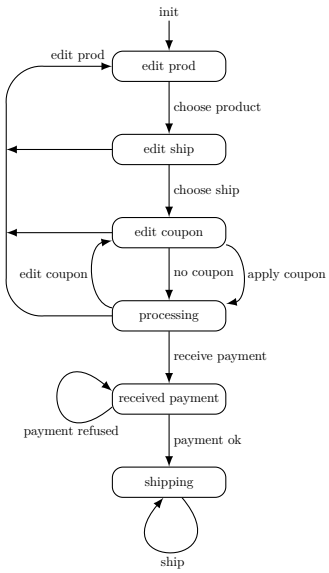


Figure 1 A linear-time process-centric view.

The main results of the paper establish under what circumstances the linear-time or branching-time service views are regular (with effective specifications). We consider the tuple artifact model used in [17, 11], and several natural restrictions derived from the GSM design methodology, some of which have been considered in the context of verification [11]. We also consider the impact of database dependencies on regularity of the views. We show that linear-time service views of tuple artifacts are ω-regular, but branching-time views are not regular. We then consider artifacts obeying a natural restriction previously used in the context of verification, called *feedback freedom* [11]. We show that branching-time views of feedback-free artifacts are still not regular, but become so under a slightly stronger restriction called *global feedback freedom*. This restriction is naturally obeyed by specifications resulting from the GSM hierarchical design methodology.

It turns out that there is a tight connection between the result on view regularity and the verification of artifact systems. Properties to be verified are specified using extensions of the classical temporal logics LTL, CTL and CTL*, in which propositions are interpreted as existential FO formulas on the tuple artifact and the underlying database. This yields the logics LTL-FO, CTL-FO and CTL*-FO [17, 11, 20]. It can be shown that regularity of the linear or branching-time service views for a class of artifact systems, with effectively constructible specifications, implies decidability of LTL-FO, resp. CTL*-FO properties for that class. The converse is false: decidability of LTL-FO or CTL*-FO properties may hold even though the corresponding service views may not be regular. Thus, regularity is a stronger result than decidability of verification. Indeed, our results imply that CTL*-FO properties are decidable for globally feedback-free artifact systems. On the other hand, we show that CTL-FO properties are undecidable for feedback-free artifact systems (also implying non-regularity of their branching-time views).

The proof techniques developed here have additional side-effects beneficial to verification. In our previous approaches to automatic verification of business artifacts [17, 11], given an artifact specification $\mathcal{A}$ and an LTL-FO property φ, the verifier either certifies satisfaction of the property or produces a counter-example run on some database $D(\mathcal{A}, \varphi)$ depending on both $\mathcal{A}$ and φ. In contrast, the techniques of the present paper allow to show that,

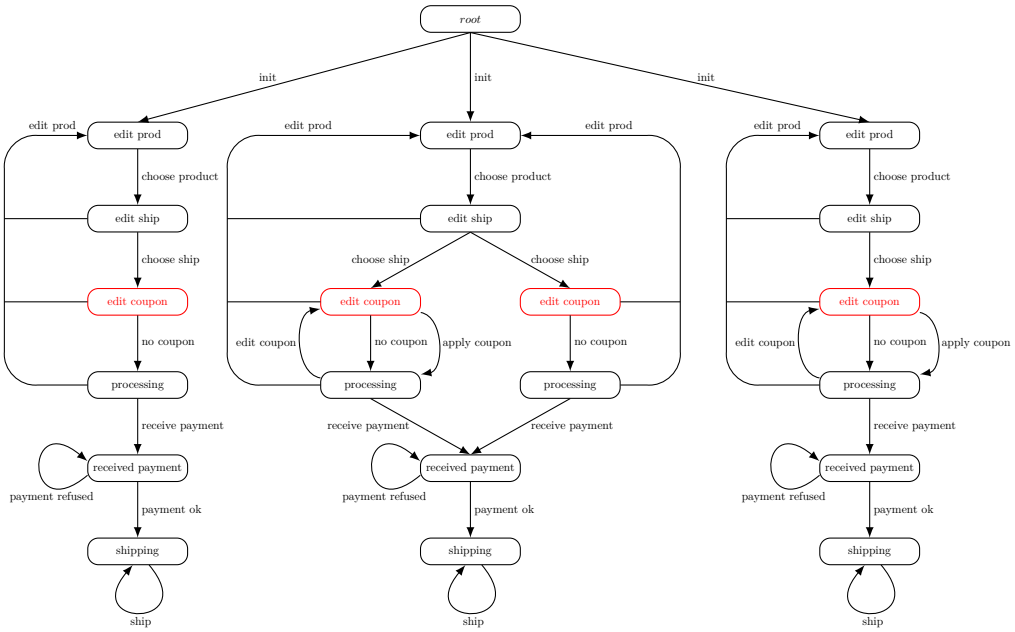


Figure 2 A branching-time process-centric view.

for specifications and properties without constants, there exists a *single* database $D(\mathcal{A})$, depending only on $\mathcal{A}$, which serves as a universal counter-example for *all* LTL-FO properties violated by $\mathcal{A}$. Pre-computing such a database may allow more efficient generation of counter-examples for specific LTL-FO properties.

Decidability results on verification of branching-time properties of infinite-state artifact systems are scarce and require significant restrictions. In [22, 13], decidability results are shown for properties expressed in an FO extension of μ-calculus, under restrictions on the artifact system orthogonal to ours. In [2], the artifact model is extended to a multi-agent setting, and decidability results are shown for properties expressed in an FO extension of CTL that can also refer to each agent's knowledge using a Kripke-style semantics. Decidability of similar FO extensions of CTL is shown in [27] for the case when the database consists of a single unary relation.

2 Background

After some basic definitions, we present the tuple artifact model, the languages LTL-FO and CTL$^{(*)}$-FO, and review previous results on verification of LTL-FO properties.

We assume an infinite data domain **dom**. A *relational schema* is a finite set of relation symbols with associated arities. An *database instance* over a relational schema $\mathcal{DB}$ is a mapping I associating to each $R \in \mathcal{DB}$ a *finite* relation over **dom** with the same arity as R (denoted $arity(R)$). We assume familiarity with first-order logic (FO) over relational schemas (e.g., see [1, 26]). FO formulas may use equality and constants from **dom**.

Artifact systems. In the tuple artifact model, an artifact consists of a finite set of variables whose values evolve during the workflow execution. Transitions are specified declaratively, using pre-and-post conditions that may query an underlying database.

▶ **Definition 2.** An *artifact schema* is a a tuple $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ where $\bar{x}$ is a finite sequence $x_1, \ldots, x_k$ of *artifact variables* and $\mathcal{DB}$ is a relational schema.

For each $\bar{x}$, we also define a set of variables $\bar{x}' = \{x' \mid x \in \bar{x}\}$ where each x' is a distinct new variable. In a transition, a primed variable represents the value of the variable in the new artifact.

▶ **Definition 3.** An *instance* of an artifact schema $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ is a tuple $A = \langle \nu, D \rangle$ where ν is a valuation of $\bar{x}$ into **dom** and D is a finite instance of $\mathcal{DB}$.

▶ **Definition 4.** A *service* over an artifact schema $\mathcal{A}$ is a pair $\sigma = \langle \pi, \psi \rangle$ where:
- $\pi(\bar{x})$, called *pre-condition*, is a quantifier-free[2] FO formula over $\mathcal{DB}$ with variables $\bar{x}$;
- $\psi(\bar{x}, \bar{x}')$, called *post-condition*, is a quantifier-free FO formula over $\mathcal{DB}$ with variables $\bar{x}, \bar{x}'$.

▶ **Definition 5.** An *artifact system* is a triple $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$, where $\mathcal{A}$ is an artifact schema, Σ is a non-empty set of services over $\mathcal{A}$, and Π is a pre-condition restricting the value of the initial artifact variables (as above, a quantifier-free FO formula over $\mathcal{DB}$, with variables $\bar{x}$).

▶ **Definition 6.** Let $\sigma = \langle \pi, \psi \rangle$ be a service over an artifact schema $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$, and let D be an instance over $\mathcal{DB}$. Let ν, ν' be valuations of $\bar{x}$. We say that ν' is a *possible successor* of ν w.r.t. σ and D (denoted $\nu \xrightarrow{\sigma} \nu'$ when D is understood) iff:
- $D \models \pi(\nu)$, and
- $D \models \psi(\nu, \nu')$.

Note that $\psi(\bar{x}, \bar{x}')$ need not bind $\bar{x}'$ to the database, so ν' may contain values not in D.

▶ **Definition 7.** Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. A *run* of Γ on database instance D over $\mathcal{DB}$ is an infinite sequence $\rho = \{(\rho_i, \sigma_i)\}_{i \geq 0}$, where for each $i \geq 0$, ρ_i is a valuation of $\bar{x}$, $\sigma_i \in \Sigma$, $\rho_i \xrightarrow{\sigma_i} \rho_{i+1}$, and $D \models \Pi(\rho_0)$.

The assumption that runs are infinite is not a restriction, since finite runs can be artificially extended to infinite runs by adding a self-looping transition. The linear-time semantics of an artifact system Γ is provided by the set of all runs. Specifically, we denote by $\mathit{Runs}_D(\Gamma)$ the set of all runs of Γ on database instance D, and by $\mathit{Runs}(\Gamma)$ the union of $\mathit{Runs}_D(\Gamma)$ for all databases D. The more informative branching-time semantics is provided by its *tree of runs*, additionally capturing all choices of transitions available at each point in the run. More precisely, $\mathit{TRuns}_D(\Gamma)$ is a labeled tree whose nodes are all finite prefixes of runs of Γ on D, such that the children of a prefix are all prefixes extending it by one transition. Each node ρ is labeled by the value of the last artifact tuple in ρ, and each edge from ρ to $\rho.(\nu, \sigma)$ is labeled by σ. The global tree of runs $\mathit{TRuns}^*(\Gamma)$ is the tree obtained by placing all trees $\mathit{TRuns}_D(\Gamma)$ (for every database D) under a common root, connected by edges with a special label *init*. The tree of runs allows to define properties in branching-time temporal logic, such as "any client has the option of canceling a purchase at any time". Note that such a property is not captured by the linear runs in $\mathit{Runs}(\Gamma)$.

Temporal properties of artifact systems. Temporal properties are specified using extensions of LTL (linear-time temporal logic) and CTL$^{(*)}$ (branching-time temporal logics). We begin with LTL. Recall that LTL is propositional logic augmented with temporal operators

[2] ∃FO conditions can be easily simulated by additional artifact variables.

G (always), **F** (eventually), **X** (next) and **U** (until) (e.g., see [32]). Informally, **G**p says that p holds at all times in the run, **F**p says that p will eventually hold, **X**p says that p holds at the next configuration, and p**U**q says that q will hold at some point in the future and p holds up to that point. For example, **G**$(p \rightarrow \mathbf{F}q)$ says that whenever p holds, q must hold sometime in the future.

The extension of LTL used in [11], called[3] LTL-FO, is obtained from LTL by replacing propositions with quantifier-free FO statements about particular artifact records in the run. The statements use the artifact variables and the database. In addition, they may use *global* variables, shared by different statements and allowing to refer to values in different records. The global variables are universally quantified over the entire property. We illustrate LTL-FO with a simple example.

▶ **Example 8.** The following specifies a desirable business rule for an e-commerce workflow with artifact variables including `amount_paid`, `amount_refunded`, `status`, `amount_owed` (with obvious meaning).

$$(\varphi)\ \forall x \mathbf{G}((\texttt{amount\_paid} = x \wedge \texttt{amount\_paid} = \texttt{amount\_owed}) \rightarrow \mathbf{F}(\texttt{status} = \text{"shipped"} \vee \texttt{amount\_refunded} = x))$$

Property φ states that if a correct payment is submitted then at some time in the future either the product is shipped or the customer is refunded the correct amount. Note the use of universally-quantified variable x to relate the value of paid and refunded amounts across distinct steps in the run sequence.

The branching-time extensions CTL-FO and CTL*-FO are defined analogously from CTL and CTL*. Recall that CTL* augments LTL with path quantifiers **A** (for all) and **E** (exists) while CTL restricts the use of path quantifiers so that they are always followed by a temporal operator (see [21]).

We note that variants of LTL-FO have been introduced in [21, 35]. The use of globally quantified variables is also similar in spirit to the *freeze quantifier* defined in the context of LTL extensions with data by Demri and Lazić [15, 16].

Verification of artifact systems. We informally review the results of [17, 11] on verification of LTL-FO properties of artifact systems. Classical model-checking applies to finite-state transition systems. Checking that an LTL property holds is done by searching for a counterexample run of the system. The finiteness of the transition system is essential and allows to decide property satisfaction in PSPACE using an automata-theoretic approach (see e.g. [10, 29]). In contrast, artifacts are infinite-state systems because of the presence of unbounded data. The main idea for dealing with the infinite search space is to explore the space of runs of the artifact system using *symbolic* runs rather than actual runs. This yields the following result.

▶ **Theorem 9.** *[17] It is* PSPACE*-complete to check, given an artifact system $\mathcal{A}$ and an LTL-FO property φ, whether $\mathcal{A}$ satisfies φ.*

Unfortunately, Theorem 9 fails even in the presence of simple data dependencies or arithmetic. Specifically, as shown in [17, 11], verification becomes undecidable as soon as

[3] The variant of LTL-FO used here differs from some previous ones in that the FO formulas interpreting propositions are quantifier-free. By slight abuse we use here the same name.

the database is equipped with at least one key dependency, *or* if the specification of the artifact system uses simple arithmetic constraints allowing to increment and decrement by one the value of some atributes. Hence, a restriction is needed to achieve decidability for these extensions. We discuss this next.

To gain some intuition, consider the undecidability of verification for artifact systems with increments and decrements. The proof of undecidability is based on the ability of such systems to simulate *counter machines*, for which the problem of state reachability is known to be undecidable [30]. To simulate counter machines, an artifact system uses an attribute for each counter. A service performs an increment (or decrement) operations by "feeding back" the incremented (or decremented) value into the next occurrence of the corresponding attribute. To simulate counters, this must be done an unbounded number of times. To prevent such computations, a restriction is imposed in [11], called *feedback freedom*, designed to limit the data flow between occurrences of the same artifact variable at different times in runs that satisfy the desired property. The formal definition considers, for each run, a graph capturing the data flow among variables, and imposes a restriction on the graph. Intuitively, paths among different occurrences of the same variable are permitted, but only as long as each value of the variable is independent on its previous values. This is ensured by a condition that takes into account both the artifact system and the property to be verified, called *feedback freedom*. It turns out that artifact systems designed in a hierarchical fashion by successive refinement, in the style of the Guard-Stage-Milestone approach [12, 23], naturally satisfy the feedback freedom condition [19]. Indeed, there is evidence that the feedback freedom condition is permissive enough to capture a wide class of applications of practical interest. This is confirmed by numerous examples of real-life business processes modeled as artifact systems, encountered in IBM's practice [11].

Feedback freedom turns out to ensure decidability of verification in the presence of arithmetic constraints, and also under a large class of data dependencies including key and foreign key constraints on the database.

▶ **Theorem 10.** *[11] It is decidable, given an artifact system* Γ *using arithmetic (linear inequalities over* $\mathbb{Q}$*) and whose database satisfies a set of key and foreign key constraints, and an LTL-FO property* φ *such that* (Γ, φ) *is feedback free, whether every run of* $\mathcal{A}$ *on a valid database satisfies* φ*.*

Unfortunately, the complexity is non-elementary with respect to the number of artifact variables.

3 Linear-time service views

In this section, we define linear-time service views of artifact systems and establish their regularity. Throughout the paper, all results establishing regularity are effective, in the sense that specifications can be effectively constructed.

We begin by recalling some basics on languages on infinite words. Let Δ be a finite alphabet. An ω-word over Δ is an infinite sequence of symbols from Δ and an ω-language is a set of ω-words. An ω-language is ω-regular if it is accepted by a *Büchi automaton* (e.g., see [36]). A Büchi automaton is a non-deterministic finite-state automaton accepting the infinite words for which some run of the automaton goes through an accepting state infinitely often. A Büchi automaton in which every state is accepting is also referred to as a finite-state transition system (or safety automaton, see [24]). Thus, a finite-state transition system defines the ω-words for which there is some non-blocking run of the system with transitions labeled by the symbols of the word.

Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. For each run $\rho = \{(\rho_i, \sigma_i)\}_{i \geq 0}$ of Γ, the service view of ρ, denoted $\mathcal{S}(\rho)$, consists of the ω-word $(\sigma_i)_{i \geq 0}$. The linear-time service view of Γ is $\mathcal{S}_{lin}(\Gamma) = \{\mathcal{S}(\rho) \mid \rho \in \mathit{Runs}(\Gamma)\}$. We say that $\mathcal{S}_{lin}(\Gamma)$ is *effectively* ω-regular for a class of artifact systems if a Büchi automaton defining $\mathcal{S}_{lin}(\Gamma)$ can be effectively constructed from each Γ in that class.

We will show that $\mathcal{S}_{lin}(\Gamma)$ is effectively ω-regular for artifact systems. To do so, we will use symbolic representations of the runs of Γ. Let C be the set of constants used in Γ. To each $x \in \bar{x}$ we associate an infinite set of new variables $\{x_i\}_{i \geq 0}$, and we denote $\bar{x}_i = \{x_i \mid x \in \bar{x}\}$. An *equality type* for variables $\bar{y}$ is an equivalence relation on $\bar{y} \cup C$ in which no distinct constants in C are equivalent. An isomorphism type of $\bar{y}$ is a pair (H, ϵ) where ϵ is an equality type for $\bar{y}$ and H is an instance of $\mathcal{DB}$ using elements in $\bar{y} \cup C$ that is consistent with ϵ.

▶ **Definition 11.** A symbolic run ϱ of Γ is a sequence $\{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ such that, for each $i \geq 0$:

- **(i)** (H_i, ϵ_i) is an isomorphism type of $\bar{x}_i \cup \bar{x}_{i+1}$,
- **(ii)** the pre-condition Π of Γ holds in the restriction of (H_0, ϵ_0) to $\bar{x}_0$,
- **(iii)** (H_i, ϵ_i) and $(H_{i+1}, \epsilon_{i+1})$ agree on $\bar{x}_{i+1}$,
- **(iv)** the pre-condition of σ_i holds in the restriction of (H_i, ϵ_i) to $\bar{x}_i$, and
- **(v)** the post-condition of σ_i holds in (H_i, ϵ_i).

We denote by $\mathit{SRuns}(\Gamma)$ the set of symbolic runs of Γ.

It is easy to see that each run of Γ has a corresponding symbolic run, representing the consecutive isomorphism types of the artifact tuples in the run. We make this more precise. Let $\approx$ be the transitive closure of $\cup_{i \geq 0} \epsilon_i$. Clearly, $\approx$ is an equivalence relation on $\cup_{i \geq 0} \bar{x}_i \cup C$. It is easily seen that ϵ_i is the restriction of $\approx$ to $\bar{x}_i \cup \bar{x}_{i+1} \cup C$. Let $[x_i]_{\epsilon_i}$ and $[x_i]_{\approx}$ denote the equivalence class of x_i in ϵ_i, resp. $\approx$.

▶ **Definition 12.** Let $\{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ be a symbolic run of Γ. An *enactment* of ϱ is a triple (D, ρ, θ) where D is a database over $\mathcal{DB}$, $\rho = \{(\rho_i, \sigma_i)\}_{i \geq 0}$ is a run of Γ over D, and θ is a mapping from $(\cup_{i \geq 0} \bar{x}_i)$ to **dom** such that, for each $i \geq 0$:

- $\theta(x_i) = \rho_i(x)$ for every $x \in \bar{x}$,
- if $y, z \in \bar{x}_i \cup \bar{x}_{i+1}$ and $(y, z) \in \epsilon_i$ then $\theta(y) = \theta(z)$
- the function $\hat{\theta}$ defined by $\hat{\theta}([y]_{\epsilon_i}) = \rho(y)$ for $y \in \bar{x}_i \cup \bar{x}_{i+1}$ is an isomorphism from H_i/ϵ_i to $D|(\rho_i \cup \rho_{i+1})$.

▶ **Lemma 13.** *For every database D over $\mathcal{DB}$ and run ρ of Γ over D there exists a symbolic run ϱ of Γ and a mapping h from $(\cup_{i \geq 0} \bar{x}_i)$ to **dom** such that (D, ρ, h) is an enactment of ϱ.*

Proof. The run ϱ can be easily constructed by induction from ρ. ◀

Consider the converse of Lemma 13: does every symbolic run have an enactment on some database? This is much less obvious. It is easy to construct, for each symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$, a triple (D_ϱ, ρ, h) satisfying the definition of enactment except for the finiteness of D_ϱ. This can be done as follows. The (infinite) domain of D_ϱ simply consists of all equivalence classes of $\approx$, h maps each x_i to $[x_i]_{\approx}$, the relations are interpreted as $(\cup_{i \geq 0} H_i)/_{\approx}$, and ρ is the image of ϱ under h. However, it is far less clear that a *finite* database D_ϱ exists with the same properties. Intuitively, different classes of $\approx$ must be merged in order to obtain a finite domain, and this must be done consistently with the H_i's. We are able to show the following key result.

▶ **Theorem 14.** *Every symbolic run* $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ *of* Γ *has an enactment* $(D_\varrho, \rho, \theta)$ *where the size of* D_ϱ *is exponential in* $|\bar{x}|$.

Proof. The roadmap of the proof is the following. We first define a normal form for artifact systems, called *linear propagation* form, requiring that the only equalities in pre-and-post conditions of services be of the form $x = x'$ where $x \in \bar{x}$ (excepting equalities with constants). We show that for every artifact system Γ we can construct an artifact system $\bar{\Gamma}$ in linear-propagation form, and a mapping h from the symbolic runs of Γ to symbolic runs of $\bar{\Gamma}$, such that from every enactment of $h(\varrho)$ one can construct an enactment of ϱ. Finally, we show that every symbolic run of an artifact system in linear-propagation form has an enactment. This is done as follows. Consider an artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ with $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ in linear propagation form. Recall that for a symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ of Γ, we denote by $\approx$ the transitive closure of $(\cup_{i\geq 0}\epsilon_i)$ and by $[x_i]_\approx$ the class of x_i wrt $\approx$. Note that, because Γ is in linear propagation form, we can assume that $[x_i]_\approx$ contains only variables x_j. We define $span([x_i]_\approx) = \{j \mid x_j \approx x_i\}$. Clearly, each span is an interval. Next, for $x \in \bar{x}$ we define $lane(x) = \{[x_i]_\approx \mid i \geq 0\}$ (totally ordered by $[x_i]_\approx \leq [x_j]_\approx$ iff $i \leq j$). The remainder of the proof consists of defining certain finite characteristics of equivalence classes of $\approx$, so that different classes in a given lane can be collapsed if they share the same characteristics. This yields an enactment of the symbolic run over the finite database whose elements are the collapsed classes. We omit the rather involved technical details. ◀

We can now show the regularity of the service view of Γ. From Lemma 13 and Theorem 14 it follows that $\mathcal{S}_{lin}(\Gamma) = \{\mathcal{S}(\varrho) \mid \varrho \in SRuns(\Gamma)\}$. We can construct a finite-state transition system $\mathcal{F}(\Gamma)$ accepting the latter as follows:

- States: the isomorphism types (H, ϵ) of $\bar{x} \cup \bar{x}'$,
- Initial states: the states whose restrictions to $\bar{x}$ satisfy Π
- Transitions: $(H, \epsilon) \xrightarrow{\sigma} (\bar{H}, \bar{\epsilon})$ if (H, ϵ) satisfies items $(iv) - (v)$ of Definition 11 for service σ and $(H, \epsilon)|\bar{x}'$ and $(\bar{H}, \bar{\epsilon})|\bar{x}$ are identical modulo renaming $\bar{x}'$ to $\bar{x}$.

The ω-language accepted by $\mathcal{F}(\Gamma)$ consists of the sequences of transition labels in all infinite runs of the system starting from some initial state. By construction, this is precisely $\{\mathcal{S}(\varrho) \mid \varrho \in SRuns(\Gamma)\}$. Thus, we have the following.

▶ **Theorem 15.** $\mathcal{S}_{\text{lin}}(\Gamma)$ *is effectively* ω*-regular for artifact systems* Γ.

Verification vs. ω-regularity. We note that the effective ω-regularity of $\mathcal{S}_{lin}(\Gamma)$ implies decidability of LTL-FO properties of artifact systems, but is strictly stronger. Decidability of verification follows from ω-regularity because for each Γ and LTL-FO property $\xi = \forall \bar{y} \varphi_f$, and each choice of isomorphism type for $\bar{y}$, one can construct an artifact system Γ_{φ_f} and an LTL formula $\bar{\varphi}$ such that $\Gamma \models \varphi_f(\bar{y})$ iff $\mathcal{S}_{lin}(\Gamma_{\varphi_f}) \models \bar{\varphi}$. Essentially, Γ_{φ_f} is obtained, for a fixed choice of $\bar{y}$, by augmenting the artifact variables and pre-and-post conditions of each service of Γ in order to record the truth values of the FO-components of φ_f in each transition. Since a finite-state transition system specifying $\mathcal{S}_{lin}(\Gamma_{\varphi_f})$ can be effectively constructed, this reduces verification to classical finite-state LTL model-checking, and yields decidability. The converse is falsified by results of [34] which imply that artifact systems equipped with a total order do not have ω-regular service views, yet verification of LTL-FO properties is decidable.

Universal test databases. The above results, notably Theorem 14, have some potentially significant benefits for verification. We can show the following rather surprising result.

▶ **Theorem 16.** *Let Γ be a constant-free artifact system with k artifact variables. One can construct a database D^* of size double exponential in $|\bar{x}|$ such that for every constant-free LTL-FO formula ξ over Γ, $\Gamma \models \xi$ iff $Runs_{D^*}(\Gamma) \models \xi$.*

Proof. Consider an LTL-FO formula ξ over Γ. As shown in [11] (Lemma 3.3), global variables can be easily eliminated, so one can assume that $\xi = \varphi_f$. Let $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ be a symbolic run of Γ. Satisfaction of ξ by ϱ is defined similarly to actual runs, by evaluating each FO component of φ_f on the consecutive H_i/ϵ_i. Consider an enactment (D, ρ, θ) of ϱ, where $\rho = \{(\rho_i, \sigma_i)\}_{i\geq 0}$. Because H_i/ϵ_i and $D|(\rho_i \cup \rho_{i+1})$ are isomorphic, it is clear that $\varrho \models \xi$ iff $\rho \models \xi$. This in conjunction with Lemma 13 and Theorem 14 shows that $\Gamma \models \xi$ iff every symbolic run of Γ satisfies ξ. Because each symbolic run has an enactment on some database of size exponential in k, $Runs^k(\Gamma) = \cup\{Runs_D(\Gamma) \mid |D| \leq exp(k)\}$ are enactments of all symbolic runs of Γ. Thus, $\Gamma \models \xi$ iff all runs in $Runs^k(\Gamma)$ satisfy ξ. Suppose Γ and ξ are constant free. There are finitely many non-isomorphic databases of size bounded by $exp(k)$, and let D^* be their disjoint union. Clearly, $\Gamma \models \xi$ iff all runs over D^* satisfy ξ. The size of D^* is double exponential in k. ◀

Thus, D^* acts as a universal test database (akin to an Armstrong relation) for satisfaction of constant-free LTL-FO properties of Γ. In particular, a fixed D^* can be pre-computed and used to generate a counter-example run for *every* constant-free LTL-FO property violated by Γ. In contrast, the counter-example databases constructed by the algorithms in [17, 11] depend on both the specification and property. Note that, if Γ and ξ use some set C of constants, then constructing a single universal test database is no longer possible: one needs a separate database for each isomorphism type over C. Constructing the most concise test databases possible, and evaluating the practical benefits, are important issues yet to be explored.

4 Branching-Time Service Views

In this section we consider branching time service views of artifact systems. Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. Recall the branching-time semantics of Γ, given by $TRuns^*(\Gamma)$. The *branching-time service view* of Γ, denoted $\mathcal{TS}^*(\Gamma)$, is the tree obtained from $TRuns^*(\Gamma)$ by ignoring the content of the nodes and retaining only the service labels of the edges. We use the following definition of regularity for infinite trees: $\mathcal{TS}^*(\Gamma)$ is regular if it is isomorphic to the unfolding of a finite-state transition system with edge labels (equivalently, $\mathcal{TS}^*(\Gamma)$ has finitely many non-isomorphic subtrees). Analogously to the linear case, we say that $\mathcal{TS}^*(\Gamma)$ is *effectively regular* for a class of artifact systems if such a finite-state transition system can be effectively constructed from each Γ in that class.

As we shall see, it turns out that branching-time service views of artifact systems are generally *not* regular. One might hope that regularity holds for artifacts obeying the natural feedback-free property that has proven so effective in overcoming undecidability of LTL-FO properties for specifications with dependencies and arithmetic [11]. Unfortunately, this is not the case. Indeed, we show that even very simple CTL-FO properties are not decidable for feedback-free artifacts, thus implying that their branching time service views are not effectively regular.

▶ **Theorem 17.** *It is undecidable, given an artifact system Γ and a CTL-FO formula ξ such that (Γ, ξ) is feedback-free, whether $\Gamma \models \xi$.*

Proof. The proof is by a reduction from the Post Correspondence Problem (PCP) [33]. We build upon a result of [17] (Theorem 4.2) showing that checking LTL-FO properties is

undecidable for databases satisfying a functional dependency (FD). This uses a reduction from the PCP. Next, we note that satisfaction of FDs by the database can be expressed as a CTL-FO property. Using this, we wish to mimick the reduction from the PCP that works for databases with FDs. However, there is a glitch: LTL-FO model checking is *decidable* for feedback-free specifications and properties even in the presence of FDs. Thus, a direct reduction from the linear-time case is not possible. Instead, we show how feedback freedom can be circumvented collectively by different branches of the tree of runs while being obeyed by each individual branch, and use this to adapt the PCP reduction to the branching-time framework. ◀

Similarly to the linear-time case, it can be shown that effective regularity of $\mathcal{TS}^*(\Gamma)$ implies decidability of CTL$^{(*)}$-FO properties. We therefore have the following.

▶ **Corollary 18.** *$\mathcal{TS}^*(\Gamma)$ is not effectively regular for feedback-free*[4] *artifact systems Γ.*

Note that Corollary 18 does not exclude the possibility that $\mathcal{TS}^*(\Gamma)$ might be regular. However, it says that even if this holds, a transition system defining $\mathcal{TS}^*(\Gamma)$ *cannot* be effectively constructed from each Γ.

Intuitively, feedback-freedom is ineffective in the branching-time setting because the restriction can be circumvented collectively by different branches of the tree of runs while being obeyed by each individual branch. Fortunately, specifications resulting from hierarchical design methodologies such as the Guard-Stage-Milestone discussed earlier, satisfy a stronger restriction that holds in the global run, called *global feedback freedom*. In a nutshell, global feedback freedom extends feedback freedom by having the computation graph take into account connections among variables in the entire tree of runs rather than just individual branches. We omit the technical details. We will show the following.

▶ **Theorem 19.** *$\mathcal{TS}^*(\Gamma)$ is effectively regular for globally feedback-free artifact systems Γ.*

The proof requires some technical development, which we sketch in the remainder of the section. The basic idea is to show that there are only finitely many subtrees $TRuns_D(\Gamma)$ of $TRuns^*(\Gamma)$ up to bisimulation. Moreover, each is realized by a database of bounded size, depending only on $|\bar{x}|$. Since bisimilar trees have the same branching-time service views, this establishes the theorem.

We recall the standard notion of bisimulation. Two infinite trees $\mathcal{T}, \mathcal{T}'$ with labeled edges are bisimilar if there exists a relation $\sim$ from the nodes of $\mathcal{T}$ to those of $\mathcal{T}'$ such that: (i) $root(\mathcal{T}) \sim root(\mathcal{T}')$, (ii) if $\alpha \sim \alpha'$ and $\alpha \xrightarrow{\sigma} \beta$ then there exists β' such that $\alpha' \xrightarrow{\sigma} \beta'$ and $\beta \sim \beta'$, and (iii) if $\alpha \sim \alpha'$ and $\alpha' \xrightarrow{\sigma} \beta'$ then there exists β such that $\alpha \xrightarrow{\sigma} \beta$ and $\beta \sim \beta'$.

We now present the main steps in the proof. Let $\Gamma = \langle\langle \bar{x}, \mathcal{DB}\rangle, \Sigma, \Pi\rangle$ be an artifact system, with $|\bar{x}| = k$. As in the global feedback freedom definition, we assume that service pre-and-post conditions are in $CQ^\neg$ form (conjunctions of literals). For a service $\sigma = (\pi, \psi)$ we denote by $f_\sigma(\bar{x}, \bar{y})$ the formula $\pi(\bar{x}) \wedge \psi(\bar{x}, \bar{y})$.

▶ **Definition 20.** The set $\mathcal{T}_n$ of *n-types* of $\bar{x}$ is defined inductively as follows.

- $\mathcal{T}_0 = \{\ true\ \}$
- For $n \geq 0$, $\mathcal{T}_{n+1}$ consists of all formulas of the form

$$\bigwedge_{\sigma \in \Sigma_0} \bigwedge_{\tau \in \mathcal{T}_\sigma} \exists \bar{y}(f_\sigma(\bar{x}, \bar{y}) \wedge \tau(\bar{y}))$$

where $\emptyset \neq \Sigma_0 \subseteq \Sigma$ and $\emptyset \neq \mathcal{T}_\sigma \subseteq \mathcal{T}_n$.

[4] An artifact system Γ is feedback free if $(\Gamma, true)$ is feedback free.

Let D be a database over $\mathcal{DB}$ and ν be a valuation of $\bar{x}$. It is easy to check that, for every ν that labels some node in $TRuns_D(\Gamma)$, and each $n \geq 0$, there exists a unique strongest[5] $\tau_n \in \mathcal{T}_n$ such that $D, \nu \models \tau_n$. We denote the latter by $\tau_n(D, \nu)$. It is clear that $\tau_{n+1}(D, \nu) \rightarrow \tau_n(D, \nu)$ for every $n \geq 0$.

Note that all subtrees of $TRuns_D(\Gamma)$ rooted at node labeled ν are isomorphic. Let $TRuns^{\nu}_D(\Gamma)$ be any such subtree. We will show that the sequence of types $\{\tau_n(D, \nu) \mid n \geq 0\}$ provides sufficient information to determine $TRuns^{\nu}_D(\Gamma)$ up to bisimilarity (Lemma 22). Before however, we need the following key lemma.

▶ **Lemma 21.** *Let Γ be a globally feedback-free artifact system. There exists $b > 0$, depending only on $|\bar{x}|$, such that for every database D, tuple ν labeling a node in $TRuns_D(\Gamma)$ and $n \geq 0$, $\tau_n(D, \nu)$ is equivalent to an FO sentence of quantifier rank $\leq b$.*

Using the lemma, we can prove the following.

▶ **Lemma 22.** *Let ν_1, ν_2 be valuations of $\bar{x}$ labeling nodes in $TRuns_D(\Gamma)$. If $\tau_n(D, \nu_1) = \tau_n(D, \nu_2)$ for every $n \geq 0$ then $TRuns^{\nu_1}_D(\Gamma)$ and $TRuns^{\nu_2}_D(\Gamma)$ are bisimilar.*

From Lemma 21 it follows that for every D and reachable ν, there exists $N > 0$ such that $\tau_n(D, \nu) \equiv \tau_N(D, \nu)$ for every $n \geq N$. We denote $\tau_N(D, \nu)$ by $\tau^*(D, \nu)$ and call it the *type* of ν in D. Thus, $\tau^*(D, \nu)$ is equivalent to $\{\tau_n(D, \nu) \mid n \geq 0\}$. Observe that, by Lemma 21, there are finitely many tuple types. The set of all tuple types is denoted by $\mathcal{T}$.

Finally, we define database types as follows.

▶ **Definition 23.** The type of a database D is $\tau(D) = \{\tau^*(D, \nu) \mid D \models \Pi(\nu)\}$.

We have the following.

▶ **Lemma 24.** *Let D_1 and D_2 be databases over $\mathcal{DB}$ such that $\tau(D_1) = \tau(D_2)$. Then $TRuns_{D_1}(\Gamma)$ and $TRuns_{D_2}(\Gamma)$ are bisimilar.*

Note that, since there are finitely many tuple types, there are also finitely many database types. Since a database type can be written as the conjunction of finitely many tuple types, Lemma 21 also applies to database types, and each can be written as an $\exists^*$FO sentence. Let d be the maximum number of existential quantifiers in these sentences. Thus, there are finitely many equivalence classes of trees of database runs under bisimulation, and each has a representative $TRuns_D(\Gamma)$ for some database D whose domain is of size $\leq d$. Since trees of runs equivalent under bisimulation have the same branching-time service views, it follows that $\mathcal{TS}^*(\Gamma)$ is ω-regular, and a finite-state transition system defining it can be effectively constructed from Γ. This concludes the proof of Theorem 19.

▶ Remark. Theorem 19 continues to hold for artifact systems extended with arithmetic (e.g., linear constraints over $\mathbb{Q}$). To see this, augment $\mathcal{DB}$ with a finite set $\mathcal{C}$ of relation symbols with fixed interpretations as linear constraints over $\mathbb{Q}$, and let the data domain be $\mathbb{Q}$. The definition of global freedom applies, by treating the relation symbols in $\mathcal{C}$ as arbitrary relations, and Lemma 21 carries through. Also, satisfiability of a type involving mixed data and arithmetic relations can be effectively tested: the only interaction between the two is via equality types.

[5] With respect to implication.

As noted earlier, effective regularity of the branching-time service views of a class of systems generally implies decidability of its CTL*-FO properties. In order for this to hold, we must however extend the global feedback freedom restriction to pairs (Γ, φ) where Γ is an artifact system and φ a CTL*-FO property. Taking into account the property is done similarly to feedback-freedom. We can then show the following.

▶ **Theorem 25.** *It is decidable, given an artifact system Γ and a CTL*-FO formula φ such that (Γ, φ) is globally feedback free, whether $\Gamma \models \varphi$.*

Proof. From a globally feedback-free (Γ, φ) one can construct a globally feedback-free artifact system $\bar{\Gamma}$ and a CTL* formula $\bar{\varphi}$ such that $\Gamma \models \varphi$ iff $\mathcal{TS}^*(\bar{\Gamma}) \models \bar{\varphi}$. Since $\mathcal{TS}^*(\bar{\Gamma})$ is specified by a finite-state transition system effectively constructed from Γ and φ, the result follows. ◀

5 The impact of data dependencies

In this section we consider the impact of data dependencies on the regularity of service views. We consider tuple and equality generating dependencies. We briefly recall (see [1] for details) that an equality-generating dependency (EGD) is an FO sentence of the form $\forall\bar{x}(\varphi(\bar{x}) \rightarrow y = z)$, where φ is a conjunction of relational atoms and $y, z \in \bar{x}$. A tuple-generating dependency (TGD) is a sentence of the form $\forall\bar{x}(\varphi(\bar{x}) \rightarrow \exists\bar{z}\psi(\bar{x}, \bar{z}))$, where φ and ψ are conjunctions of relational atoms. If $\bar{z}$ is empty, the dependency is called *full*; if every atom in $\psi(\bar{x}, \bar{z})$ contains an existentially quantified variable in $\bar{z}$, it is called *embedded* (note that every TGD can be written as a conjunction of full and embedded TGDs). A set of TGDs is *acyclic* if the following graph is acyclic: the nodes are database relations and there is an edge from P to Q if P occurs in the body of a TGD whose head contains Q. Inclusion dependencies are instances of TGDs and functional dependencies (FDs) are EGDs.

Linear-time service views. We first consider the impact of EGDs. Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. For a set Δ of dependencies, $\mathcal{S}^{\Delta}_{lin}(\Gamma) = \{\mathcal{S}(\rho) \mid \rho \in \mathit{Runs}_D(\Gamma), D \models \Delta\}$. We say that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively regular for a class $\mathbf{A}$ of artifact systems and $\mathbf{D}$ of dependencies, if a Büchi automaton defining $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ can be effectively constructed from each Γ in $\mathbf{A}$ and set Δ of dependencies in $\mathbf{D}$.

We can show the following.

▶ **Theorem 26.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is not effectively ω-regular for artifact systems Γ and sets Δ of EGDs. Moreover, this holds even if Δ consists of a single FD.*

Proof. It can be shown that it is undecidable, given an artifact system Γ, a set Δ of EGDs, and a service σ, whether there exists a run of Γ on a database satisfying Δ in which service σ is used. This holds even if Δ consists of a single FD. The result follows. ◀

Note that, similarly to Corollary 18, Theorem 26 leaves open the possibility that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ might be ω-regular.

▶ Remark. One might wonder if $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ can be characterized by some natural extension of ω-regular languages. It turns out that Theorem 26 can be extended to any family $\mathcal{L}$ of ω-languages with the following properties: (i) $\mathcal{L}$ is closed under intersection with ω-regular languages, and (ii) emptiness of languages in $\mathcal{L}$ is decidable. This assumes a finite specification mechanism for languages in $\mathcal{L}$, and that (i) is effective, i.e. the specification of the intersection of a language in $\mathcal{L}$ with the ω-language defined by a Büchi automaton must be computable. One example of such $\mathcal{L}$ is the family of ω-context-free languages, defined by infinitary extensions of pushdown automata and context-free grammars (see [6, 36]).

We now consider TGDs. Rather surprisingly, the easy case is that of embedded TGDs.

▶ **Theorem 27.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular for artifact systems Γ and sets Δ of embedded TGDs.*

Proof. It is enough to show that every symbolic run ϱ of Γ has an enactment on a database satisfying Δ. Indeed, this implies that $\mathcal{S}^{\Delta}_{lin}(\Gamma) = \mathcal{S}_{lin}(\Gamma)$, thus establishing effective ω-regularity. Let ϱ be a symbolic run of Γ. By Theorem 14, ϱ has an enactment (D, ρ, θ). Let d be some domain value not occurring in D or the constants of Γ. Observe that an extension $\bar{D}$ of D satisfying Δ can be obtained by chasing D with the TGDs in Δ so that d is used as a witness to every existentially quantified variable in the head of a TGD. Since $\bar{D}$ is an extension of D, $(\bar{D}, \rho, \theta)$ is also an enactment of ϱ. ◀

For full TGDs we have the following.

▶ **Theorem 28.** *There exists an artifact system Γ and set Δ of full TGDs such that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is not ω-regular.*

Proof. Let the database schema of Γ consist of a binary relation R and Δ be the full TGD $\forall x \forall y \forall z\ (R(x, y) \wedge R(y, z) \rightarrow R(x, z))$, guaranteeing that R is transitive. Γ has one attribute variable x and two services *init* and *next*. The global precondition is $\neg R(0, 0) \wedge x = 0$ where 0 is a constant. The pre-condition of *init* is $x \neq 0$ and its post-condition is $x' = 0$. The pre-condition of *next* is *true* and its post-condition is $R(x, x') \wedge \neg R(x', x')$. Runs of Γ consist of stepping through R using *next*, starting from 0, using only elements which do not belong to a cycle, until *init* reinitializes x to 0 and the process is restarted. Since R is finite, $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ consists of all ω-words of the form $(next)^{n_1} \cdot init \cdot (next)^{n_2} \cdot init \cdots$ such that for each word there exists $N > 0$ for which $n_i \leq N$ for all $i \geq 1$. It is easy to see that this language, and therefore $\mathcal{S}^{\Delta}_{lin}(\Gamma)$, is not ω-regular. ◀

It turns out that effective ω-regularity is recovered for *acyclic* full TGDs.

▶ **Theorem 29.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular for artifact systems Γ and acyclic sets of full TGDs Δ.*

Proof. Recall the finite-state transition system $\mathcal{F}(\Gamma)$ used earlier to define $\mathcal{S}_{lin}(\Gamma)$. Its states consist of the isomorphism types of Γ, and edges are labeled by services. The same transition system can be viewed as defining the language $SRuns(\Gamma)$, by taking into account the isomorphism type of each state in addition to the edge labels.

Consider Δ. A *partial unfolding* of a TGD is obtained by replacing one relational atom $R(\bar{z})$ in its body by the body of any TGD in Δ with R in its head (if such exists), with appropriate renaming of variables. Let Δ^* by the closure of Δ under partial unfoldings. Obviously, Δ^* and Δ are equivalent. Because Δ is acyclic, Δ^* is finite.

The idea of the proof is to define a Büchi automaton $\mathcal{B}$ that accepts the runs of $\mathcal{F}(\Gamma)$ that are *inconsistent* with some TGD in Δ^*. Using Δ^* instead of Δ facilitates this task by allowing to ignore compositions of TGDs. Let $\xi = \forall \bar{y}(\exists \bar{z} \varphi(\bar{y}, \bar{z}) \rightarrow R(\bar{y}))$ in Δ^*. An inconsistency with ξ occurs in a symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ if for some $j \geq 0$ and $\bar{y} \subseteq \bar{x}_j \cup \bar{x}_{j+1}$, $\neg R(\bar{y})$ is in H_j and there exist $\bar{z} \subseteq \cup_{i \geq 0} \bar{x}_i$ such that $\varphi(\bar{y}, \bar{z})$ is satisfied by $\cup_{i \geq 0} H_i$. It can be seen, using the construction in the proof of Theorem 14, that a symbolic run is consistent with Δ^* iff it has an enactment on a database satisfying Δ.

The Büchi automaton non-deterministically guesses an inconsistency. The first component of the inconsistency, $\neg R(\bar{y})$, can be guessed by $\mathcal{B}$ whenever $\neg R(\bar{y})$ is in H_j for the current j. To enable checking the second component of an inconsistency, the states of $\mathcal{B}$ also contain

variables $\bar{z}$. The values of the variables $\bar{z}$ are non-deterministically guessed throughout the run, and the connections between them, as specified by the isomorphism types, are recorded. A run is accepted whenever $\neg R(\bar{y})$ and $\varphi(\bar{y}, \bar{z})$ hold for some TGD and guessed $\bar{y}$ and $\bar{z}$. The set of symbolic runs consistent with Δ^* is then $SRuns(\Gamma) \cap \mathcal{B}^c$, where $\mathcal{B}^c$ is the complement of $\mathcal{B}$. Finally, $\mathcal{S}^{\Delta^*}_{lin}(\Gamma) = h(SRuns(\Gamma) \cap \mathcal{B}^c)$, where h is the homomorphism removing the isomorphism types and retaining just the service names. Since ω-regular languages are closed under complement, intersection, and homomorphism (with effective constructions), $\mathcal{S}^{\Delta^*}_{lin}(\Gamma)$ is effectively ω-regular. ◀

We finally consider feedback-free artifact systems. Recall that these are particularly well-behaved with respect to verification. In particular, while model-checking is undecidable for artifact systems in the presence of FDs, it becomes decidable for feedback-free systems [11]. One might hope that feedback-free systems are similarly well-behaved with respect to linear service views. Indeed, in contrast to Theorems 26 and 28, we have the following.

▶ **Theorem 30.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular for feedback-free artifact systems Γ and sets Δ of EGDs and full TGDs.*

Proof. The approach is similar to that of [11] for showing decidability of model-checking. Consider a symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ of Γ. For each $i \geq 0$, let $\nu_i(\bar{x}_i)$ be the formula $\exists \bar{x}_0 \ldots \exists \bar{x}_{i-1}(\Pi(\bar{x}_0) \wedge \bigwedge_{0 \leq j < i} \sigma_j(\bar{x}_j, \bar{x}_{j+1}))$. Intuitively, $\nu_i(\bar{x}_i)$ completely specifies the constraints placed on $\bar{x}_i$ by the first i transitions. Let $\Phi = \{\exists \bar{x}_i \nu_i(\bar{x}_i) \mid i \geq 0\}$. It can be shown that there exists an enactment of ϱ on a database D satisfying Δ iff $D \models \Phi \cup \Delta$ (this uses the finiteness of D and a pigeonhole argument). As shown in [11], because Γ is feedback-free, each formula in Φ can be rewritten as a formula of quantifier rank bounded by $|\bar{x}|^2$. Since there are finitely many non-equivalent formulas of bounded quantifier rank [26], Φ is equivalent to a single $\exists$FO formula φ. Moreover, because all formulas in Δ are universally quantified, if ϱ has an enactment on a database satisfying Δ, it also has an enactment on such a database whose domain is bounded by the number of variables (say v) in φ. Thus, $\mathcal{S}^{\Delta}_{lin}(\Gamma) = \cup\{\mathcal{S}_{lin}(Runs_D(\Gamma)) \mid D \models \Delta, |dom(D)| \leq v\}$. Since each $\mathcal{S}_{lin}(Runs_D(\Gamma))$ is ω-regular, $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular. ◀

Branching-time service views. We now consider briefly the impact of data dependencies on branching-time service views. Recall that these views are not regular, even for feedback-free systems. However, by Theorem 19, the views are regular for globally feedback-free systems.

Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. For a set Δ of dependencies over $\mathcal{DB}$, $TRuns^*_{\Delta}(\Gamma)$ is the tree obtained by placing all $TRuns_D(\Gamma)$ under a common root, where $D \models \Delta$. The branching-time service view, denoted $\mathcal{TS}^*_{\Delta}(\Gamma)$, is obtained as before from $TRuns^*_{\Delta}(\Gamma)$ by ignoring the content of the nodes and retaining only the service labels of the edges.

In the presence of data-dependencies, we have the following.

▶ **Theorem 31.** *$\mathcal{TS}^*_{\Delta}(\Gamma)$ is effectively regular for globally feedback-free artifact systems Γ and sets Δ of EGDs and full TGDs.*

Proof. Recall the proof of Theorem 19 and the formulas $\exists^*$FO defining database types, whose number of existential quantifiers is bounded by some b depending only on Γ. Note that the EGDs and full TGDs in Δ can be expressed by a sentence in $\forall^*$FO. Suppose there is a database D of type τ satisfying Δ. Then there exists $D_0 \subseteq D$, whose domain consists of b witnesses to the existentially quantified variables of τ, that also has type τ and satisfies Δ. Thus, every database type that includes an instance satisfying Δ, also has a

representative satisfying Δ whose domain is bounded by b. It follows that $\mathcal{TS}^*_\Delta(\Gamma)$ is regular, and a specification can be effectively constructed from Γ and Δ. ◀

▶ Remark. Theorem 31 alternatively holds for sets Δ of EGDs and arbitrary TGDs (full and embedded), as long as the set of TGDs is acyclic.

6 Conclusions

We considered the problem of extracting process-centric views from highly declarative, data-driven workflows. Classical process-centric workflow specification frameworks provide a variety of means for describing the valid sequences (or trees) of events in the workflow, with finite-state transition diagrams at their core. We considered views consisting of the sequences of services applied during linear or branching-time runs of an artifact system. The results establish when such views are regular and can be specified effectively by finite-state transition systems. Thus, we showed that linear-time service views are regular, while branching-time views are regular only under certain restrictions (satisfied naturally by systems produced by hierarchical design methodologies in the spirit of GSM). We also considered the impact of data dependencies (tuple and equality generating dependencies) on regularity of views. We showed that linear-time views are no longer regular in presence of FDs or cyclical full TGDs, but remain regular with acyclic or embedded TGDs. Regularity of branching-time service views is preserved in the presence of EGDs and full TGDs.

Our results also have some interesting connections to verification. For instance, the techniques developed to show regularity of linear-time views yield potentially more efficient ways to generate counterexample databases witnessing violation of LTL-FO properties. As a side-effect of results on branching-time service views, we showed that CTL-FO properties are undecidable for artifact systems, but model-checking CTL*-FO becomes decidable under the same restrictions guaranteeing regularity of branching-time views.

Several interesting questions remain to be investigated. If a class of declarative workflows does not have regular service views, two courses of action are plausible. First, one might seek an extension of regular languages powerful enough to describe the views while remaining palatable to users. Alternatively, one might opt for a regular approximation of the view, resulting from relaxations that users are likely to find reasonable. In all cases, the views could be made more expressive and informative by augmenting the purely process-centric specifications with light-weight annotations on transitions with conditions on the data, in the spirit of BPEL and YAWL [38]. Besides the technical problems *per se*, this brings into play interesting HCI and usability issues.

References

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

2 F. Belardinelli, A. Lomuscio, and F. Patrizi. An abstraction technique for the verification of artifact-centric systems. In *Proc. Intl. Conf. on Knowledge Representation*, 2012.

3 K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Sys. Journal*, 46(4), 2007.

4 K. Bhattacharya et al. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1), 2005.

5 BizAgi and Cordys and IBM and Oracle and SAP AG and Singularity (OMG Submitters) and Agile Enterprise Design and Stiftelsen SINTEF and TIBCO and Trisotech (Co-Authors). Case Management Model and Notation (CMMN), FTF Beta 1, Jan. 2013. OMG Document Number dtc/2013-01-01, Object Management Group.

6 L. Boasson and M. Nivat. Adherences of languages. *J. Comput. System Sci.*, 20(3), 1980.

7 A. Bozzon, M. Brambilla, S. Ceri, and A. Mauri. Reactive crowdsourcing. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 153–164, 2013.

8 A. Bozzon, M. Brambilla, S. Ceri, A. Mauri, and R. Volonterio. Pattern-based specification of crowdsourcing applications. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, pages 218–235, 2014.

9 T. Chao et al. Artifact-based transformation of IBM Global Financing: A case study. In *BPM*, 2009.

10 E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.

11 E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *ACM Transactions on Database Systems*, 37(3), 2012. Preliminary version in *ICDT 2011*.

12 E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Information Systems*, 38:561–584, 2013.

13 G. De Giacomo, R. De Masellis, and R. Rosati. Verification of conjunctive artifact-centric services. *Int. J. Cooperative Inf. Syst.*, 21(2):111–140, 2012.

14 H. de Man. Case management: Cordys approach. BP Trends (`www.bptrends.com`), 2009.

15 S. Demri and R. Lazić. LTL with the Freeze Quantifier and Register Automata. In *LICS*, 2006.

16 S. Demri, R. Lazić, and A. Sangnier. Model checking freeze LTL over one-counter automata. In *FoSSaCS*, 2008.

17 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, 2009.

18 A. Deutsch, R. Hull, and V. Vianu. Automatic verification of data-driven systems. *Sigmod Record*, 2014.

19 A. Deutsch, Y. Li, and V. Vianu. Hierarchical artifact systems. In preparation.

20 A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.

21 E. Allen Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.

22 B. Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In *PODS*, 2013.

23 R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. Heath III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *ACM DEBS*, 2011.

24 Dimitri Isaak and Christof Löding. Efficient inclusion testing for simple classes of unambiguous -automata. *Inf. Process. Lett.*, 112(14-15), 2012.

25 S. Kumaran, P. Nandi, T. Heath, K. Bhaskaran, and R. Das. ADoc-oriented programming. In *Symp. on Applications and the Internet (SAINT)*, 2003.

26 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

27 A. Lomuscio and J. Michaliszyn. Model checking unbounded artifact-centric systems. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*, 2014.

28 M. Marin, R. Hull, and R. Vaculín. Data centric BPM and the emerging case management standard: A short survey. In *BPM Workshops*, 2012.

29 S. Merz. Model checking: a tutorial overview. In *Modeling and verification of parallel processes*. Springer-Verlag New York, 2001.

30 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, 1967.

31 A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3), 2003.

32 Amir Pnueli. The temporal logic of programs. In *FOCS*, 1977.

33 E. L. Post. Recursive unsolvability of a problem of Thue. *J. of Symbolic Logic*, 12:1–11, 1947.

34 L. Segoufin and S. Torunczyk. Automata based verification over linearly ordered data domains. In *STACS*, 2011.

35 M. Spielmann. Verification of relational transducers for electronic commerce. *JCSS.*, 66(1):40–65, 2003.

36 Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*. Elsevier, 1990.

37 W. van der Aalst and M. Song. Mining social networks: Uncovering interaction patterns in business processes. In *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer Berlin Heidelberg, 2004.

38 W. van der Aalst and A. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4), 2005.

39 W.-D. Zhu et al. Advanced Case Management with IBM Case Manager. Available at `http://www.redbooks.ibm.com/abstracts/sg247929.html?Open`.

On The I/O Complexity of Dynamic Distinct Counting*

Xiaocheng Hu[1], Yufei Tao[1], Yi Yang[2], Shengyu Zhang[1], and Shuigeng Zhou[2]

1 **Chinese University of Hong Kong**
Hong Kong, China
{xchu, taoyf, syzhang}@cse.cuhk.edu.hk

2 **Fudan University**
Shanghai, China
{yyang1, sgzhou}@fudan.edu.cn

Abstract

In *dynamic distinct counting*, we want to maintain a multi-set $\mathcal{S}$ of integers under insertions to answer efficiently the query: how many distinct elements are there in $\mathcal{S}$? In external memory, the problem admits two standard solutions. The first one maintains $\mathcal{S}$ in a hash structure, so that the distinct count can be incrementally updated after each insertion using $O(1)$ expected I/Os. A query is answered for free. The second one stores $\mathcal{S}$ in a linked list, and thus supports an insertion in $O(1/B)$ amortized I/Os. A query can be answered in $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os by sorting, where $N = |\mathcal{S}|$, B is the block size, and M is the memory size.

In this paper, we show that the above two naive solutions are already optimal within a polylog factor. Specifically, for any Las Vegas structure using $N^{O(1)}$ blocks, if its expected amortized insertion cost is $o(\frac{1}{\log B})$, then it must incur $\Omega(\frac{N}{B\log B})$ expected I/Os answering a query in the worst case, under the (realistic) condition that N is a polynomial of B. This means that the problem is repugnant to update buffering: the query cost jumps from 0 dramatically to almost linearity as soon as the insertion cost drops slightly below $\Omega(1)$.

1998 ACM Subject Classification F.2.2. [Analysis of algorithms and problem complexity]: Nonnumerical algorithms and problems

Keywords and phrases Distinct counting, lower bound, external memory

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.265

1 Introduction

This paper studies the *dynamic distinct counting problem* defined as follows. Let $[2^w]$ represent the set of integers $\{0, 1, ..., 2^w - 1\}$, where w is the number of bits in a machine word. We want to support two operations on an initially empty multi-set $\mathcal{S}$:

- Insert(e): add an integer $e \in [2^w]$ to $\mathcal{S}$.
- Query: report the number of distinct elements in $\mathcal{S}$.[1]

* Xiaocheng Hu and Yufei Tao were supported in part by Projects GRF 4168/13 and GRF 142072/14 from HKRGC. Yi Yang and Shuigeng Zhou were supported in part by the Research Innovation Program of Shanghai Municipal Education Commission under grant No. 13ZZ003. Shengyu Zhang was supported in part by Project GRF 4194/13 from HKRGC.

[1] This problem should not be confused with *ϵ-approximate distinct counting* [10], where a query is allowed to return only an approximate answer.

This is a classic problem in computer science. Indeed, distinct queries are useful in such a large variety of contexts that database systems have made them a first-class citizen with direct SQL support: *select distinct count(...)*.

We consider the problem in the standard *external memory* (EM) model of computation (a.k.a. the *I/O model*). In this model, a machine has M words of memory, and a disk of an unbounded size. The disk has been formatted into disjoint *blocks* of size B words. It holds that $M \geq 2B$, i.e., the memory can accommodate at least two blocks. An I/O either reads a block from the disk into memory, or conversely writes B words from memory to a disk block. The *cost* of an algorithm is measured as the number of I/Os performed. The *space* of a structure is measured as the number of blocks occupied. CPU computation is free, but can take place only on memory data. We use N to denote the problem size (e.g., for the dynamic distinct counting problem, N equals the number of insertions). A structure is said to consume *polynomial space* if its space consumption is bounded by $N^{O(1)}$ in the worst case.

Dynamic distinct counting admits two standard solutions:

- The first one is to maintain $\mathcal{S}$ in a hash structure (e.g., [6]) of linear space $O(N/B)$. Given an INSERT(e), by probing the bucket of e, one can incrementally maintain the distinct count in $O(1)$ expected I/Os. A query can be answered by simply returning this count for free.
- The second solution organizes $\mathcal{S}$ in a linked list with the last block pinned in memory. The block is flushed to the disk after it has accumulated $\Omega(B)$ elements. This achieves the lowest amortized cost of $O(1/B)$ I/Os per insertion. A query can be answered by sorting $\mathcal{S}$ from scratch using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os [1].

Rather naive as these solutions may appear, they still represent the best update-query tradeoffs to this date.

1.1 Our Results

In this paper, we show that both of the aforementioned naive solutions – exactly how a DBMS supports dynamic distinct counting – are already optimal up to a small factor. Specifically, no Las Vegas structure of polynomial space can do much better than $\Omega(N/B)$ in query cost if it must support fast updates:

▶ **Theorem 1.** *Let t_u be the expected amortized insertion cost of a polynomial-space Las Vegas structure for dynamic distinct counting, and t_q be its expected query cost, where both expectations are taken over the random choices made by the structure. In the scenario where $N = B^c, M = B^{c'}$ (for any integer constants $c' \geq 1$ and $c \geq c' + 1$), and $3 \log N \leq w = O(\log N)$, if $t_u = o(\frac{1}{\log B})$, then $t_q = \Omega(\frac{N}{B \log B})$.*

The theorem holds even for structures that defy the *indivisibility assumption*[2]. Furthermore, by fitting in some typical values for N, M, and B, one would quickly realize that N and M are almost always polynomials of B in practice.

Theorem 2 indicates that dynamic element counting is "repugnant" to update buffering. When there is no buffering (e.g., hashing), one can achieve $t_u = O(1)$ and $t_q = 0$ (free queries). On the other hand, if t_u needs to be improved by just an $\omega(\log B)$ factor, t_q surges dramatically to almost $\Omega(N/B)$, that is, any query algorithm must spend nearly the same cost as reading the entire dataset $\mathcal{S}$.

[2] This assumption says that every data element must be stored as an atom occupying a word. Thus, one cannot, for example, compress the bits of an element to save space.

Technical Overview. We will consider instead the *dynamic element distinctness problem*, where we want to support two operations on an initial empty multi-set $\mathcal{S}$:

- INSERT(e): add an integer $e \in [2^w]$ to $\mathcal{S}$.
- QUERY: report *whether* all the elements in $\mathcal{S}$ are distinct.

A structure solving dynamic distinct counting also solves dynamic element distinctness with exactly the same space, query, and update cost: first obtain the number x of distinct elements in $\mathcal{S}$, and declares that all elements in $\mathcal{S}$ are distinct if and only if x equals the number of insertions in history. Therefore, a lower bound on the latter problem also carries over to the former. Indeed, the main result of this paper is:

▶ **Theorem 2.** *Let t_u be the expected amortized insertion cost of a polynomial-space Las Vegas structure for the dynamic element distinctness problem, and t_q be its expected query cost, where both expectations are taken over the random choices made by the structure. In the scenario where $N = B^c, M = B^{c'}$ (for any integer constants $c' \geq 1$ and $c \geq c' + 1$), and $3 \log N \leq w = O(\log N)$, if $t_u = o(\frac{1}{\log B})$, then $t_q = \Omega(\frac{N}{B \log B})$.*

We establish Theorem 2 by working under the cell-probe model. An immediate obstacle is that, every insertion obviously must probe at least one cell (recall that the cell-probe model is *stateless*, i.e., no information is passed between two operations), which is at odds with our goal of having an $o(1)$ bound on t_u. A main idea behind our techniques is to prove a tradeoff between the query cost and the cost of a *group* of $N/B = \Omega(M)$ insertions. Then, by requiring each group to probe $o(N/B)$ cells, we get essentially an $o(1)$ amortized bound on t_u, thus overcoming the obstacle. The tradeoff (between the query and group update costs) is obtained by a novel reduction from set disjointness.

1.2 Previous Work: Lower Bounds in EM with $o(1)$ Update Cost

In the EM model, an important line of research is to understand the limitation of buffering, or more specifically: what is the best query time achievable if the amortized update cost needs to be $o(1)$? Our work belongs to this category of work. In this subsection, we review the existing results under the category to the best of our knowledge.

The *offline* version of the dynamic element distinctness problem, where the goal is to determine if a static set $\mathcal{S}$ has duplicate elements, is known to require at least $c\frac{N}{B}\log_{M/B}\frac{N}{B}$ I/Os, for some constant c, in EM under the indivisibility assumption [2, 3]. This implies the following *dynamic* lower bound: if $t_u \leq \frac{c}{2B}\log_{M/B}\frac{N}{B}$, then $t_q = \Omega(\frac{N}{B}\log_{M/B}\frac{N}{B})$. In turn, this tradeoff implies that no structure (obeying the indivisibility assumption) with $o(1)$ update cost can answer a query faster than sorting when $\log_{M/B}(N/B) = \Omega(B)$, that is, N is exponential in B. In the more realistic settings where N and M are polynomials of B, however, the tradeoff loses its significance because it requires the impossible that $t_u = o(1/B)$. The above discussion also applies to dynamic distinct counting.

There is considerable work [4, 9, 12, 13, 15] in understanding the I/O complexity of the *dynamic membership problem*, where the goal is to maintain a set S of elements under insertions, such that queries of the following form "does element e belong to S?" can be answered efficiently. The lower bounds by Brodal and Fagerberg [4] and Wei et al. [13] were proved under the indivisibility assumption, while the others hold without the assumption. More specifically, the techniques of Yi and Zhang [15] are geared to establish a tradeoff of the following form (abusing notations slightly, let t_u, t_q be the expected amortized insertion cost and expected query cost respectively also for dynamic membership): if $t_q \leq 1 + \delta$ where δ is a sufficiently small constant, then $t_u = \Omega(1)$. Verbin and Zhang [12] showed a different

tradeoff: if $t_u \leq 1 - \epsilon$ where $0 < \epsilon < 1$ is any constant, then $t_q = \Omega(\log_B N)$. Iacono and Patrascu [9] presented an alternative tradeoff between t_u and the worst-case query cost t_q^{worst}: if $t_u \leq 1 - \epsilon$, then $t_q^{worst} = \Omega(\frac{\log N}{\log(B \cdot t_u)})$.

The results in both [9] and [15] were derived from the *chronogram technique* [7, 11]. The hard input consists of an insertion sequence followed by a single query. The sequence is then divided into subsequences, called *epochs*, whose lengths increase geometrically when they are ordered reverse chronologically. The crux of a lower-bound argument is to show that when t_u is small, for every epoch, the query must read at least a block "exclusively belonging to" that epoch with constant probability. Thus, the expected query cost is at least the number of epochs. Unfortunately, the chronogram technique does not appear to be the right tool for dynamic element distinctness. This is because the number of epochs is logarithmic to the number N of insertions, thus making it difficult to prove a super-logarithmic lower bound on the query cost. Our goal, as shown in Theorem 2, is to establish an almost linear lower bound.

Finally, Yi [14] studied *dynamic 1d range reporting*, where the goal is to maintain a set S of elements from an ordered domain under insertions so that the following queries can be answered efficiently: given a range $[e_1, e_2]$ where e_1, e_2 are elements from the domain, report $S \cap [e_1, e_2]$. He presented lower bound tradeoffs between the amortized insertion cost and the query cost of deterministic structures, based on a dynamic version of the *indexability model* [8], and thus, still inheriting the indivisibility assumption.

2 An I/O Lower Bound of Dynamic Element Distinctness

2.1 Cell-Probe Model

The core of Theorem 2 is a lower bound in a cell-probe model of computation defined as follows. The machine is equipped with a CPU and an array of memory *cells* of size wB bits. The CPU has a register of an unbounded size. A *cell probe* either reads or writes a cell. The *cost* of an algorithm is measured as the number of cells probed (CPU calculation is free). The *space* of a structure is measured as the number of cells occupied.

For a deterministic structure, an operation can be modeled as follows. At the beginning, the register contains nothing but the operation's input parameter. At each step, the operation probes a cell c, such that the address of c and whether the probe is a read or a write are both functions of the register. If the probe is a read, the register is updated as a function of the register's current form and the contents of c. If it is a write, then the value written to c is a function of the register, after which the register is updated as a function of its current form. We model a randomized structure by assuming that it has free access to a random bit sequence, and that it behaves as a deterministic structure after the sequence has been fixed.

2.2 Hard Input

Set $N = B^c$ and $M = B^{c'}$ where c and c' can be any integer constants such that $c' \geq 1$ and $c \geq c' + 1$. Fix w to be any integer such that $3 \log N \leq w = O(\log N)$. We use the term (N/B)-*subset* to refer to a set of N/B (distinct) integers from $[2^w]$. We consider a variant of the element distinctness problem where a structure maintains an initially empty multi-set $\mathcal{S}$ under two operations:

- G-Insert(G): add an (N/B)-subset G to $\mathcal{S}$.
- Query: decide whether the elements of $\mathcal{S}$ are distinct.

We refer to the above as the *element distinctness with group insertions* (EDGI) *problem*.

Our hard input consists of a sequence Σ of B *batches*, where each batch has two operations: a G-INSERT(G) followed by a single QUERY, where G is taken uniformly at random from all the $\binom{2^w}{N/B}$ possible (N/B)-subsets. At the end of Σ, $\mathcal{S}$ has exactly N elements, but they are not necessarily distinct. We denote by $\boldsymbol{\Sigma}$ the set of all possible sequences as generated above. Note that Σ follows the uniform distribution over $\boldsymbol{\Sigma}$. We say that a structure uses *polynomial space* if it occupies at most $N^{O(1)}$ cells when it processes any Σ of $\boldsymbol{\Sigma}$. Our discussion will focus only on such structures.

2.3 Set Disjointness

This is a communication complexity problem – denoted as Disj henceforth – defined as follows. Alice is given a subset X of $[2^w]$, and Bob is given a subset Y of $[2^w]$. Their goal is to determine whether $X \cap Y = \emptyset$ by sending each other messages, each being a sequence of bits, according to a pre-agreed protocol Π. For our purpose, it suffices to consider that Π is *deterministic* defined as follows. The person sending the first message is always fixed (regardless of X and Y). Then, Alice and Bob take turns to send messages, such that every message is a function of the previous messages and the sender's input. We denote by $\Pi_A(X, Y)$ the bit sequence concatenating chronologically all the messages sent by Alice on input (X, Y), and by $|\Pi_A(X, Y)|$ the number of bits in $\Pi_A(X, Y)$. Let $\Pi_B(X, Y)$ and $|\Pi_B(X, Y)|$ be defined similarly for Bob.

We denote by $\boldsymbol{\mathcal{I}}$ the set of all possible inputs (X, Y) to Disj. Let μ be a probability density function (pdf) over $\boldsymbol{\mathcal{I}}$, namely, $\mu(X, Y)$ gives the probability that Alice's and Bob's subsets are X and Y, respectively. Define:

$$\alpha_\mu(\Pi) = \sum_{(X,Y)\in\boldsymbol{\mathcal{I}}} |\Pi_A(X, Y)| \cdot \mu(X, Y)$$

$$\beta_\mu(\Pi) = \sum_{(X,Y)\in\boldsymbol{\mathcal{I}}} |\Pi_B(X, Y)| \cdot \mu(X, Y).$$

We call $\alpha_\mu(\Pi)$ the *μ-average Alice cost* of Π, and $\beta_\mu(\Pi)$ the *μ-average Bob cost* of Π.

We will be particularly interested in the scenario where X and Y have specific sizes n and m, respectively, where n and m are integers in $[1, 2^w]$. Let $\boldsymbol{\mathcal{I}}^{n,m}$ represent the set of all possible inputs (X, Y) to Disj such that $|X| = n$ and $|Y| = m$. When the input to Disj is drawn *only* from $\boldsymbol{\mathcal{I}}^{n,m}$, we will refer to Disj as (n, m)-Disj.

We use $\mathsf{unif}^{n,m}$ to denote the uniform distribution μ over $\boldsymbol{\mathcal{I}}^{n,m}$, namely, $\mu(X, Y) = 1/|\boldsymbol{\mathcal{I}}^{n,m}|$ if $(X, Y) \in \boldsymbol{\mathcal{I}}^{n,m}$, while $\mu(X, Y) = 0$ for any $(X, Y) \in \boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{I}}^{n,m}$. The appendix contains a proof for the following tradeoff between $\alpha_{\mathsf{unif}^{n,m}}(\Pi)$ and $\beta_{\mathsf{unif}^{n,m}}(\Pi)$:

▶ **Theorem 3.** *When $3 \cdot \max\{\log n, \log m\} \leq w = o(\min\{n, m\})$, for any deterministic protocol Π solving (n, m)-Disj, it must hold that either $\alpha_{\mathsf{unif}^{n,m}}(\Pi) = \Omega(n)$ or $\beta_{\mathsf{unif}^{n,m}}(\Pi) = \Omega(m)$.*

2.4 Cell-Probe Lower Bound for EDGI

Let us fix a *deterministic* polynomial-space cell-probe structure Υ on the EDGI problem. For each sequence $\Sigma \in \boldsymbol{\Sigma}$, denote by $T(\Sigma)$ the cost of Υ (in the number of cells probed) in processing Σ. Note that $T(\Sigma)$ is a random variable because Σ is uniformly distributed in $\boldsymbol{\Sigma}$. Next, we analyze the expectation of $T(\Sigma)$.

Recall that Σ consists of B batches, each of which has a G-INSERT operation followed by a query. Let G_i $(1 \leq i \leq B)$ be the (N/B)-subset added to $\mathcal{S}$ by the G-INSERT operation in

the i-th batch of Σ. We say that Σ is *i-distinct* if $G_j \cap G_{j'} = \emptyset$ for each pair of j, j' satisfying $1 \leq j < j' \leq i$. Denote by $T_i(\Sigma)$ the number of cells probed by Υ in handling the i-th batch. Thus, $T(\Sigma) = \sum_i T_i(\Sigma)$. By the linearity of expectation, we know: $\mathbf{E}[T(\Sigma)] = \sum_i \mathbf{E}[T_i(\Sigma)]$.

We prove in Section 3 the following relationship between set disjointness and EDGI:

▶ **Lemma 4.** *For each $i \in [1, B-1]$, there exists a deterministic protocol Π solving the $(N/B, iN/B)$-*Disj *problem with $\alpha_{\mathsf{unif}^{N/B,iN/B}}(\Pi) \leq \lambda \cdot O(\log N)$ and $\beta_{\mathsf{unif}^{N/B,iN/B}}(\Pi) \leq \lambda \cdot wB$, where $\lambda = \mathbf{E}[T_{i+1}(\Sigma) \mid \Sigma$ is i-distinct$]$.*

Combining Lemma 4 and Theorem 3, we have the following when $w \geq 3 \log N$:

$$\mathbf{E}[T_{i+1}(\Sigma) \mid \Sigma \text{ is } i\text{-distinct}] \quad = \quad \Omega\left(\min\left\{\frac{N}{B \log N}, \frac{iN}{wB^2}\right\}\right). \tag{1}$$

▶ **Observation 5.** *When $w \geq 3 \log N$, it must hold that $\mathbf{Pr}[\Sigma$ is B-distinct$] \geq 1/2$.*

Proof.

$$\begin{aligned}
\mathbf{Pr}[\Sigma \text{ is } B\text{-distinct}] &\geq 1 - \sum_{1 \leq i < j \leq B} \sum_{x \in [2^w]} \mathbf{Pr}[x \in G_i \ \wedge \ x \in G_j] \\
&= 1 - \binom{B}{2} \cdot 2^w \cdot \left(\frac{N/B}{2^w}\right)^2 \\
&\geq 1 - \frac{1}{2} B^2 \cdot 2^w \cdot \frac{N^2}{B^2 \cdot 2^{2w}} \\
&\geq 1/2. \qquad (\text{by } w \geq 3 \log N)
\end{aligned}$$

◀

When $3 \log N \leq w = \Theta(\log N)$, we have from the above:

$$\begin{aligned}
\mathbf{E}[T(\Sigma)] &\geq \sum_{i=1+B/2}^{B} \mathbf{E}[T_i(\Sigma)] \\
&\geq \sum_{i=B/2}^{B-1} \mathbf{E}[T_{i+1}(\Sigma) \mid \Sigma \text{ is } i\text{-distinct}] \cdot \mathbf{Pr}[\Sigma \text{ is } i\text{-distinct}] \\
&\geq (1/2) \sum_{i=B/2}^{B-1} \mathbf{E}[T_{i+1}(\Sigma) \mid \Sigma \text{ is } i\text{-distinct}] \\
&\quad (\text{as } \mathbf{Pr}[\Sigma \text{ is } i\text{-distinct}] \geq \mathbf{Pr}[\Sigma \text{ is } B\text{-distinct}]) \\
&\geq (1/2) \sum_{i=B/2}^{B-1} \Omega\left(\frac{N}{B \log N}\right) \qquad (\text{by (1)}) \\
&= \Omega\left(\frac{N}{\log N}\right)
\end{aligned} \tag{2}$$

where the last inequality used the fact that $\frac{N}{B \log N} = \Theta(\frac{iN}{wB^2})$ for $i \in [B/2, B]$ and $w = \Theta(\log N)$. Now it is easy to obtain the following lower bound:

▶ **Lemma 6.** *Given $N = B^c$ for some integer constant $c \geq 2$, and w such that $3 \log N \leq w = O(\log N)$, for any Las Vegas polynomial-space structure, there is at least a sequence $\Sigma_0 \in \mathbf{\Sigma}$ such that the structure requires $\Omega(N/\log B)$ expected cost processing Σ_0.*

Proof. We have proved that when Σ is uniformly distributed in $\mathbf{\Sigma}$, *every* deterministic structure must incur $\Omega(N/\log N) = \Omega(N/\log B)$ expected cost in processing Σ (applying (2) and $\log N = \Theta(\log B)$). The lemma then follows from Yao's minimax principle. ◀

2.5 From Cell-Probe to EM

Still set $N = B^c$ and $M = B^{c'}$ (for integer constants $c' \geq 1$ and $c \geq c' + 1$), and $3 \log N \leq w = O(\log N)$. In this scenario, next we show that if an EM structure Υ' solves the dynamic element distinctness problem with expected amortized insertion cost t_u and expected query cost t_q, then there is a cell-probe structure Υ for the EDGI problem, such that for every sequence $\Sigma \in \mathbf{\Sigma}$, Υ processes Σ in $t_u N + t_q B + 4M$ expected cost.

The main idea is to simulate Υ' in the cell-probe model by setting aside M/B fixed cells to preserve the memory of Υ' across two operations – refer to those cells as the *state cells*. In addition, at any moment, every disk block occupied by Υ' corresponds to a unique cell with the same contents. At the beginning, Υ (in the cell-probe model) and Υ' (in the EM model) are empty, and so are the state cells. Υ behaves according to how Υ' works. Consider, in general, the processing of the i-th ($1 \leq i \leq B$) batch (G-Insert(G), Query) of Σ. Υ first reads the M/B state cells so that the (cell-probe) CPU register contains all the information in the memory of Υ' (in EM). Then, we invoke the insertion algorithm of Υ' to insert the N/B elements of G to Υ' (the ordering does not matter). In this process, (i) whenever Υ' reads (writes) a block, Υ reads (writes) the block's corresponding cell; (ii) whenever Υ' performs CPU computation, Υ performs the same computation in the register. After Υ' has finished inserting the elements of G, Υ writes the memory of Υ' to the state cells with cost M/B. At this point, Υ has completed the operation G-Insert(G). In the same manner, Υ handles a Query by first reading the state cells, simulating Υ', and then writing the state cells. By definitions of t_u and t_q, Υ' performs at most $t_u N + t_q B$ expected I/Os in doing N insertions and B queries. Hence, Υ probes at most $t_u N + t_q B + 4M$ cells in expectation, noticing that $4M/B$ probes are needed for reading and writing the state cells in each batch of Σ.

We thus know from Lemma 6 that $t_u N + t_q B + 4M = \Omega(N/\log B)$, which gives $t_u + \frac{t_q B}{N} + 4/B = \Omega(1/\log B)$ as $N \geq MB$. Thus, if $t_u = o(1/\log B)$, then t_q must be $\Omega(\frac{N}{B \log B})$, as claimed in Theorem 2.

▶ Remark. Echoing the high level description in Section 1.1, it is worth mentioning that Lemma 6 implies an update-query tradeoff for EDGI. Let τ_u be the expected amortized G-Insert cost of an EDGI structure, and τ_q be its expected query cost. If the structure uses polynomial space, the lemma shows that $\tau_u B + \tau_q B = \Omega(N/\log B)$.

3 Reduction from Set Disjointness to EDGI

In this section, which serves as a proof of Lemma 4, let us set $n = N/B$ and $m = iN/B$. As before, denote by Υ the (deterministic) EDGI structure in the context of the lemma. Consider the $\mathcal{S}$ after having processed the first i batches of Σ. Note that $\mathcal{S}$ is a set (as opposed to a multi-set) because Σ is i-distinct. Let G be the (N/B)-subset to be inserted in the $(i+1)$-th batch. Clearly, $\mathcal{S} \cap G = \emptyset$ if and only if the query of the $(i+1)$-th batch returns "distinct". Based on this observation, next we design a protocol for the $(N/B, iN/B)$-Disj problem that works by asking Alice and Bob to simulate the execution of Υ.

Let us first introduce some notions useful in the subsequent discussion. Let $G_1, ..., G_i$ be (N/B)-subsets of $[2^w]$ that are mutually disjoint. These i subsets make an *i-distinct sequence* $\mathcal{G} = (G_1, G_2, ..., G_i)$. In general, given $\mathcal{G}$, we define $S(\mathcal{G}) = \cup_{j=1}^{i} G_j$. Let $\boldsymbol{\mathcal{G}}$ be the set of all different i-distinct sequences. Define $g = |\boldsymbol{\mathcal{G}}|$, which equals $\frac{(2^w)!}{(2^w - m)!((N/B)!)^i}$. *Conditioned on* the fact that Σ is i-distinct, every i-distinct sequence has $1/g$ probability of being the sequence of (N/B)-subsets that are added to $\mathcal{S}$ by the first i batches of Σ.

Furthermore, since Υ is deterministic, every $\mathcal{G}$ defines an instance of Υ – denoted as $\Upsilon(\mathcal{G})$ – which is exactly the cell contents after Υ has processed i batches: (G-INSERT(G_1), QUERY), ..., (G-INSERT(G_i), QUERY).

Fix a $\mathcal{G}$ and therefore $\Upsilon(\mathcal{G})$. Given an (N/B)-subset X, let $C(\Upsilon(\mathcal{G}), X)$ be the cost of processing batch (G-INSERT(X), QUERY) on $\Upsilon(\mathcal{G})$. Define:

$$C(\mathcal{G}) = \frac{1}{\binom{2^w}{n}} \sum_X C(\Upsilon(\mathcal{G}), X). \tag{3}$$

In other words, $C(\mathcal{G})$ is the average $C(\Upsilon(\mathcal{G}), X)$ over all the possible X.

Now, fix Y as an (iN/B)-subset of $[2^w]$. Consider the set $Z(Y)$ of i-distinct sequences $\mathcal{G}$ such that $S(\mathcal{G}) = Y$, that is, each $\mathcal{G} \in Z(Y)$ is a possible way to break Y into a sequence of (N/B)-subsets. Let $z = \min_{\mathcal{G} \in Z(Y)} C(\mathcal{G})$. Define $\mathcal{G}(Y)$ to be the $\mathcal{G} \in Z(Y)$ with the smallest $C(\mathcal{G})$; if multiple i-distinct sequences $\mathcal{G}$ of $Z(Y)$ satisfy $C(\mathcal{G}) = z$, define $\mathcal{G}(Y)$ to be the first one among them in lexicographical order. Note that $\mathcal{G}(Y)$ is indeed a function of Y by the determinism of Υ.

We are now ready to describe a protocol for (n, m)-Disj. Let X and Y be the inputs of Alice and Bob, respectively. Bob first identifies $\mathcal{G}(Y)$ and builds a structure $\Upsilon = \Upsilon(\mathcal{G}(Y))$ locally with no communication. Then, Alice simulates what the cell-probe CPU does to perform a batch (G-INSERT(X), QUERY) on Υ. Specifically, she maintains locally a set R of cells to simulate the CPU register. R is initially empty. Whenever G-INSERT(X) writes a cell c, she does not communicate any bit, but simply adds c to R, i.e., R remembers the address and contents of c. Whenever G-INSERT(X) reads a cell c, Alice first checks whether $c \in R$. If so, she takes the contents of c directly from R. Otherwise, she requests c from Bob by sending $O(\log N)$ bits to address c – note that $O(\log N)$ suffices because Υ uses polynomial space – and then, Bob sends back the contents of c in wB bits. *Without* clearing R, Alice proceeds to simulate QUERY in the same manner. At the end, if QUERY reports "disjoint", Alice notifies Bob in one bit that $X \cap Y = \emptyset$; otherwise, her notification bit indicates $X \cap Y \neq \emptyset$.

We argue that the protocol, denoted as Π, has low $\alpha_{\mathsf{unif}^{n,m}}(\Pi)$ and $\beta_{\mathsf{unif}^{n,m}}(\Pi)$. Note that the protocol executes in *rounds*, in each of which Alice and Bob communicate $O(\log N)$ and wB bits, respectively. Let $r(X, Y)$ be the number of rounds in the protocol on input (X, Y). Define $\bar{r} = \frac{1}{\binom{2^w}{n}\binom{2^w}{m}} \sum_{X,Y} r(X, Y)$, namely, $\bar{r}$ is the average number of rounds over all the inputs to (n, m)-Disj. Hence, $\alpha_{\mathsf{unif}^{n,m}}(\Pi) \leq \bar{r} \cdot O(\log N)$ and $\beta_{\mathsf{unif}^{n,m}}(\Pi) \leq \bar{r} \cdot wB$.

Since our protocol ensures $r(X, Y) \leq C(\Upsilon(\mathcal{G}(Y)), X)$, the following relationship becomes obvious:

$$\bar{r} \leq \frac{1}{\binom{2^w}{n}\binom{2^w}{m}} \sum_{X,Y} C(\Upsilon(\mathcal{G}(Y)), X). \tag{4}$$

Next, we complete the proof of Lemma 4 by showing that the right hand side of (4) is at most $\mathbf{E}[T_{i+1}(\Sigma) \mid \Sigma \text{ is } i\text{-distinct}]$ through the following derivation:

$$\begin{aligned}
\frac{1}{\binom{2^w}{n}\binom{2^w}{m}}\sum_{X,Y} C(\Upsilon(\mathcal{G}(Y)),X) &\leq \frac{1}{\binom{2^w}{n}\binom{2^w}{m}}\sum_{X,Y}\left(\frac{((N/B)!)^i}{m!}\sum_{\mathcal{G}\in Z(Y)} C(\Upsilon(\mathcal{G}),X)\right)\\
&\quad \text{(Note: } |Z(Y)| = m!/((N/B)!)^i\text{, and}\\
&\quad C(\Upsilon(\mathcal{G}(Y)),X) \leq C(\Upsilon(\mathcal{G}),X) \text{ for every } \mathcal{G}\in Z(Y))\\
&= \frac{1}{\binom{2^w}{n}\binom{2^w}{m}}\frac{((N/B)!)^i}{m!}\sum_{X,\mathcal{G}\in\boldsymbol{\mathcal{G}}} C(\Upsilon(\mathcal{G}),X)\\
&\quad \text{(Note: Every } \mathcal{G}\in\boldsymbol{\mathcal{G}} \text{ belongs to the } Z(Y) \text{ of a unique } Y)\\
&= \frac{1}{\binom{2^w}{n}\cdot g}\sum_{X,\mathcal{G}\in\boldsymbol{\mathcal{G}}} C(\Upsilon(\mathcal{G}),X)\\
&= \mathbf{E}[T_{i+1}(\Sigma) \mid \Sigma \text{ is } i\text{-distinct}].
\end{aligned}$$

References

1. Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.
2. Lars Arge, Mikael Knudsen, and Kirsten Larsen. A general lower bound on the I/O-complexity of comparison-based algorithms. In *Algorithms and Data Structures Workshop (WADS)*, pages 83–94, 1993.
3. Lars Arge and Peter Bro Miltersen. On showing lower bounds for external-memory computational geometry problems. *DIMACS Series in Discrete Mathematics*, pages 139–159, 1999.
4. Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 546–554, 2003.
5. Anirban Dasgupta, Ravi Kumar, and D. Sivakumar. Sparse and lopsided set disjointness via information theory. In *APPROX-RANDOM*, pages 517–528, 2012.
6. Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Patrascu. De dictionariis dynamicis pauco spatio utentibus (*lat.* on dynamic dictionaries using little space). In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 349–361, 2006.
7. Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989.
8. Joseph M. Hellerstein, Elias Koutsoupias, Daniel P. Miranker, Christos H. Papadimitriou, and Vasilis Samoladas. On a model of indexability and its bounds for range queries. *Journal of the ACM (JACM)*, 49(1):35–55, 2002.
9. John Iacono and Mihai Patrascu. Using hashing to solve the dictionary problem. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 570–582, 2012.
10. Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010.
11. Mihai Patrascu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 546–553, 2004.

12 Elad Verbin and Qin Zhang. The limits of buffering: a tight lower bound for dynamic membership in the external memory model. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 447–456, 2010.

13 Zhewei Wei, Ke Yi, and Qin Zhang. Dynamic external hashing: the limit of buffering. In *Proceedings of Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 253–259, 2009.

14 Ke Yi. Dynamic indexability and the optimality of B-trees. *Journal of the ACM (JACM)*, 59(4), 2012.

15 Ke Yi and Qin Zhang. On the cell probe complexity of dynamic membership. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 123–133, 2010.

A Proof of Theorem 3

Set $W = 2^w$. Recall that the set disjointness problem Disj was defined by having (X, Y) drawn from the domain of $[W]$. We will from now on denote the problem as Disj_W, namely, by indicating the domain size explicitly. Also, it will be convenient to regard an input (X, Y) to Disj_W as a pair of W-dimensional vectors $\boldsymbol{x} = (x_0, x_1, ..., x_{W-1})$ and $\boldsymbol{y} = (y_0, y_1, ..., y_{W-1})$ such that for each $i \in [W]$:

$$x_i = 1 \text{ if } i \in X, \text{ or } = 0 \text{ otherwise;}$$
$$y_i = 1 \text{ if } i \in Y, \text{ or } = 0 \text{ otherwise.}$$

We will use $|\boldsymbol{x}|$ to denote the number of 1's in $\boldsymbol{x}$ (similarly, $|\boldsymbol{y}|$).

We will build our proof on a result of [5] of Dasgupta, Kumar, and Sivakumar. They considered Disj_W under a distribution of $(\boldsymbol{x}, \boldsymbol{y})$ parameterized by real values $p, q \in [0, 1]$ – denoted as $\mathsf{rand}_W^{p,q}$ – where, independently for each $i \in [W]$: (i) x_i equals 1 with probability p, and (ii) independently, $y_i = 1$ with probability q. They proved[3]:

▶ **Lemma 7 ([5]).** *For any deterministic protocol* Π *solving* Disj_W*, it holds that either* $\alpha_{\mathsf{rand}_W^{p,q}}(\Pi) = \Omega(pW)$ *or* $\beta_{\mathsf{rand}_W^{p,q}}(\Pi) = \Omega(qW)$.

Theorem 3, on the other hand, is concerned with distribution $\mathsf{unif}_W^{n,m}$, that is, the uniform distribution over the set of all inputs $(\boldsymbol{x}, \boldsymbol{y}) \in \{0,1\}^W \times \{0,1\}^W$ satisfying $|\boldsymbol{x}| = n$ and $|\boldsymbol{y}| = m$. We will defer the proof of the next lemma till later:

▶ **Lemma 8.** *Suppose that* $\max\{n, m\} \leq \sqrt{W}/2$*, and that we are given a deterministic protocol* Π *for* (n, m)-Disj_W*. Then, for any* $n' \in [n/2, n]$ *and* $m' \in [m/2, m]$*, there is a deterministic protocol* $\Pi^{n',m'}$ *for* (n', m')-$\mathsf{Disj}_{W/2}$ *such that*

$$\alpha_{\mathsf{unif}_{W/2}^{n',m'}}(\Pi^{n',m'}) < 3 \cdot \alpha_{\mathsf{unif}_W^{n,m}}(\Pi), \text{ and } \beta_{\mathsf{unif}_{W/2}^{n',m'}}(\Pi^{n',m'}) < 3 \cdot \beta_{\mathsf{unif}_W^{n,m}}(\Pi).$$

Now, given a deterministic protocol Π solving (n, m)-Disj_W, we design a protocol Π' for $\mathsf{Disj}_{W/2}$. Given an input $(\boldsymbol{x}', \boldsymbol{y}')$ to $\mathsf{Disj}_{W/2}$, Π' works as follows:

- If $|\boldsymbol{x}'| \notin [n/2, n]$, Alice sends $\boldsymbol{x}$ to Bob, who then sends back the result with one more bit. Conversely, if $|\boldsymbol{y}'| \notin [m/2, m]$, Bob sends $\boldsymbol{y}$ to Alice, who then sends back the result.
- In all other cases, Alice and Bob send $n' = |\boldsymbol{x}'|$ and $m' = |\boldsymbol{y}'|$, respectively, after which they run the protocol $\Pi^{n',m'}$ of Lemma 8 that is obtained from Π for (n', m')-$\mathsf{Disj}_{W/2}$.

[3] By combining Lemmas 5-8 of [5] with parameters $\theta = 0$, $\alpha = p$ and $\beta = q$.

Next we argue that Π is efficient under the distribution $\mathsf{rand}^{p,q}_{W/2}$ with

$$p = (3n/4)/(W/2), \quad \text{and} \quad q = (3m/4)/(W/2).$$

Define $\mathtt{out}$ to be the event where either $|\boldsymbol{x}'| \notin [n/2, n]$ or $|\boldsymbol{y}'| \notin [m/2, m]$ holds. Clearly:

$$\alpha_{\mathsf{rand}^{p,q}_{W/2}}(\Pi') \quad \leq \quad \mathbf{Pr}[\mathtt{out}](O(1) + W/2) + \mathbf{Pr}[\neg\mathtt{out}](2\log W + \alpha'),$$

where

$$\alpha' \quad = \sum_{\substack{n' \in [\frac{n}{2}, n], \\ m' \in [\frac{m}{2}, m]}} \left(\mathbf{Pr}\left[|\boldsymbol{x}'| = n', |\boldsymbol{y}'| = m' \mid \neg\mathtt{out}\right] \cdot \alpha_{\mathsf{unif}^{n',m'}_{W/2}}(\Pi^{n',m'}) \right) \leq 3 \cdot \alpha_{\mathsf{unif}^{n,m}_{w}}(\Pi). \quad (5)$$

(by Lemma 8)

By Chernoff's bounds, $\mathbf{Pr}[\mathtt{out}] \leq e^{-\Omega(n)}$ which, together with $w = o(n)$, gives $\mathbf{Pr}[\mathtt{out}]W/2 = o(n)$. Therefore:

$$\alpha_{\mathsf{rand}^{p,q}_{W/2}}(\Pi') = o(n) + 2\log W + 3 \cdot \alpha_{\mathsf{unif}^{n,m}_{w}}(\Pi) = o(n) + 3 \cdot \alpha_{\mathsf{unif}^{n,m}_{w}}(\Pi).$$

Similar analysis shows that

$$\beta_{\mathsf{rand}^{p,q}_{W/2}}(\Pi') = o(m) + 3\beta_{\mathsf{unif}^{n,m}_{W}}(\Pi).$$

On the other hand, Lemma 7 states that either $\alpha_{\mathsf{rand}^{p,q}_{W/2}}(\Pi') = \Omega(pW/2) = \Omega(n)$ or $\beta_{\mathsf{rand}^{p,q}_{W/2}}(\Pi') = \Omega(qW/2) = \Omega(m)$. Therefore, either $\alpha_{\mathsf{unif}^{n,m}_{W}}(\Pi) = \Omega(n)$ or $\beta_{\mathsf{unif}^{n,m}_{W}}(\Pi) = \Omega(m)$, as claimed.

B Proof of Lemma 8

It suffices to consider that Π never incurs more than W bits of communication. Let us define T to be the set consisting of all triplets (S, U, V) satisfying both conditions below:

- S, U, and V are pairwise disjoint subsets of $[W]$;
- $|S| = W/2$, $|U| = n - n'$, and $|V| = m - m'$.

Given a triplet (S, U, V), we design a protocol Π^{SUV} for $\mathsf{Disj}_{W/2}$ as follows. Upon an input $(\boldsymbol{x}', \boldsymbol{y}') \in \{0,1\}^{W/2} \times \{0,1\}^{W/2}$, Π^{SUV} runs Π on $(\boldsymbol{x}, \boldsymbol{y})$ where

- $\boldsymbol{x}$ is the W-dimensional vector such that
  - the projection of $\boldsymbol{x}$ onto the subspace defined by S gives $\boldsymbol{x}'$;
  - $x_i = 1$ for all $i \in U$; $x_i = 0$ for all $i \in [W] - S - U$.
- $\boldsymbol{y}$ is defined in the same way after replacing $\boldsymbol{x}'$ with $\boldsymbol{y}'$, and U with V.

It is easy to verify that Π^{SUV} is correct on all $(\boldsymbol{x}', \boldsymbol{y}')$.

For each $b \in \{0,1\}$ and any $\boldsymbol{x} \in \{0,1\}^W$, define $\boldsymbol{x}^b = \{i \in [W] \mid x_i = b\}$, namely, the set of dimensions on which $\boldsymbol{x}$ takes the value b. Now, we induce from $\mathsf{unif}^{n,m}_{W}$ a distribution ν on T, by using the following 5-step process to generate a triplet (S, U, V):

1. Pick $(\boldsymbol{x}, \boldsymbol{y})$ according to $\mathsf{unif}^{n,m}_{W}$, subject to the constraint that $|\boldsymbol{x}^1 \cap \boldsymbol{y}^1| \leq \min\{n', m'\}$.
2. Pick uniformly at random $U \subseteq \boldsymbol{x}^1 \cap \boldsymbol{y}^0$ with $|U| = n - n'$ (this is always possible because $|\boldsymbol{x}^1 \cap \boldsymbol{y}^0| = n - |\boldsymbol{x}^1 \cap \boldsymbol{y}^1| \geq n - n'$).
3. Pick uniformly at random $V \subseteq \boldsymbol{x}^0 \cap \boldsymbol{y}^1$ with $|V| = m - m'$.
4. Pick uniformly at random $S' \subseteq \boldsymbol{x}^0 \cap \boldsymbol{y}^0$ with $|S'| = W/2 - |U| - |V|$ (this is always possible because $|\boldsymbol{x}^0 \cap \boldsymbol{y}^0| = W - |\boldsymbol{x}^1 \cup \boldsymbol{y}^1| \geq W - (n + m) > W/2$).
5. Finally, let $S = [W] - U - V - S'$ (note that $|S|$ is always $W/2$).

Denote by δ the probability that an input $(\boldsymbol{x}, \boldsymbol{y})$ drawn from $\mathsf{unif}_W^{n,w}$ satisfies $|\boldsymbol{x}^1 \cap \boldsymbol{y}^1| > \min\{n', m'\}$. Applying the fact $n' \geq n/2$, we know

$$\mathbf{Pr}[|\boldsymbol{x}^1 \cap \boldsymbol{y}^1| > n'] \leq \binom{n}{n/2}\binom{W - n/2}{m - n/2} / \binom{W}{m} \leq \binom{n}{n/2}\left(\frac{m - n/2}{W}\right)^{n/2} \leq (4/\sqrt{W})^{n/2},$$

and similarly $\mathbf{Pr}[|\boldsymbol{x}^1 \cap \boldsymbol{y}^1| > m'] \leq (4/\sqrt{W})^{m/2}$. Therefore,

$$\delta \leq \mathbf{Pr}[|\boldsymbol{x}^1 \cap \boldsymbol{y}^1| > n'] + \mathbf{Pr}[|\boldsymbol{x}^1 \cap \boldsymbol{y}^1| > m'] = o(1/W).$$

Observe that

$$\begin{aligned}
\alpha_{\mathsf{unif}_W^{n,m}}(\Pi) &= \mathbf{E}_{(\boldsymbol{x},\boldsymbol{y}) \leftarrow \mathsf{unif}_W^{n,m}}\left[\left|\Pi_A(\boldsymbol{x}, \boldsymbol{y})\right|\right] \\
&\geq (1 - \delta) \cdot \mathbf{E}_{(S,U,V) \leftarrow \nu} \mathbf{E}_{(\boldsymbol{x}', \boldsymbol{y}') \leftarrow \mathsf{unif}_{W/2}^{n',m'}}\left[\left|\Pi_A^{SUV}(\boldsymbol{x}', \boldsymbol{y}')\right|\right] \\
&= (1 - \delta) \cdot \mathbf{E}_{(S,U,V) \leftarrow \nu}\left[\alpha_{\mathsf{unif}_{W/2}^{n',m'}}(\Pi^{SUV})\right].
\end{aligned}$$

Turning this around gives

$$\mathbf{E}_{(S,U,V) \leftarrow \nu}\left[\alpha_{\mathsf{unif}_{W/2}^{n',m'}}(\Pi^{SUV})\right] \leq \frac{\alpha_{\mathsf{unif}_W^{n,m}}(\Pi)}{1 - \delta} = (1 + o(1)) \cdot \alpha_{\mathsf{unif}_W^{n,m}}(\Pi).$$

By Markov's inequality, if we draw (S, U, V) from ν, $\alpha_{\mathsf{unif}_{W/2}^{n',m'}}(\Pi^{SUV}) \leq 2.5(1 + o(1)) \cdot \alpha_{\mathsf{unif}_W^{n,m}}(\Pi) < 3\alpha_{\mathsf{unif}_W^{n,m}}(\Pi)$ holds with probability at least probability $1 - 1/2.5 = 0.6$. Similarly, with probability at least 0.6 we have $\beta_{\mathsf{unif}_{W/2}^{n',m'}}(\Pi^{SUV}) < 3\beta_{\mathsf{unif}_W^{n,m}}(\Pi)$. Therefore, both are true simultaneously with a positive probability, thus completing the proof.

Shared-Constraint Range Reporting

Sudip Biswas[1], Manish Patil[1], Rahul Shah[1], and Sharma V. Thankachan[2]

1 **Louisiana State University, USA**
{sbiswa7,mpatil,rahul}@csc.lsu.edu
2 **Georgia Institute of Technology, USA**
sharma.thankachan@gatech.edu

Abstract

Orthogonal range reporting is one of the classic and most fundamental data structure problems. (2,1,1) query is a 3 dimensional query with two-sided constraint on the first dimension and one sided constraint on each of the 2nd and 3rd dimension. Given a set of N points in three dimension, a particular formulation of such a $(2, 1, 1)$ query (known as four-sided range reporting in three-dimension) asks to report all those K points within a query region $[a, b] \times (-\infty, c] \times [d, \infty)$. These queries have overall 4 constraints. In Word-RAM model, the best known structure capable of answering such queries with optimal query time takes $O(N \log^{\epsilon} N)$ space, where $\epsilon > 0$ is any positive constant. It has been shown that any external memory structure in optimal I/Os must use $\Omega(N \log N / \log \log_B N)$ space (in words), where B is the block size [Arge et al., PODS 1999]. In this paper, we study a special type of $(2, 1, 1)$ queries, where the query parameters a and c are the same i.e., $a = c$. Even though the query is still four-sided, the number of independent constraints is only three. In other words, one constraint is shared. We call this as a *Shared-Constraint Range Reporting* (SCRR) problem. We study this problem in both internal as well as external memory models. In RAM model where coordinates can only be compared, we achieve linear-space and $O(\log N + K)$ query time solution, matching the best-known three dimensional dominance query bound. Whereas in external memory, we present a linear space structure with $O(\log_B N + \log \log N + K/B)$ query I/Os. We also present an I/O-optimal (i.e., $O(\log_B N + K/B)$ I/Os) data structure which occupies $O(N \log \log N)$-word space. We achieve these results by employing a novel divide and conquer approach. SCRR finds application in database queries containing sharing among the constraints. We also show that SCRR queries naturally arise in many well known problems such as top-k color reporting, range skyline reporting and ranked document retrieval.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases data structure, shared constraint, multi-slab, point partitioning

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.277

1 Introduction

Orthogonal range searching is one of the central data structure problems which arises in various fields. Many database applications benefit from the structures which answer range queries in two or more dimensions. Goal of orthogonal range searching is to design a data structure to represent a given set of N points in d-dimensional space, such that given an axis-aligned query rectangle, one can efficiently list all points contained in the rectangle. One simple example of orthogonal range searching data structure represents a set of N points in 1-dimensional space, such that given a query interval, it can report all the points falling within the interval. A balanced binary tree taking linear space can support such queries in optimal

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 277–290

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$O(\log N + K)$ time. Orthogonal range searching gets harder in higher dimensions and with more constraints. The hardest range reporting, yet having a linear-space and optimal time (or query I/Os in external memory) solution is the three dimensional dominance reporting query, also known as $(1, 1, 1)$ query [1] with one-sided constraint on each dimension. Here the points are in three-dimensions and the query asks to report all those points within an input region $[q_1, \infty) \times [q_2, \infty) \times [q_3, \infty)$. A query of the form $[q_1, q_2] \times [q_3, \infty)$ is known as $(2, 1)$ query, which can be seen as a particular case of $(1, 1, 1)$ query. However, four (and higher) sided queries are known to be much harder and no linear-space solution exists even for the simplest two dimensional case which matches the optimal query time of three dimensional dominance reporting. In Word-RAM model, the best result (with optimal query time) is $O(N \log^{\epsilon} N)$ words [10], where N is the number of points and $\epsilon > 0$ is an arbitrary small positive constant. In external memory, there exists an $\Omega(N \log N / \log \log_B N)$-space lower bound (and a matching upper bound) for any two-dimensional four-sided range reporting structure with optimal query I/Os [6]. Therefore, we cannot hope for a linear-space or almost-linear space structure with $O(\log_B N + K/B)$ I/Os for orthogonal range reporting queries with four or more constraints. The model of computation we assume is a unit-cost RAM with word size logarithmic in n. In RAM model, random access of any memory cell and basic arithmetic operations can be performed in constant time.

Motivated by database queries with constraint sharing and several well known problems (More details in Section 2), we study a special four sided range reporting query problem, which we call as the *Shared-Constraint Range Reporting* (SCRR) problem. Given a set $\mathcal{P}$ of N three dimensional points, the query input is a triplet (a, b, c), and our task is to report all those points within a region $[a, b] \times (-\infty, a] \times [c, \infty)$. We can report points within any region $[a, b] \times (-\infty, f(a)] \times [c, \infty)$, where $f(\cdot)$ is a pre-defined monotonic function (using a simple transformation). The query is four sided with only three independent constraints. Many applications which model their formulation as 4-sided problems actually have this sharing among the constraints and hence better bounds can be obtained for them using SCRR data structures. Formally, we have the following definition.

▶ **Definition 1.** A SCRR query $Q_{\mathcal{P}}(a, b, c)$ on a set $\mathcal{P}$ of three dimensional points asks to report all those points within the region $[a, b] \times (-\infty, a] \times [c, \infty)$.

The following theorems summarize our main results.

▶ **Theorem 2** (SCRR in Ram Model). *There exists a linear space RAM model data structure for answering SCRR queries on the set $\mathcal{P}$ in $O(\log N + K)$ time, where $N = |\mathcal{P}|$ and K is the output size.*

▶ **Theorem 3** (Linear space SCRR in External Memory). *SCRR queries on the set $\mathcal{P}$ can be answered in $O(\log_B N + \log \log N + K/B)$ I/Os using an $O(N)$-word structure, where $N = |\mathcal{P}|$, K is the output size and B is the block size.*

▶ **Theorem 4** (Optimal Time SCRR in External Memory). *SCRR queries on the set $\mathcal{P}$ can be answered in optimal $O(\log_B N + K/B)$ I/Os using an $O(N \log \log N)$-word structure, where $N = |\mathcal{P}|$, K is the output size and B is the block size.*

Our Approach. Most geometric range searching data structures use point partitioning scheme with appropriate properties, and recursively using the data structure for each partition. Our paper uses a novel approach of partitioning the points which seem to fit SCRR problem very well. Our data structure uses rank-space reduction on the given point-set, divide the SCRR query data structure based on small and large output size, takes advantage

of some existent range reporting data structure to obtain efficient solution and then bootstrap the data structure for smaller ranges.

Related Work. The importance of two-dimensional three-sided range reporting is mirrored in the number of publications on the problem. The general two-dimensional orthogonal range searching has been extensively studied in internal memory [2, 3, 4, 11, 12, 13, 9, 7]. The best I/O model solution to the three-sided range reporting problem in two-dimensions is due to Arge et al. [6], which occupies linear space and answers queries in $O(\log_B N + K/B)$ I/Os. Vengroff and Vitter [20] addressed the problem of dominance reporting in three dimensions in external memory model and proposed $O(N \log N)$ space data structure that can answer queries in optimal $O(\log_B N + K/B)$ I/Os. Recently, Afshani [1] improved the space requirement to linear space while achieving same optimal I/O bound. For the general two-dimensional orthogonal range reporting queries in external memory settings Arge et al. [6] gave $O((N/B)\log_2 N/\log_2 \log_B N)$ blocks of space solution achieving optimal $O(\log_B N + K/B)$ I/Os. Another external memory data structure is by Arge et al. [5] where the query I/Os is $O(\sqrt{N/B} + k/B)$ and the index space is linear. In the case when all points lie on a $U \times U$ grid, the data structure of Nekrich [19] answers range reporting queries in $O(\log\log_B U + K/B)$ I/Os. In [19] the author also described data structures for three-sided queries that use $O(N/B)$ blocks of space and answer queries in $O(\log\log_B U + K/B)$ I/Os on a $U \times U$ grid and $O(\log_B^{(h)} N)$ I/Os on an $N \times N$ grid for any constant $h > 0$. Very recently, Larsen and Pagh [17] showed that three-sided point reporting queries can be answered in $O(1 + K/B)$ I/Os using $O(N/B)$ blocks of space.

Outline. In section 2, we show how SCRR arises in database queries and relate SCRR problem to well known problems of colored range reporting, ranked document retrieval, range skyline queries and two-dimensional range reporting. In section 3 we discuss rank-space reduction of the input point-set to make sure no two points share the same x-coordinate. In section 4 we introduce a novel way to partition the point-set for answering SCRR queries which works efficiently for larger output size. Section 5 explains how to answer SCRR queries for smaller output size. Using these two data structures, section 6 obtains linear space and $O(\log N + K)$ time data structure for SCRR queries in RAM model thus proving theorem 2. Section 7 discusses SCRR queries in external memory, which includes a linear space but sub-optimal I/O and an optimal I/O but sub-optimal space data structures.

2 Applications

In this section, we show application of SCRR in database queries and list some of the well known problems, which could be directly reduced to SCRR. We start with two simple examples to illustrate shared constraint queries in database:

1. National Climatic Data Center contains data for various geographic locations. Sustained wind speed and gust wind speed are related to the mean wind speed for a particular time. Suppose we want to retrieve the stations having ($sustained\_wind\_speed, gust\_wind\_speed$) satisfying criteria 1: $mean\_wind\_speed < sustained\_wind\_speed < max\_wind\_speed$ and criteria 2: $gust\_wind\_speed < mean\_wind\_speed$. Here $mean\_wind\_speed$ and max_wind_speed comes as query parameters. Note that both these criteria have one constraint shared, thus effectively reducing number of independent constraints by one. By representing each station as the 2-dimensional point ($sustained\_wind\_speed$, $gust\_wind\_speed$), this query translates into the orthogonal range query specified by the

(unbounded) axis-aligned rectangle $[mean\_wind\_speed : max\_wind\_speed] \times (-\infty : mean\_wind\_speed]$.

2. Consider the world data bank which contains data for Gross domestic product (gdp), and we are interested in those countries that have gdp within the range of minimum and maximum gdp among all countries and gdp growth is greater than certain proportion of the minimum gdp. Our query might look like: $min\_gdp < gdp < max\_gdp$ and $c \times min\_gdp < gdp\_growth$, where c is a constant. Here min_gdp and max_gdp comes as query parameters. The constraint on gdp_growth is proportional to the lower constraint of gdp, which means the number of independent constraint is only two. This query can be similarly converted to orthogonal range reporting problem by representing each country as the point (gdp, gdp_growth), and asking to report all the points contained in the (unbounded) axis-aligned rectangle $[min\_gdp : max\_gdp] \times [c \times min\_gdp : \infty)$.

We can take advantage of such sharing among constraints to construct more effective data structure for query answering. This serves as a motivation for SCRR data structures. Below we show the relation between SCRR and some well known problems.

Colored Range Reporting. In colored range reporting, we are given an array A, where each element is assigned a color, and each color has a priority. For a query $[a, b]$ and a threshold c (or a parameter K) we have to report all distinct colors with priority $\geq c$ (or K colors with highest priority) within $A[a, b]$ [15]. We use the chaining idea by muthukrishnan [18] to reduce the colored range reporting to SCRR problem.

We map each element $A[i]$ to a weighted point (x_i, y_i) such that (1) $x_i = i$, (2) y_i is the highest $j < i$ such that both $A[i]$ and $A[j]$ have the same color (if such a y_i does not exist then $y_i = -\infty$) and (3) its weight w_i is same as the priority of color associated with $A[i]$. Then, the colored range reporting problem is equivalent to the following SCRR problem: report all points in $[a, b] \times (-\infty, a)$ with weight $\geq c$. By maintaining a additional linear space structure, for any given a, b and K, a threshold c can be computed in constant time such that number of colors reported is at least K and at most $\Omega(K)$ (we defer details to the full version). Then, by finding the Kth color with highest color among this (using selection algorithm) and filtering out colors with lesser priority, we shall obtain the top-K colors in additional $O(K/B)$ I/Os or $O(K)$ time.

Document Retrieval Problems. In string databases or in string retrieval systems, we have a collection $\mathcal{D}$ of documents (strings) of total length N. Define $score(P, d)$, the *score* of a document d with respect to a pattern P, which is a function of the locations of all P's occurrences in d. Then our goal is to preprocess $\mathcal{D}$ and maintain a structure such that, given a query pattern P and a threshold c, all those documents d_i with $score(P, d_i) \geq c$ can be retrieved efficiently. Hon et. al. [14] showed that the document retrieval problem can be reduced to the following problem: Given a collection of N intervals (y_i, x_i) with weights w_i and a query (a, b, c), output all the intervals such that $y_i \leq a \leq x_i \leq b$ and $w_i \geq c$. This is precisely the SCRR problem that we have investigated in this article.

Range Skyline Queries. Given a set S of N points in two-dimensions, a point (x_i, y_i) is said to be dominated by a point (x_j, y_j) if $x_i < x_j$ and $y_i < y_j$. Skyline of S is subset of S which consists of all the points in S which are not dominated by any other point in S. In Range-Skyline problem, the emphasis is to quickly generate those points within a query region R, which are not dominated by any other point in R. There exists optimal solutions

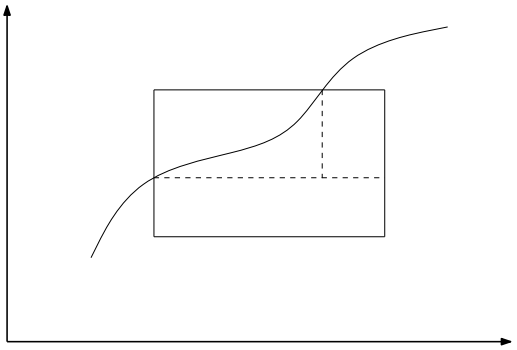


Figure 1 Special Two-dimensional Range Reporting Query.

in internal as well as external memory models for the case where R is a three-sided region of the form $[a, b] \times [c, +\infty)$ [16, 8].

We can reduce the range skyline query to SCRR by mapping each two-dimensional input point $p_i = (x_i, y_i)$ to a three-dimensional point x'_i, y'_i, z'_i as follows: (1) $x'_i = x_i$, (2) y'_i is the the x-coordinate of the leftmost point dominating p_i and (3) $z'_i = y_i$. Then range skyline query with three-sided region $[a, b] \times [c, +\infty)$ as input can be answered by reporting the output of SCRR query $[a, b] \times (-\infty, a] \times [c, +\infty)$.

Two-dimensional Range Reporting. Even though general four-sided queries are known to be hard as noted earlier, we can efficiently answer "special" four-sided queries efficiently. Any four-sided query with query rectangle R with one of its corners on the line $x = y$ can be viewed as a SCRR query. In fact any query rectangle R which intersect with $x = y$ line (or a predefined monotonic curve) can be reduced to SCRR (Figure 1).

3 Rank-Space Reduction of Points

We use rank-space reduction on the given point-set. Although rank-space reduction does not save any space for our data structure, it helps to avoid predecessor/successor search while querying and facilitate our partitioning technique. Without loss of generality, we assume that the points $p_i = (x_i, y_i, z_i) \in \mathcal{P}$ satisfy the following conditions: $x_i \leq x_{i+1}$ for all $i \in [1, N-1]$ and also $y_i \leq x_i$ for all $i \in [1, N]$. Note that $x_i \leq x_{i+1}$ can be ensured by sorting the point-set with respect to their x-coordinates and any point not satisfying $y_i \leq x_i$ can not be answer of our SCRR query, so we can remove them from consideration. In this section, we describe how to transform each point $p_i = (x_i, y_i, z_i) \in \mathcal{P}$ to a point $p'_i = (x'_i, y'_i, z'_i) \in \mathcal{P}'$ with the following additional properties guaranteed:

- Points in $\mathcal{P}'$ are on an $[1, N] \times [1, N] \times [1, N]$ grid (i.e., $x_i, y_i, z_i \in [1, N]$)
- $x'_i < x'_{i+1}$ for all $i \in [1, N-1]$. If $y_i \leq y_j$ (resp., $z_i \leq z_j$), then $y'_i \leq y'_j$ (resp., $z'_i \leq z'_j$) for all $i, j \in [1, N-1]$.

Such a mapping is given below: (1) The x-coordinate of the transformed point is same as the rank of that point itself. i.e., $x'_i = i$ (ties are broken arbitrarily), (2) Let $y_i \in (x_{k-1}, x_k]$, then $y'_i = k$, (3) Replace each z_i by the size of the set. i.e., $z'_i = \{j | z_j \leq z_i, j \in [1, N]\}$. We now prove the following lemma.

▶ **Lemma 5.** *If there exists an $S(N)$-space structure for answering SCRR queries on $\mathcal{P}'$ in optimal time in RAM model (or I/Os in external memory), then there exists an $S(N)+O(N)$-space structure for answering SCRR queries on $\mathcal{P}$ in optimal time (or I/Os).*

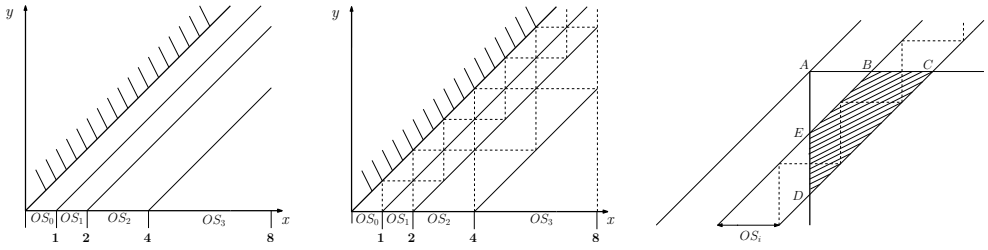


Figure 2 Point partitioning schemes: (a) Oblique slabs (b) Step partitions.

Proof. Assume we have an $S(N)$ space structure for SCRR queries on $\mathcal{P}'$. Now, whenever a query $Q_{\mathcal{P}}(a, b, c)$ comes, our first task is to identify the parameters a', b' and c' such that a point p_j is an output of $Q_{\mathcal{P}}(a, b, c)$ if and only if p'_j is an output of $Q_{\mathcal{P}}(a', b', c')$ and vice versa. Therefore, if point p'_j is an output for $Q_{\mathcal{P}}(a', b', c')$, we can simply output p_j as an answer to our original query. Based on our rank-space reduction, a', b' and c' are given as follows: (1) $x_{a'-1} < a \leq x_{a'}$ (assume $x'_0 = 0$), (2) $x_{b'} \leq b < x_{b'+1}$ (assume $x'_{N+1} = N + 1$), (3) Let z_j be the successor of c, then $c' = z'_j$.

By maintaining a list of all points in $\mathcal{P}$ in the sorted order of their x-coordinate values (along with a B-tree or binary search over it), we can compute a' and b' in $O(\log N)$ time(or $O(\log_B N)$ I/Os). Similarly, c' can also be computed using another list, where the points in $\mathcal{P}$ are arranged in the sorted order of z-coordinate value. The space occupancy of this additional structure is $O(N)$. Notice that this extra $O(\log N)$ time or $O(\log_B N)$ I/Os is optimal if we do not assume any thing about the coordinate values of points in $\mathcal{P}$. ◀

4 The Framework

In this section we introduce a new point partitioning scheme which will allow us to reduce the SCRR query into a logarithmic number of disjoint planar 3-sided or three dimensional dominance queries. From now onwards, we assume points in $\mathcal{P}$ to be in rank-space (Section 3). We begin by proving the result summarized in following theorem.

▶ **Lemma 6.** *By maintaining an $O(|\mathcal{P}|)$-word structure, any SCRR query $Q_{\mathcal{P}}(\cdot, \cdot, \cdot)$ can be answered in $O(\log^2 N + K)$ time in the RAM model, where K is the output size.*

For simplicity, we treat each point $p_i \in \mathcal{P}$ as a weighted point (x_i, y_i) in an $[1, N] \times [1, N]$ grid with z_i as its weight. The proposed framework utilizes divide-and-conquer technique based on the following partitioning schemes:

- *Oblique Slabs*: We partition the $[1, N] \times [1, N]$ grid into multi-slabs $OS_0, OS_1, ..., OS_{\lceil \log N \rceil}$ induced by lines $x = y + 2^i$ for $i = 0, 1, ...\lceil \log N \rceil$ as shown in figure 2(a). To be precise, OS_0 is the region between the lines $x = y$ and $x = y+1$ and OS_i for $i = 1, 2, 3, ..., \lceil \log N \rceil$ be the region between lines $x = y + 2^{i-1}$ and $x = y + 2^i$.
- *Step Partitions:* Each slab OS_i for $i = 1, 2, ...$ is further divided into regions with right-angled triangle shape (which we call as *tiles*) using axis parallel lines $x = (2^{(i-1)} * (1 + j))$ and $y = 2^{(i-1)} * j$ for $j = 1, 2, ...$ as depicted in Figure 2(b). OS_0 is divided using axis parallel lines $x = j$ and $y = j$ for $j = 1, 2, ...$Notice that the (axis parallel) boundaries of these triangles within any particular oblique slab looks like a step function.

Our partitioning scheme ensures property summarized by following lemma.

▶ **Lemma 7.** *Any region $[a, b] \times [1, a]$ intersects with at most $O(\log N)$ tiles.*

Proof. Let ϕ_i be the area of a tile in the oblique slab OS_i. Note that $\phi_0 = \frac{1}{2}$ and $\phi_i = \frac{1}{2}(2^{i-1})^2$ for $i \in [1, \lceil \log N \rceil]$. And let A_i be the area of the overlapping region between OS_i and the query region $[a, b] \times [1, a]$. Now our task is to simply show A_i/ϕ_i is a constant for all values of i. Assume $b = n$ in the extreme case. Then the overlapping region between OS_i and $[a, n] \times [1, a]$ will be trapezoid in shape and its area is given by $\phi_{i+1} - \phi_i$ (See Figure 2(c) for a pictorial proof). Therefore number of tiles needed for covering this trapezoidal region is $A_i/\phi_i = O(1)$. Which means the entires region can be covered by $O(\log N)$ tiles ($O(1)$ per oblique slab). ◀

In the light of the above lemma, a given SCRR query $Q_{\mathcal{P}}(a, b, c)$ can be decomposed into $O(\log N)$ subqueries of the type $Q_{\mathcal{P}_t}(a, b, c)$. Here $\mathcal{P}_t$ be the set of points within the region covered by a tile t. In the next lemma, we show that each of the $Q_{\mathcal{P}_t}(a, b, c)$ can be answered in optimal time (i.e., $O(\log |\mathcal{P}_t|)$ plus $O(1)$ time per output). Therefore, in total $O(N)$-space, we can maintain such structures for every tile t with at least one point within it. Then by combining with the result in lemma 7, the query $Q_{\mathcal{P}}(a, b, c)$ can be answered in $O(\log N * \log N + K) = O(\log^2 N + K)$ time, and lemma 6 follows.

▶ **Lemma 8.** *Let $\mathcal{P}_t$ be the set of points within the region covered by a tile t. Then a SCRR query $Q_{\mathcal{P}_t}(a, b, c)$ can be answered in $O(\log |\mathcal{P}_t| + k)$ time using a linear-space (i.e., $O(|\mathcal{P}_t|)$ words) structure, where k is the output size.*

Proof. The first step is to maintain necessary structure for answering all possible axis aligned three-dimensional dominance queries over the points in $\mathcal{P}_t$, which takes linear-space (i.e., $O(|\mathcal{P}_t|)$ words or $O(|\mathcal{P}_t| \log |\mathcal{P}_t|)$ bits). Let α and β be the starting and ending position of the interval obtained by projecting tile t to x-axis (see Figure 3). Then if the tile t intersects with the query region $[a, b] \times [1, a]$, then we have the following cases (see Figure 3):

1. $\alpha \leq a \leq \beta \leq b$: In this case, all points in $p_i \in \mathcal{P}_t$ implicitly satisfy the condition $x_i \leq b$. Therefore $Q_{\mathcal{P}_t}(a, b, c)$ can be obtained by a three sided query with $[a, N] \times [1, a] \times [c, N]$ as the input region or a two dimensional dominance query with $[a, N] \times [1, N] \times [c, N]$ as the input region (Figure 3(a)).
2. $a \leq \alpha \leq \beta \leq b$: In this case, all points in $p_i \in \mathcal{P}_t$ implicitly satisfy the condition $x_i \in [a, b]$. Therefore, $Q_{\mathcal{P}_t}(a, b, c)$ can be obtained by a two dimensional dominance query with $[1, N] \times [1, a] \times [c, N]$ as the input region (Figure 3(b)).
3. $a \leq \alpha \leq b \leq \beta$: In this case, all points in $p_i \in \mathcal{P}_t$ implicitly satisfy the condition $x_i \geq a$. Therefore $Q_{\mathcal{P}_t}(a, b, c)$ can be obtained by a three dimensional dominance query with $[1, b] \times [1, a] \times [c, N]$ as the input region (Figure 3(c)).
4. $\alpha \leq a \leq b \leq \beta$: Notice that the line between the points (a, a) and (b, a) are completely outside (and above) the tile t. Therefore, all points in $p_i \in \mathcal{P}_t$ implicitly satisfy the condition $y_i \leq a$. Therefore, $Q_{\mathcal{P}_t}(a, b, c)$ can be obtained by a three sided query with $[a, b] \times [1, N] \times [c, N]$ as the input region (Figure 3(d)).

Note that tiles can have two orientations. We have discussed four cases for one of the tile orientations. Cases for other orientation is mirror of the above four cases and can be handled easily. ◀

5 Towards $O(\log N + K)$ Time Solution

Our result in lemma 6 is optimal for $K \geq \log^2 N$. In this section, we take a step forward to achieve more efficient time solution for smaller values of K using multi-slab ideas. Using a parameter Δ (to be fixed later), we partition the $[1, N] \times [1, N]$ grid into $L = \lceil N/\Delta \rceil$ vertical

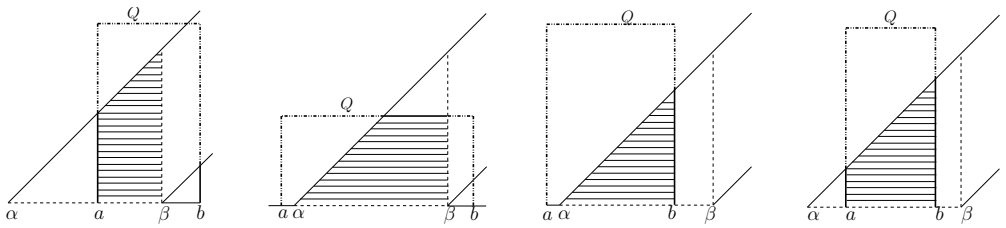


Figure 3 $Q_{\mathcal{P}}(a, b, c)$ and tile t intersections.

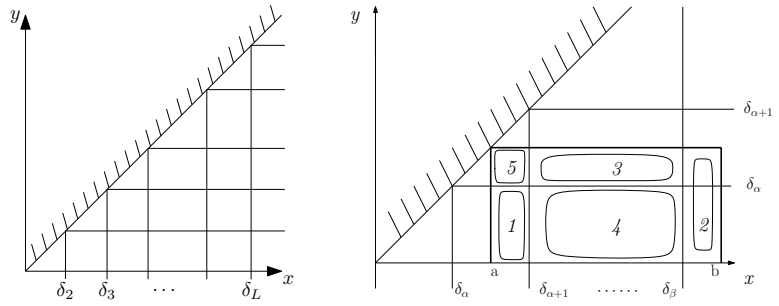


Figure 4 Divide-and-conquer scheme using Δ

slabs (Figure 4(a)). Multi-slabs $VS_0, VS_1, ..., VS_L$ are the slabs induced by lines $x = i\Delta$ for $i = 0, 1, ..., L$. Denote by δ_i $(i \in [0, L])$ the minimum x-coordinate in VS_i. For notational convenience, we define $\delta_{L+1} = \infty$. By slight abuse of notation, we use VS_i to represent the set of points in the corresponding slab.

A query $Q_{\mathcal{P}}$ with (a, b, c) as an input is called *inter-slab* query if it overlaps two or more vertical slabs, otherwise if it is entirely contained within a single vertical slab we call it an *intra-slab* query. In this section, we propose a data structure that can answer inter-slab queries optimally.

▶ **Lemma 9.** *Inter-slab SCRR queries can be answered in $O(\log N + K)$ time in RAM model using a data structure occupying $O(N)$ words space, where K represents the number of output.*

Proof. Given a query $Q_{\mathcal{P}}(a, b, c)$ such that $a \leq b$, let α, β be integers that satisfy $\delta_\alpha \leq a < \delta_{\alpha+1}$ and $\delta_\beta \leq b < \delta_{\beta+1}$. The x interval of an inter-slab query i.e. $[a, b]$ spreads across at least two vertical slabs. Therefore, $Q_{\mathcal{P}}$ can be decomposed into five subqueries $Q^1_{\mathcal{P}}$, $Q^2_{\mathcal{P}}$, $Q^3_{\mathcal{P}}$, $Q^4_{\mathcal{P}}$ and $Q^5_{\mathcal{P}}$ as illustrated in Figure 4(b). These subqueries are defined as follows.

- $Q^1_{\mathcal{P}}$ is the part of $Q_{\mathcal{P}}$ which is in $[\delta_\alpha, \delta_{\alpha+1}) \times [1, \delta_\alpha) \times [c, N]$.
- $Q^2_{\mathcal{P}}$ is the part of $Q_{\mathcal{P}}$ which is in $[\delta_\beta, \delta_{\beta+1}) \times [1, \delta_{\alpha+1}) \times [c, N]$.
- $Q^3_{\mathcal{P}}$ is the part of $Q_{\mathcal{P}}$ which is in $[\delta_{\alpha+1}, \delta_\beta) \times [\delta_\alpha, \delta_{\alpha+1}) \times [c, N]$.
- $Q^4_{\mathcal{P}}$ is the part of $Q_{\mathcal{P}}$ which is in $[\delta_{\alpha+1}, \delta_\beta) \times [1, \delta_\alpha) \times [c, N]$.
- $Q^5_{\mathcal{P}}$ is the part of $Q_{\mathcal{P}}$ which is in $[\delta_\alpha, \delta_{\alpha+1}) \times (\delta_\alpha, \delta_{\alpha+1}) \times [c, N]$.

If $\alpha + 1 = \beta$ then we only need to consider subqueries $Q^1_{\mathcal{P}}$, $Q^2_{\mathcal{P}}$ and $Q^5_{\mathcal{P}}$. Each of these subqueries can now be answered as follows.

Answering $Q^1_{\mathcal{P}}$. The subquery $Q^1_{\mathcal{P}}$ can be answered by retrieving all points in $VS_\alpha \cap [a, N] \times [1, N]$ with weight $\geq c$. This is a two-dimensional dominance query in VS_α. This can be achieved by maintaining a three-dimensional dominance query structure for RAM model [1] for the points in VS_i for $i = 1, \ldots, L$ separately. The query time will be $O(\log |VS_\alpha| + K_1) = O(\log N + K_1)$, where K_1 is the output size and index space is $O(\sum_{i=1}^{L} |VS_i|) = O(N)$ words.

Answering $Q^2_{\mathcal{P}}$. To answer subquery $Q^2_{\mathcal{P}}$ we will retrieve all points in $VS_\beta \cap [1, b) \times [1, a)$ with weight $\geq c$. By maintaining a collection of three-dimensional dominance query structures [1] occupying linear space overall, $Q^2_{\mathcal{P}}$ can be answered in $O(\log N + K_2)$ time, where K_2 is the output size.

Answering $Q^3_{\mathcal{P}}$. To answer subquery $Q^3_{\mathcal{P}}$, we begin by partitioning the set of points $\mathcal{P}$ into L horizontal slabs $HS_1, HS_2, \ldots, HS_L$ induced by lines $y = i\Delta$, such that $HS_i = \mathcal{P} \cap [\delta_{i+1}, N] \times [\delta_i, \delta_{i+1})$. The subquery $Q^3_{\mathcal{P}}$ can now be answered by retrieving all points in $HS_\alpha \cap [1, \delta_\beta) \times [1, a)$ with weight $\geq c$. This can be achieved by maintaining a three-dimensional dominance query structure [1] for the points in HS_i for $i = 1, ..., L$ separately. Since each point in S belongs to at most one HS_i the overall space can be bounded by $O(N)$ words and the query time can be bounded by $O(\log |HS_\alpha|) + K_3) = O(\log N + K_3)$ time, with K_3 being the number of output.

Answering $Q^5_{\mathcal{P}}$. To answer subquery $Q^5_{\mathcal{P}}$ we will retrieve all points in $VS_\alpha \cap (a, N] \times [1, a)$ with weight $\geq c$. By maintaining a collection of three-dimensional dominance query structures occupying linear space overall as described in earlier subsections, $Q^5_{\mathcal{P}}$ can be answered in $O(\log |VS_\alpha| + K_5) = O(\log N + K_5)$ time, where K_5 is the output size.

Answering $Q^4_{\mathcal{P}}$. We begin by describing a naive way of answering $Q^4_{\mathcal{P}}$ by using a collection of three-dimensional dominance query structures built for answering $Q^1_{\mathcal{P}}$. We query VS_i to retrieve all the points in $VS_i \cap [1, N] \times [1, \delta_\alpha)$ with weight $\geq c$ for $i = \alpha + 1, ..., \beta - 1$. Such a query execution requires $O((\beta - \alpha + 1) \log N + K_4)$ time, where K_4 is the output size. We are required to spend $O(\log N)$ time for each vertical slab even if the query on a particular VS_i does not produce any output. To answer subquery $Q^4_{\mathcal{P}}$ in $O(\log N + K_4)$ time, we make following crucial observations: (1) All three boundaries of $Q^4_{\mathcal{P}}$ are on the partition lines, (2) The left boundary of $Q^4_{\mathcal{P}}$ (i.e., line $x = \delta_{\alpha+1}$) is always the successor of the top boundary (i.e., line $y = \delta_\alpha$), (3) The output size is bounded by $O(\log^2 N)$.

We use these observations to construct following data structure: Since the top left and bottom right corner of $Q^4_{\mathcal{P}}$ falls on the partition lines, there are at most $(N/\Delta)^2$ possible different rectangles for $Q^4_{\mathcal{P}}$. For each of these we store at most top-$O(\log^2 N)$ points in sorted order of their weight. Space requirement of this data structure is $O((N/\Delta)^2 \log^2 N)$ words. Query algorithm first identifies the rectangle that matches with $Q^4_{\mathcal{P}}$ among $(N/\Delta)^2$ rectangles and then simply reports the points with weight greater than c in optimal time. Finally to achieve linear space, we choose $\Delta = \sqrt{N} \log N$.

Thus, we can obtain $K = K_1 + K_2 + K_3 + K_4 + K_5$ output in $O(K)$ time. Also, in the divide and conquer scheme, the point sets used for answering subqueries $Q^1_{\mathcal{P}}, ..., Q^5_{\mathcal{P}}$ are disjoint, hence all reported answers are unique. Now given a query $Q_{\mathcal{P}}(a, b, c)$, if the subquery $Q^4_{\mathcal{P}}$ in the structure just described returns $K_4 = \log^2 N$ output, it suggest that output size $K > \log^2 N$. Therefore, we can query the structure in lemma 6 and still retrieve all output in optimal time. This completes the proof of lemma 9. ◀

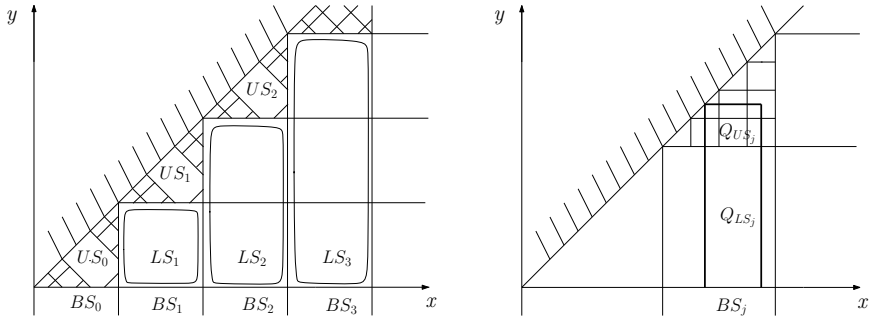


Figure 5 Optimal time SCRR query data structure.

6 Linear Space and $O(\log N + K)$ Time Data Structure in RAM Model

In this section, we show how to obtain our $O(\log N + K)$ time result stated in Theorem 2 via bootstrapping our previous data structure.

We construct $\psi_1, \psi_2, ..., \psi_{\lceil \log\log N \rceil}$ levels of data structures, where $\psi_1 = \sqrt{N}\log N$ and $\psi_i = \sqrt{\psi_{i-1}}\log\psi_{i-1}$, for $i = 2, 3, ..., \lceil \log\log N \rceil$. At each level, we use multi-slabs to partition the points. More formally, at each level ψ_i, the $[1, N] \times [1, N]$ grid is partitioned into $g = \lceil N/\psi_i \rceil$ vertical slabs or multi-slabs. At level ψ_i, multi-slabs $BS_0, BS_1, ..., BS_g$ are the slabs induced by lines $x = j\psi_i$ for $j = 0, 1, ..., g$. Each multi-slab BS_j is further partitioned into disjoint upper partition US_j and lower partition LS_j (Figure 5). Below we describe the data structure and query answering in details.

For a multi-slab BS_j and a $SCRR$ query, US_j can have more constraints than LS_j, making it more difficult to answer query on US_j. Our idea is to exclude the points of US_j from each level, and build subsequent levels based on only these points. Query answering for LS_j can be done using the inter-slab query data structure and three-sided range reporting data structure.

At each level ψ_i, we store the data structure described in Lemma 9 capable of answering inter-slab queries by taking slab width $\Delta = \sqrt{\psi_i}\log\psi_i$. Also we store separate three-sided range reporting data structures for each LS_j, $j = 0, 1, ..., g$. The points in US_j at level ψ_i (for $i = 1, \cdots, \lceil \log\log N \rceil - 1$) are removed from consideration. These removed points are considered in subsequent levels. Note that the inter-slab query data structure stored here is slightly different from the data structure described in lemma 9, since we removed the points of US_j (region 5 of figure 4b). $\psi_{\lceil \log\log N \rceil}$ is the bottom level and no point is removed from here. Level ψ_{i+1} contains only the points of all the US_j partitions from previous level ψ_i, and again the upper partition points are removed at level ψ_{i+1}. More specifically, level ψ_1 contains an inter-slab query data structure and $\sqrt{N}\log N$ number of separate two-dimensional three-sided query data structures over each of the lower partitions LS_j. Level ψ_2 contains $\sqrt{N}\log N$ number of data structures similar to level ψ_1 corresponding to each of the upper partitions US_j of level ψ_1. Subsequent levels are constructed in a similar way. No point is repeated at any level, two-dimensional three-sided query data structures and inter-slab query data structures take linear space, giving linear space bound for the entire data structure.

A SCRR query $Q_{\mathcal{P}}$ can be either an inter-slab or an intra-slab query at level ψ_i (illustrated in figure 6). An intra-slab SCRR query $Q_{\mathcal{P}}$ can be divided into Q_{US_i} and Q_{LS_i}. Q_{LS_i} is

three-sided query in LS_i, which can be answered by the three-sided range reporting structure stored at LS_i. Q_{US_i} is issued as a SCRR query for level ψ_{i+1} and can be answered by traversing at most $\lceil \log\log N \rceil$ levels. An inter-slab SCRR query $Q_{\mathcal{P}}$ can be decomposed into 5 sub-queries: Q_1, Q_2, Q_3, Q_4 and Q_5. Q_1, Q_2, Q_3 and Q_4 can be answered using the optimal inter-slab query data structure of lemma 9 in similar way described in details in section 5. Again Q_5 is issued as a SCRR query for level ψ_{i+1} and can be answered in subsequent levels. At each level $O(\log \psi_i + K_i)$ time is needed where K_i is the output size obtained at level ψ_i. Since no point is repeated at any level, all reported answers are unique. At most $\log\log N$ levels need to be accessed for answering $Q_{\mathcal{P}}$. Total time is bounded by $O(\sum_{i=1}^{\log\log N}(\log \psi_i) + \log\log N + \sum_{i=1}^{\log\log N}(K_i)) = O(\log N + K)$, thus proving theorem 2.

7 SCRR Query in External Memory

In this section we discuss two external memory data structures for SCRR query, one achieving optimal I/O and another achieving linear space. Both these data structures are obtained by modifying our RAM model data structure. We use the multi-slab ideas similar to section 5. We assume points in $\mathcal{P}$ to be in rank-space. We begin by stating external memory variants of lemma 6 and lemma 9.

▶ **Lemma 10.** *By maintaining an $O(N)$-word structure, any SCRR query $Q_{\mathcal{P}}(\cdot,\cdot,\cdot)$ can be answered in $O(\log^2(N/B) + K/B)$ I/Os, where K is the output size.*

Proof. We use $\lceil \log(N/B) \rceil$ number of oblique slabs induced by lines $x = y + 2^i B$ for $i = 0, 1, ...\lceil \log(N/B) \rceil$. Each oblique slab is partitioned into tiles using axis parallel lines $x = (2^{(i-1)} * (1 + j))B$ and $y = 2^{(i-1)} * jB$ for $j = 1, 2, ...$ It can be easily shown that any SCRR query $Q_{\mathcal{P}}(a, b, d)$ intersects with at most $O(\log(N/B))$ tiles, each of which can be resolved in linear space and optimal I/Os using three-dimensional query structure [1] in each tile, achieving $O(\log^2(N/B) + K/B)$ total I/Os. ◀

▶ **Lemma 11.** *Inter-slab SCRR queries can be answered in $O(\log_B N + K/B)$ I/Os using a data structure occupying $O(N)$ words space, where K represents the number of outputs.*

Proof. This can be achieved by using a data structure similar to the one described in lemma 9 with $\Delta = \sqrt{NB}\log_B N$. We use external memory counterparts for three sided and three dimensional dominance reporting and for answering query Q_p^4 we maintain top $O(log_B^2 N)$ points from each of the rectangle. ◀

7.1 Linear Space Data Structure

The linear space data structure is similar to the RAM model linear space and optimal time structure described in section 6. Major difference is we use $\psi_i = \sqrt{\psi_{i-1}B}\log_B \psi_{i-1}$ for bootstrapping and use the external memory counterparts of the data structures.

We construct $\psi_1, \psi_2, ..., \psi_{\lceil \log\log N \rceil}$ levels of data structures, where $\psi_1 = \sqrt{NB}\log_B N$ and any $\psi_i = \sqrt{\psi_{i-1}B}\log_B \psi_{i-1}$. At each level ψ_i, the $[1, N] \times [1, N]$ grid is partitioned into $g = \lceil N/\psi_i \rceil$ vertical slabs. Multi-slabs BS_j, upper partition US_j and lower partition LS_j for $j = 0, 1, ..., g$ are defined similar to section 6.

At each level ψ_i, we store the data structure described in Lemma 11 capable of answering inter-slab queries in optimal I/Os by taking slab width $\Delta = \sqrt{\psi_i B}\log_B \psi_i$. Also we store separate three-sided external memory range reporting data structures for each LS_j, $j = 0, 1, ..., g$. In order to maintain linear space, the points in US_j at level ψ_i (for $i = 1, \cdots, \lceil \log\log N \rceil - 1$)

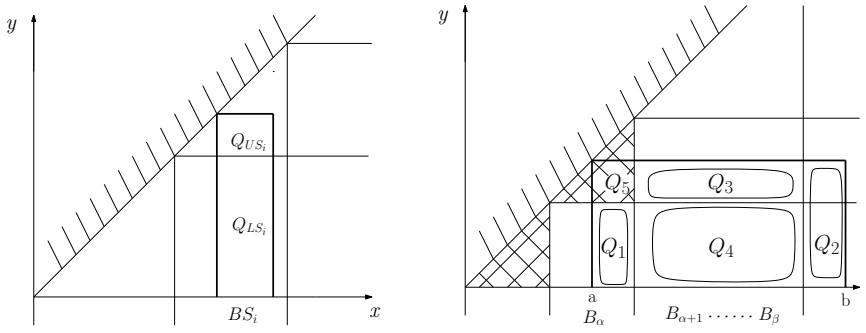


Figure 6 Intra-slab and Inter-slab query for linear space data structure.

are removed from consideration. These removed points are considered in subsequent levels. $\psi_{\lceil \log\log N \rceil}$ is the bottom level and no point is removed from here. Level ψ_{i+1} contains only the points of all the US_j partitions from previous level ψ_i, and again the upper partition points are removed at level ψ_{i+1}. External memory two-dimensional three-sided query data structures and optimal inter-slab query data structures for external memory take linear space, and we avoided repetition of points in different levels, thus the overall data structure takes linear space.

Query answering is similar to section 6. If the SCRR query $Q_{\mathcal{P}}$ is intra-slab, then $Q_{\mathcal{P}}$ can be divided into Q_{US_i} and Q_{LS_i}. Q_{LS_i} is three-sided query in LS_i, which can be answered by the three-sided range reporting structure stored at LS_i. Q_{US_i} is issued as a SCRR query for level ψ_{i+1} and can be answered by traversing at most $\lceil \log\log N \rceil$ levels. An inter-slab SCRR query $Q_{\mathcal{P}}$ can be decomposed into 5 sub-queries: Q_1, Q_2, Q_3, Q_4 and Q_5. Q_1, Q_2, Q_3 and Q_4 can be answered using the optimal inter-slab query data structure of lemma 11 in similar way described in details in section 5. Again Q_5 is issued as a SCRR query for level ψ_{i+1} and can be answered in subsequent levels. At each level $O(\log_B \psi_i + K_i/B)$ I/Os are needed where K_i is the output size obtained at level ψ_i. Since no point is repeated at any level, all reported answers are unique. At most $\log\log N$ levels need to be accessed for answering $Q_{\mathcal{P}}$. Total I/O is bounded by $O(\sum_{i=1}^{\log\log N}(\log_B \psi_i) + \log\log N + \sum_{i=1}^{\log\log N}(K_i/B))$ $= O(\log_B N + \log\log N + K/B)$ thus proving theorem 3.

7.2 I/O Optimal Data Structure

I/O optimal data structure is quite similar to the linear space data structure. The major difference is the points in US_j at level ψ_i (for $i = 1, \cdots, \lceil \log\log N \rceil - 1$) are not removed from consideration. Instead at each US_j we store external memory data structure capable of answering inter-slab query optimally(Lemma 11). All the points of US_j $(j = 0, 1, ..., g)$ of level ψ_i (for $i = 1, \cdots, \lceil \log\log N \rceil - 1$) are repeated in the next level ψ_{i+1}. This will ensure that we will have to use only one level to answer the query. For LS_j $(j = 0, 1, ..., g)$, we store three-sided query structures. Since there are $\log\log N$ levels, and at each level data structure uses $O(N)$ space, total space is bounded by $O(N \log\log N)$. To answer a query $Q_{\mathcal{P}}$, we query the structures associated with ψ_i such that $Q_{\mathcal{P}}$ is an intra-slab query for ψ_i and is inter-slab for ψ_{i+1}. We can decompose the query $Q_{\mathcal{P}}$ into Q_{US} and Q_{LS}, where $Q_{US}(Q_{LS})$ falls completely within $US_j(LS_j)$. Since, Q_{US} is an inter-slab query for the inter-slab query data structure stored at US_j, it can be answered optimally. Also Q_{LS} is a simple three-sided query for which output can be retrieved in optimal time using the three-sided structure of LS_j. This completes the proof for Theorem 4.

8 Conclusions

In many applications which require range queries, some of the input constraints are shared and not really independent. We give first non trivial indexes for handling such cases breaking the currently known $O(N \log^{\epsilon} N)$ space barrier for four-sided queries in Word-RAM model. In Word-RAM model, we obtained linear space and optimal time index for answering SCRR queries. Our optimal I/O index in external memory takes $O(N \log \log N)$ words of space and answer queries optimally. We also present a linear space index for external memory. We leave it as an open problem to achieve optimal space bounds, avoiding the $O(\log \log N)$ blowup in external memory model. Also it will be interesting to see whether such results can be obtained in Cache Oblivious model.

Acknowledgements. This work is supported by US NSF Grants CCF–1017623, CCF–1218904.

References

1 Peyman Afshani. On dominance reporting in 3d. In *ESA*, pages 41–51, 2008.

2 Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting in three and higher dimensions. In *FOCS*, pages 149–158, 2009.

3 Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Symposium on Computational Geometry*, pages 240–246, 2010.

4 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *FOCS*, pages 198–207, 2000.

5 Lars Arge, Mark de Berg, Herman J. Haverkort, and Ke Yi. The priority r-tree: A practically efficient and worst-case optimal r-tree. In *SIGMOD Conference*, pages 347–358, 2004.

6 Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On two-dimensional indexability and optimal range search indexing. In *PODS*, pages 346–357, 1999.

7 Jon Louis Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

8 Gerth Stølting Brodal and Kasper Green Larsen. Optimal planar orthogonal skyline counting queries. *CoRR*, abs/1304.7959, 2013.

9 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the ram, revisited. In *Symposium on Computational Geometry*, pages 1–10, 2011.

10 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.

11 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.

12 Bernard Chazelle. Lower bounds for orthogonal range searching i. the reporting case. *J. ACM*, 37(2):200–212, 1990.

13 Bernard Chazelle. Lower bounds for orthogonal range searching ii. the arithmetic model. *J. ACM*, 37(3):439–463, 1990.

14 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *J. ACM*, 61(2):9, 2014.

15 Marek Karpinski and Yakov Nekrich. Top-k color queries for document retrieval. In *SODA*, pages 401–411, 2011.

16 Casper Kejlberg-Rasmussen, Yufei Tao, Konstantinos Tsakalidis, Kostas Tsichlas, and Jeonghun Yoon. I/o-efficient planar range skyline and attrition priority queues. In *PODS*, pages 103–114, 2013.

17 Kasper Green Larsen and Rasmus Pagh. I/o-efficient data structures for colored range and prefix reporting. In *SODA*, pages 583–592, 2012.

18 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 657–666, 2002.

19 Yakov Nekrich. External memory range reporting on a grid. In *ISAAC*, pages 525–535, 2007.

20 Darren Erik Vengroff and Jeffrey Scott Vitter. Efficient 3-d range searching in external memory. In *STOC*, pages 192–201, 1996.

Optimal Broadcasting Strategies for Conjunctive Queries over Distributed Data

Bas Ketsman* **and Frank Neven**

Hasselt University and transnational University of Limburg
Belgium
first.lastname@uhasselt.be

Abstract

In a distributed context where data is dispersed over many computing nodes, monotone queries can be evaluated in an eventually consistent and coordination-free manner through a simple but naive broadcasting strategy which makes all data available on every computing node. In this paper, we investigate more economical broadcasting strategies for full conjunctive queries without self-joins that only transmit a part of the local data necessary to evaluate the query at hand. We consider oblivious broadcasting strategies which determine which local facts to broadcast independent of the data at other computing nodes. We introduce the notion of broadcast dependency set (BDS) as a sound and complete formalism to represent local optimal oblivious broadcasting functions. We provide algorithms to construct a BDS for a given conjunctive query and study the complexity of various decision problems related to these algorithms.

1998 ACM Subject Classification H.2.3 Query Languages, H.2.4 Distributed databases

Keywords and phrases Coordination-free evaluation, conjunctive queries, broadcasting

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.291

1 Introduction

We assume the setting introduced in the context of declarative networking [6, 14], where queries are specified on a logical level over a global schema and are evaluated by multiple computing nodes over which the input database is distributed. These nodes can perform local computations and communicate asynchronously with each other via messages. The model then operates under the assumption that messages can never be lost but can be arbitrarily delayed. It is known that every monotone query can be evaluated in an eventually consistent and coordination-free manner through a naive broadcasting strategy that makes all data available to all nodes [14].[1] Indeed, every computing node sends all its local data to every other node and reevaluates the query every time new data arrives. This evaluation is eventually consistent as, because of monotonicity, no facts will be derived which later have to be retracted and, furthermore, when all transmitted data has arrived, the output of every node will correspond to the result of the query. In addition, the computation requires no coordination between the nodes.

Obviously, the above strategy leads to a very careless evaluation as the whole database is send to every node and every node independently computes the complete answer for the targeted query. In the present paper, we are interested in more economical broadcasting

* PhD Fellow of the Research Foundation – Flanders (FWO)

[1] Actually, this observation is the straightforward part of the CALM-conjecture [14]. It is the converse direction which is more surprising: that every query which can be evaluated in an eventually consistent and coordination-free manner has to be monotone [6].

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 291–307

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

strategies where only a subset of the local data is transmitted and where each computing node contributes to the answer of the query by outputting only a subset of the answer tuples. The result of the query then is the union of the tuples output by the computing nodes. In particular, we focus on full conjunctive queries without self-joins and we consider *oblivious broadcasting strategies* where every computing node determines which facts will be broadcast solely on the content of its own local database (so, oblivious of the data at other nodes). These facts are referred to as *broadcast facts*. Facts that are not initially broadcast are called *static*. We illustrate the ideas behind such strategies by means of an example.

▶ **Example 1.** Let Q_1 be the query $Q_1(x, y, z) \leftarrow A(x, y), B(y, x), C(x, z)$ and let $I = \{A(1, 2), A(2, 2), B(2, 1), B(2, 2), B(4, 4), C(1, 3)\}$ be a database instance. Consider a network of two computing nodes c and c' containing the facts $I(c) = \{A(2, 2), B(2, 1), B(2, 2)\}$ and $I(c') = \{A(1, 2), B(4, 4), C(1, 3)\}$, respectively.

Naive broadcasting strategy. The naive broadcasting algorithm outlined above sends all facts in $I(c)$ to c' and all facts in $I(c')$ to c. Eventually, both c and c' receive all data and both of them compute the result of the query, that is, $Q_1(I) = \{(1, 2, 3)\}$.

Improved oblivious broadcasting strategy. The just described strategy is clearly oblivious but also rather wasteful. Therefore consider the following strategy which broadcasts all of the C-facts but none of the A-facts. Furthermore, a B-fact $B(i, j)$ is broadcast only when $A(j, i)$ does not occur in the local database. Executing this strategy for every computing node in our example results in c broadcasting the set $\{B(2, 1)\}$ while c' broadcasts $\{B(4, 4), C(1, 3)\}$. So, eventually, $I^*(c) = \{A(2, 2), B(2, 1), B(2, 2), B(4, 4), C(1, 3)\}$ and $I^*(c') = \{A(1, 2), B(2, 1), B(4, 4), C(1, 3)\}$. Here, we denote by $I^*(d)$ the instance at node d when all transmitted messages have arrived. Therefore, $Q_1(I^*(c)) = \emptyset$ and $Q_1(I^*(c')) = \{(1, 2, 3)\}$, and $Q_1(I)$ equals $Q_1(I^*(c)) \cup Q_1(I^*(c'))$. Intuitively, this strategy is correct in general as the following invariant holds for every computing node d: when a fact $B(i, j)$ is *not* broadcast at a node d, then every satisfying valuation V for Q on I that maps (x, y) to (i, j) can be realized locally in $I^*(d)$. Notice that, a similar strategy reversing the roles of A- and B-facts would work as well.

We will formalize oblivious broadcasting functions as generic mappings. This means that decisions on whether to broadcast facts do not depend only on the name of the predicate but can also depend on the equality type of the fact under consideration. Therefore, the following strategy would be valid as well: always broadcast facts of the form $C(i, j)$ with $i \neq j$ and keep all facts of the form $C(i, i)$ static; broadcast all B-facts; broadcast a fact $A(i, j)$ only when the fact $C(i, i)$ is not present in the local database. While not immediately obvious, this strategy correctly computes Q on every distributed database.

Both strategies will be presented more formally in Section 5 in terms of *broadcast dependency sets* and are formalized further in Example 12(1) and 12(3). ◀

In this paper, we make the following contributions:

(i) We provide a semantical characterization of when an oblivious broadcasting function (OBF) correctly evaluates a given conjunctive query. While it is desirable to construct OBFs that minimize the overall amount of transmitted facts over all distributed databases, we show that there is no optimal OBF for any conjunctive query with at least two distinct atoms in its body. Therefore, we turn to a slightly weaker notion of optimality, called local optimal, which requires that an OBF is optimal w.r.t. the local instance at every computing node. Intuitively, this means that no broadcast fact can be made static without sacrificing correctness. We provide a semantical characterization for when an OBF is local optimal for a given conjunctive query.

(ii) We introduce the notion of a broadcast dependency set (BDS) as a formalism to specify OBFs. In brief, a BDS $\mathcal{S}$ is a set of pairs (τ, T) where τ is a partial atomic type and T is a set of partial atomic types. Every such pair encodes a rule that can be interpreted roughly as follows: when a fact $\mathbf{f}$ matches type τ, it will be broadcast at a computing node c when the set of facts induced by T is not present at c. We present necessary and sufficient syntactic conditions for when a BDS is correct for a given query and also for when it is local optimal w.r.t. that query. Furthermore, we study the complexity of deciding whether a BDS is correct for a query and whether it is local optimal. Finally, and most importantly, we show that the formalism of BDS is expressively complete w.r.t. local optimal OBFs by obtaining that every local optimal OBF can be represented by a BDS. In fact, every local optimal OBF can already be represented by a BDS that only uses complete types, that is, types where the equalities between all variables are fully specified.

(iii) Based on the syntactic criteria of when a BDS is correct for Q and when it is local optimal, we obtain an algorithm BDS-BUILD(Q) that computes a local optimal OBF (represented as a BDS) for a given conjunctive query Q. When restricting to open types (these are types without restrictions on the equalities between variables), BDS-BUILD(Q) computes a local optimal OBF in time polynomial in the size of Q. When considering complete types, BDS-BUILD(Q) computes a local optimal OBF in time exponential in the size of Q simply because there are exponentially many complete types.

Outline. We discuss related work in Section 2 and introduce the necessary definitions and concepts in Section 3. In Section 4, we discuss oblivious broadcasting functions and local optimality. In Section 5, we discuss broadcast dependency sets and study their properties. In Section 6, we provide an algorithm to construct a local optimal oblivious broadcasting function for a given conjunctive query. We conclude in Section 7.

2 Related Work

CALM. The approach in this paper is motivated by the work on the CALM-conjecture. Hellerstein [14] formulated the CALM-principle which suggests a link between logical monotonicity and distributed consistency without the need for coordination. The latter principle is, for instance, embedded in BLOOM, a declarative language for distributed programming, for which practical program analysis technique have been developed detecting potential consistency anomalies [3, 4, 11]. Ameloot et al. [6] formalized (and proved) the CALM-conjecture in terms of relational transducer networks. Zinn et al. [19] showed that the generalization of the conjecture to stronger variants of relational transducer networks fails. Ameloot et al. [5] then subsequently provided a more fine-grained answer to the CALM-conjecture by relating these stronger variants of relational transducer networks to weaker notions of monotonicity. All of these works considered naive evaluation strategies that broadcast *all* of the local data. In particular, none of these works considered more economic broadcasting evaluation of conjunctive queries.

Massive parallel model. The networked relational transducer model is just one paradigm for studying distributed query evaluation. In the massively parallel (MP) model, introduced by Koutris and Suciu [15], computation proceeds in a sequence of parallel steps, each followed by global synchronization of all servers. In this model, evaluation of conjunctive queries [15, 7] as well as skyline queries [2] have been considered. Recently, Beame et al. [8] proved a

matching upper and lower bound for the amount of communication needed to compute a full conjunctive query without self-joins in one communication round. The upper bound is provided by a randomized algorithm called Hypercube which dates back to Ganguli et al. [13] and was described by Afrati and Ullman [1] in the context of MapReduce algorithms. Hypercube is motivated by modern massively distributed systems like, for instance, Spark [18], where entire computations reside in main memory, replay is used to recover, and the dominant cost is that of communication. We note that one-round Hypercube is coordination-free and can be easily employed within the framework of relational transducer networks as well. A characteristic of Hypercube-style algorithms is that the space of computing nodes (over which the input data will be distributed) needs to be known in advance. The broadcasting strategies considered in this paper are motivated by a cloud computing setting where data is already initially distributed and the complete space of computing nodes is not necessarily known in advance. In this respect, Hypercube-style and broadcasting algorithms are orthogonal.

Relevance. One approach to minimize data transfer for a query Q, is to find the smallest subset J of a distributed instance I for which $Q(I) = Q(J)$ and then only broadcast the relevant subset J. Determining which part of a database is relevant for answering a query is a problem that arises in different contexts. For instance, causality in databases aims to determine which tuples in the database instance caused the output to a query [16, 17]. Then, the contingency set asks for the smallest set K such that $Q(I) \neq Q(I - K)$. So, any set $I - K$ extended with one element is relevant. Similarly, "where" and "why" provenance refer to the location(s) in the source databases from which the output was extracted or by which the output was influenced [10, 9]. Fan et al. [12] study the problem of scale independence where, through access patterns, the result of a query depends only on a bounded part of the database. It would be interesting to investigate how these different approaches translate to a distributed setting. Most surely, any lower bounds for the sequential setting imply lower bounds for the distributed setting, but upper bounds need to take into account that the initial database instance I is distributed as well.

3 Preliminaries

Instances and queries. For a finite set S, we denote by $|S|$ its cardinality and by 2^S its powerset. We denote $\{1, \ldots, n\}$ by $[n]$, for $n \in \mathbb{N}$. We assume an infinite set **dom** of data values. A *database schema* σ is a collection of relation names R where every R has arity $ar(R) > 0$. We call $R(\bar{d})$ a *fact* when R is a relation name and $\bar{d}$ is a tuple in **dom**. We say that a fact $R(d_1, \ldots, d_k)$ is *over* a database schema σ if $R \in \sigma$ and $ar(R) = k$. A *(database) instance* I over σ is simply a finite set of facts over σ. We denote by $Adom(I)$ the set of all values that occur in facts of I. When $I = \{\mathbf{f}\}$, we simply write $Adom(\mathbf{f})$ rather than $Adom(\{\mathbf{f}\})$. A *query* over a schema σ to a schema σ' is a generic mapping Q from instances over σ to instances over σ'. Genericity means that for every permutation π of **dom** and every instance I, $Q(\pi(I)) = \pi(Q(I))$. For the remainder of the paper, we assume given a database schema σ over which all queries are defined and do not refer to it anymore. A query Q is *monotone* if $Q(I) \subseteq Q(J)$ for all instances I, J with $I \subseteq J$. We only consider monotone queries in the sequel.

Conjunctive queries. Let **var** be the universe of variables, disjoint from **dom**. An *atom* A is of the form $R(u_1, \ldots, u_k)$ where R is a relation name and each $u_i \in$ **var**. We call R the *predicate* and denote it by $pred(A)$. We denote the variables occurring in A by $Vars(A) =$

$\{u_1, \ldots, u_k\}$. We say that A is an atom *over* the database schema σ if $pred(A) \in \sigma$ and $k = ar(pred(A))$. A *conjunctive query* Q (CQ) is an expression of the form $A_0 \leftarrow A_1, \ldots, A_n$, where for every $i \in [n]$, A_i is an atom over the schema and A_0 is an atom not over the schema. In particular, A_0 is the head of Q, denoted $head_Q$, and $A_1, \ldots, A_n$ is the body of Q, denoted $body_Q$. By $Vars(Q)$ we denote all the variables occurring in Q. A *valuation* for Q on an instance I is a function $V : Vars(Q) \to Adom(I)$. The *application* of V to an atom $A = R(u_1, \ldots, u_k)$, denoted $V(A)$, results in the fact $R(a_1, \ldots, a_k)$ where $a_i = V(u_i)$ for each $i \in [k]$. The valuation V is said to be *satisfying* for Q if $V(A) \in I$ for all atoms A in the body of Q. In that case, V derives the fact $V(A_0)$. The result of Q on I, denoted $Q(I)$ is defined as the set of facts that can be derived by satisfying valuations.

In what follows, we assume that every CQ is full and does not contain self-joins. Formally, we require that $pred(A_i) \neq pred(A_j)$ for $i \neq j$ and $Vars(A_0) = \bigcup_{i \in [n]} Vars(A_i)$. That is, every atom has a unique relation symbol and all variables occurring in the body occur in the head as well. For instance, $Q_1(x, y, z) \leftarrow A(x, y), B(x, z), C(y, y)$ is full and does not contain self-joins, while $Q_2(x, y) \leftarrow A(x, y), B(x, z), C(y, y)$ is not full and $Q_3(x, y, z) \leftarrow A(x, y), A(x, z), C(y, y)$ contains a self-join.

Distributed database. A *network* $\mathcal{N}$ is a nonempty finite set of values from **dom**, which we call *nodes*. A *distribution* of an instance I over $\mathcal{N}$ is a function H that maps each $c \in \mathcal{N}$ to an instance such that $I = \bigcup_{c \in \mathcal{N}} H(c)$. Notice that facts can be replicated. We also refer to each of the $H(c)$ as the *local instances*. We consider a model where nodes have unlimited computational power and can send messages to all other nodes. These messages can never be lost but can be arbitrarily delayed.

4 Oblivious broadcasting

We refrain from introducing the formalism of relational transducer networks from [6], but present a simpler setting more suitable for our needs. In particular, the relational transducer networks needed in this paper only perform two actions: decide which facts to broadcast (and transmit those) and evaluate the query under consideration whenever new data arrives. The only parameter is the used broadcasting strategy and, therefore, forms the focus of our formalization. In brief, we consider broadcasting strategies where computing nodes partition their local database into *static* and *broadcast* facts. Static facts are kept local while broadcast facts, as the name already indicates, are sent to all other nodes in the network. As we only consider conjunctive queries which are monotone, the target query can be recomputed whenever new data arrives.

4.1 Oblivious broadcasting functions

We now formally define oblivious broadcasting function.

▶ **Definition 2.** An *oblivious broadcasting function (OBF)* f is a generic mapping that maps instances to instances such that $f(J) \subseteq J$ for all instances J.

An OBF specifies which local facts are broadcast. Specifically, $f(J)$ are the broadcast facts while $J \setminus f(J)$ are the static facts. We use the term oblivious as broadcast facts only depend on the local database instance and their choice is independent of the facts at other computing nodes. An OBF f is *naive* when there are no static facts, that is, $f(J) = J$ for all instances J.

Given a CQ Q, an instance I, a distribution H of I, and a network $\mathcal{N}$, an OBF f implies a broadcasting algorithm in the following way. Let $B(f, H) = \bigcup_{c \in \mathcal{N}} f(H(c))$ be the set of

broadcast facts. Then, define $\mathit{eval}(Q, f, H) = \bigcup_{c \in \mathcal{N}} Q(H(c) \cup B(f, H)))$ as the union of the query result at every computing node over the local instance extended with all broadcast facts.[2]

▶ Remark. We note that the function $\mathit{eval}(Q, f, H)$ implies an evaluation that can be executed by a transducer program $\pi_{f,Q}$ at every node c as follows: (1) $R = \emptyset$, output $Q(H(c))$, broadcast $f(H(c))$; (2) whenever a fact $\mathbf{f}$ arrives, $R = R \cup \{\mathbf{f}\}$, output $Q(H(c) \cup R)$. Correctness then follows from the genericity and monotonicity of f. We refer to the execution strategy induced by $\mathrm{eval}(Q, f, H)$ as a *broadcasting algorithm*. Coordination-freeness intuitively follows as $\pi_{f,Q}$ never waits. Formally, a transducer is coordination-free [6] if there is a so-called *ideal* distribution, on which the query is already computed by a prefix of a run that does not process any of the incoming facts. For $\pi_{f,Q}$ this is the distribution that puts the complete instance at every node. We refer to [6] for a more formal treatment of coordination-freeness.

▶ **Definition 3.** An OBF f is *correct* for a CQ Q when $Q(I) = \mathit{eval}(Q, f, H)$ for all instances I and all distributions H of I.

When f is correct for Q, we also say that f is an OBF *for* Q. The following lemma characterizes correctness in that two compatible facts residing at different computing nodes can never be both static. Indeed, if they are, then the valuation witnessing compatibility is never realized at any computing node and consequently f can not be correct for Q.

We say that two distinct facts $\mathbf{f}$ and $\mathbf{g}$ are *compatible w.r.t* Q, denoted $\mathbf{f} \sim_Q \mathbf{g}$, when they are assigned to two atoms from the body of Q under one valuation, i.e., there is a valuation V for Q and atoms $A, B \in \mathit{body}_Q$, such that $V(A) = \mathbf{f}$ and $V(B) = \mathbf{g}$.

▶ **Lemma 4.** *Let Q be a CQ and f be an OBF. Then, the following are equivalent:*

1. *f is correct for Q; and*
2. *there are no instances I, J, and facts $\boldsymbol{f}, \boldsymbol{g}$, with $\boldsymbol{f} \sim_Q \boldsymbol{g}$, $\boldsymbol{g} \notin I, \boldsymbol{f} \notin J$ such that $\boldsymbol{f} \notin f(I \cup \{\boldsymbol{f}\})$ and $\boldsymbol{g} \notin f(J \cup \{\boldsymbol{g}\})$.*

Proof. (1)⇒(2) We start by showing that every OBF for Q satisfies the above condition. The proof is by contraposition, so we assume that there are instances I and J and compatible facts $\mathbf{f}$ and $\mathbf{g}$ w.r.t. Q, where $\mathbf{g} \notin I$ and $\mathbf{f} \notin J$, but $\mathbf{f} \notin f(I \cup \{\mathbf{f}\})$ and $\mathbf{g} \notin f(J \cup \{\mathbf{g}\})$. Let K be an instance and let V be a satisfying valuation for Q on K witnessing compatibility of $\mathbf{f}$ and $\mathbf{g}$. Then consider a network $\mathcal{N} = \{1, 2, 3\}$ and an instance $L = I \cup J \cup V(\mathit{body}_Q)$ with the following distribution H: $H(1) = I \cup \{\mathbf{f}\}$, $H(2) = J \cup \{\mathbf{g}\}$, and $H(3) = V(\mathit{body}_Q) \setminus \{\mathbf{f}, \mathbf{g}\}$. Clearly, $V(\mathit{head}_Q) \in Q(L)$. As Q is full, $V(\mathit{head}_Q) \notin \bigcup_{c \in \mathcal{N}} Q(H(c) \cup B(f, H))$ because none of the computing nodes contain both $\mathbf{f}$ and $\mathbf{g}$, and $\mathbf{f}$ and $\mathbf{g}$ are not broadcast. Thus, $Q(I) \neq \bigcup_{x \in \mathcal{N}} Q(H(x) \cup B(f, H)) = \mathit{eval}(Q, f, H)$ and f is not an OBF for Q.

(2)⇒(1) It remains to show that if the above condition is satisfied, then f is an OBF for Q. For this, let I be an instance, $\mathcal{N}$ a network, and H a distribution of I over $\mathcal{N}$. We prove that $Q(I) = \mathit{eval}(Q, f, H) = \bigcup_{c \in \mathcal{N}} Q(H(c) \cup B(f, H))$. As Q is monotone, $Q(H(c) \cup B(f, H)) \subseteq Q(I)$ for every $c \in \mathcal{N}$. Hence, it suffices to show that $Q(I) \subseteq \bigcup_{c \in \mathcal{N}} Q(H(c) \cup B(f, H))$. Thereto, let $\mathbf{f} \in Q(I)$, let V be a satisfying valuation for Q over I for which $V(\mathit{head}_Q) = \mathbf{f}$. Let $J = V(\mathit{body}_Q) \setminus B(f, H)$, and c a node for which $|H(c) \cap J|$ is maximal. We claim that $J \subseteq H(c)$, obviously implying that $\mathbf{f}$ will be derived at node c. Towards a contradiction, assume there is an $\mathbf{f}_i \in J \setminus H(c)$. As $\mathbf{f}_i \in I$ there is a $d \in \mathcal{N}$, $c \neq d$, such that $\mathbf{f}_i \in H(d)$. Moreover, by choice of c, $|H(d) \cap J| \leq |H(c) \cap J|$ and thus there must be a fact $\mathbf{f}_j \in H(c)$

[2] To simplify notation, in the definition of B and *eval*, we do not mention I and $\mathcal{N}$ as they are implied by H.

that is not in $H(d)$. However, as $\mathbf{f}_i \sim_Q \mathbf{f}_j$, $\mathbf{f}_i \notin H(c)$, and $\mathbf{f}_j \notin H(d)$, instances $H(d)$, $H(c)$, and facts $\mathbf{f}_i, \mathbf{f}_j$ contradict condition (2). ◀

4.2 Local optimality

We are interested in OBFs that transmit as little data as possible. Thereto, we investigate sensible notions of optimality. We fix a query Q, an instance I, a distribution H of I, and a network $\mathcal{N}$. The total number of transmitted facts equals $||B(f,H)|| = \sum_{c\in\mathcal{N}} |f(H(c))|$. Of course, $||B(f,H)|| \geq |B(f,H)|$.

▶ **Definition 5.** An OBF f for a CQ Q is *optimal* iff $||B(f,H)|| \leq ||B(g,H)||$ for every other OBF g for Q and for every instance I and distribution H.

Intuitively, an OBF is optimal when it transmits the least amount of data over all instances and all distributions. The next result, however, shows that this notion of optimality, although desirable, is unattainable.

▶ **Lemma 6.** *There is no optimal OBF for any conjunctive query with at least two distinct atoms in its body.*

Proof. Let Q be the conjunctive query $A_0 \leftarrow A_1, \ldots, A_n$ with $n \geq 2$. Towards a contradiction assume there is an optimal OBF f for Q. Let I be the canonical instance for Q where for every $i \in [n]$, the relation $pred(A_i)$ is interpreted by the fact A_i.[3] Now, consider a network $\mathcal{N} = [n]$ and a distribution H that places every fact in I on a distinct node. As all of the n facts in I need to be gathered at one node, at least $n-1$ facts must be broadcast. Let $\mathbf{g}$ be the fact in I that is not broadcast by f and assume w.l.o.g. that $pred(\mathbf{g}) = A_n$. As the OBF that broadcasts all A_i-facts for $i < n$ and keeps all A_n-facts static is correct for Q and only transmits $n-1$ facts on I, by assumption on the minimality of f, $||B(f,H)|| = n-1$. Now, consider $I' = I \setminus \{\mathbf{g}\}$. And let H' equal H restricted to only facts in I' over $\mathcal{N}$. Then, as $\mathbf{g}$ is not broadcast in H, $||B(f,H)|| = ||B(f,H')||$. However, the OBF that broadcasts all A_i-facts for $i > 1$ and keeps all A_1-facts static is correct for Q and only broadcasts $n-2$ facts on I' contradicting the optimality of f. ◀

We next turn to a different form of optimality. For two OBFs f and g, we say that f is *included* in g, denoted $f \subseteq g$, iff $f(I) \subseteq g(I)$ for every instance I.

▶ **Definition 7.** An OBF f for a CQ Q is *local optimal* iff for every other broadcasting function g for Q, $g \subseteq f$ implies $f = g$.

Intuitively, when f is local optimal there is no subdivision of f that transmits only a strict subset of the facts broadcast by f.

The next lemma gives a sufficient criteria for when an OBF can not be local optimal. Specifically, a condition is given for when a broadcast fact $\mathbf{f}$ can be kept static and a more economical OBF f' can be derived.

▶ **Lemma 8.** *Let Q be a CQ and let f be an OBF for Q. If there is an instance I and fact $\boldsymbol{f}$ for which $\boldsymbol{f} \in f(I \cup \{\boldsymbol{f}\})$, but there is no instance J and no fact $\boldsymbol{g}$ for which $\boldsymbol{f} \sim_Q \boldsymbol{g}$, $\boldsymbol{g} \notin I$, $\boldsymbol{f} \notin J$, and $\boldsymbol{g} \notin f(J \cup \{\boldsymbol{g}\})$, then there is an OBF f' for Q for which $f' \subsetneq f$.*

[3] Notice that we abuse the notation and interpret variables as values.

Proof. Assume f, I, and $\mathbf{f}$ as given by the statement of the lemma. The proof is now by construction. Let $I_{\mathbf{f},J}$ be the set of facts that (by genericity) relate the same way to J, as $\mathbf{f}$ to I. That is, $I_{\mathbf{f},J} = \{\pi(\mathbf{f}) \mid \pi \text{ a permutation s.t. } \pi(I) = J\}$. Then, define f' as the mapping where for every instance J, $f'(J) = f(J) \setminus I_{\mathbf{f},J}$. Notice that $f' \subsetneq f$ by construction of f'. Furthermore, f' is generic and is an oblivious broadcasting function. It remains to show that f' is an oblivious broadcasting function for Q. Towards a contradiction, assume that f' is not an oblivious broadcasting function for Q. Then, by Lemma 4, there are instances J_1 and J_2 and facts $\mathbf{g}_1$ and $\mathbf{g}_2$, for which $\mathbf{g}_1 \sim_Q \mathbf{g}_2, \mathbf{g}_2 \notin J_1$, $\mathbf{g}_1 \notin J_2$, and $\mathbf{g}_1 \notin f'(J_1 \cup \{\mathbf{g}_1\})$ and $\mathbf{g}_2 \notin f'(J_2 \cup \{\mathbf{g}_2\})$. As f is an oblivious broadcasting function for Q, it holds that

$$\mathbf{g}_1 \in f(J_1 \cup \{\mathbf{g}_1\}) \text{ or } \mathbf{g}_2 \in f(J_2 \cup \{\mathbf{g}_2\}).$$

Say that $\mathbf{g}_1 \in f(J_1 \cup \{\mathbf{g}_1\})$. Then, $\mathbf{g}_1 \in I_{\mathbf{f},J_1}$, implying $J_1 = \pi(I)$ and $\mathbf{g}_1 = \pi(\mathbf{f})$ for some permutation π. As Q does not contain self-joins and $\mathbf{g}_1 \sim_Q \mathbf{g}_2$, this means that $\mathbf{g}_2 \notin I_{\mathbf{f},J}$. Therefore, $\mathbf{g}_2 \notin f(J_2 \cup \{\mathbf{g}_2\})$ which contradicts the condition of the lemma (taking $\pi^{-1}(\mathbf{g}_1)$ and $\pi^{-1}(J_2)$ as $\mathbf{g}$ and J, respectively). ◀

The following lemma now characterizes when an OBF for a query is local optimal.

▶ **Lemma 9.** *Let Q be a CQ and let f be an OBF for Q. The following are equivalent:*

1. *f is local optimal; and*
2. *for every instance I and fact $\boldsymbol{f}$ for which $\boldsymbol{f} \in f(I \cup \{\boldsymbol{f}\})$, there is an instance J and a fact $\boldsymbol{g}$ such that $\boldsymbol{f} \sim_Q \boldsymbol{g}$, $\boldsymbol{g} \notin I$, $\boldsymbol{f} \notin J$, and $\boldsymbol{g} \notin f(J \cup \{\boldsymbol{g}\})$.*

Proof. We can assume that Q contains at least two atoms. Indeed, when Q contains one atom, the only local optimal OBF is the one that broadcasts no facts and the lemma trivially holds. The direction from (1) to (2) follows from Lemma 8.

(2)⇒(1) Let f be an OBF for Q. Towards a contradiction assume that f is not local optimal. That is, there exists another OBF f' for Q such that $f' \subsetneq f$. In particular, there is an instance I and a fact $\mathbf{f}$ such that $\mathbf{f} \notin f'(I \cup \{\mathbf{f}\})$, while $\mathbf{f} \in f(I \cup \{\mathbf{f}\})$. By Lemma 4, for every fact $\mathbf{g}$ with $\mathbf{f} \sim_Q \mathbf{g}$ where $\mathbf{g} \notin I$, and for every instance J, where $\mathbf{f} \notin J$, it must be that $\mathbf{g} \in f'(J \cup \{\mathbf{g}\})$. The latter then implies that for every such $\mathbf{g}$ and J, $\mathbf{g} \in f(J \cup \{\mathbf{g}\})$ which contradicts condition (2) of the present lemma. ◀

5 Broadcasting functions based on dependency sets

In this section, we introduce the notion of a broadcast dependency set (BDS) as a formalism to specify OBFs. We present necessary and sufficient conditions for when a BDS induces an OBF which is correct for a given query and also for when it is local optimal. Furthermore, we study the complexity of the corresponding decision problems. Finally, we show that every local optimal OBF can be represented by a BDS thereby obtaining that BDS is complete as a representation formalism for local optimal OBFs.

5.1 Broadcast dependency sets

Let Q be the CQ $A_0 \leftarrow A_1, \ldots, A_n$. We assume Q is full and does not contain self-joins. Therefore an atom A_i in $body_Q$ is uniquely identified by its predicate $pred(A_i)$. For a predicate R, we denote by $atom(R)$ the unique atom $A \in body_Q$ for which $pred(A) = R$.

For a finite set of variables X, a *partial (equality) type over* X is a pair of binary relations $\varphi = (E_\varphi, I_\varphi)$ representing equalities and inequalities among elements in X. Formally, we require that $E_\varphi \cup I_\varphi \subseteq X \times X$, E_φ is an equivalence relation, and I_φ is irreflexive and

symmetric. We abuse notation and also use φ to denote the formula $\bigwedge\{x = y \mid (x, y) \in E_\varphi\} \wedge \bigwedge\{x \neq y \mid (x, y) \in I_\varphi\}$. We tacitly assume that partial types are always consistent. That is, we always assume that there is a tuple $\bar{a}$ such that the formula $\varphi(\bar{a})$ evaluates to true. When for all $(x, y) \in X \times X$, either $(x, y) \in E_\varphi$ or $(x, y) \in I_\varphi$, then φ completely specifies all relations between variables in X and we call φ a *type*. For emphasis, we sometimes say *complete type* rather than just *type* even though type always means complete type.

A *partial atomic type* (over Q) is a pair $\tau = (R_\tau, \varphi_\tau)$, where R_τ is a database predicate and φ_τ is a partial type over $\mathit{Vars}(\mathit{atom}(R_\tau))$, that is, the variables occurring in the unique atom $A \in \mathit{body}_Q$ for which $\mathit{pred}(A) = R_\tau$. By $\mathit{Vars}(\tau)$ we denote the variables over which τ is defined, i.e., $\mathit{Vars}(\tau) = \mathit{Vars}(\mathit{atom}(R_\tau))$. Sometimes we write $\mathit{atom}(\tau)$ to abbreviate $\mathit{atom}(R_\tau)$. We say that τ is an *atomic type* when φ_τ is a type. To improve readability, we denote partial atomic types with τ and (complete) atomic types with ω. We denote by $\mathit{PTypes}(Q)$ and $\mathit{Types}(Q)$ the set of all partial atomic types and atomic types over Q, respectively.

A fact $\mathbf{f}$ *is of type* τ or *satisfies* τ, denoted $\mathbf{f} \models \tau$, when there is a valuation h from the variables in $\mathit{atom}(R_\tau)$ onto $\mathit{Adom}(\mathbf{f})$ such that $h(\mathit{atom}(R_\tau)) = \mathbf{f}$ and the formula φ_τ evaluates to true where each x_i is substituted by $h(x_i)$. Notice that h is unique for $\mathbf{f}$. Hereafter we will refer to h as $V_\mathbf{f}$. By $\mathit{type}(\mathbf{f})$, we denote the unique atomic type satisfied by $\mathbf{f}$ when it exists. As atomic types are defined w.r.t. Q, $\mathit{type}(\mathbf{f})$ is not always defined. Indeed, when $\mathbf{f} = R(a, b)$ (with $a \neq b$) and $\mathit{atom}(R) = R(x, x)$, then there is no τ with $\mathbf{f} \models \tau$. Two partial atomic types τ, τ' are *compatible w.r.t.* Q, denoted $\tau \sim_Q \tau'$, when there are facts $\mathbf{f}$ and $\mathbf{g}$ with $\mathbf{f} \models \tau$ and $\mathbf{g} \models \tau'$ such that $\mathbf{f} \sim_Q \mathbf{g}$. We say that τ *implies* τ', denoted $\tau \models \tau'$, if for all facts $\mathbf{f}$, $\mathbf{f} \models \tau$ implies $\mathbf{f} \models \tau'$. We can think of a partial atomic type as a disjunction of types for a shared predicate symbol. Define $\mathit{Types}(\tau) = \{\omega \in \mathit{Types}(Q) \mid \omega \models \tau\}$ as the set of all atomic types ω which imply τ. Notice that, $\omega \models \tau$ iff $\omega \in \mathit{Types}(\tau)$ for any atomic type ω. For a set of partial atomic types T, we use $\mathit{Types}(T)$ as an abbreviation for $\bigcup_{\tau \in T} \mathit{Types}(\tau)$.

For a set of variables X and Y, and a partial atomic type τ, $X \subseteq_\tau Y$ if for all $x \in X$ either $x \in Y$ or there is an $y \in Y$ such that $(x, y) \in E_{\varphi_\tau}$. That is, X is a subset of Y when taking the equalities in E_{φ_τ} into account. For instance, let τ be a type such that $(y, z) \in E_{\varphi_\tau}$, then $\{x, y, z\} \subseteq_\tau \{x, y\}$.

For a set of pairs $\mathcal{S}$, we define $\mathit{Keys}(\mathcal{S}) = \{a \mid (a, b) \in \mathcal{S}\}$ and $\mathit{Values}(\mathcal{S}) = \{b \mid (a, b) \in \mathcal{S}\}$.

▶ **Definition 10.** A *broadcast dependency set (BDS)* for a CQ Q is a set $\mathcal{S}$ of pairs (τ, T), where $\tau \in \mathit{PTypes}(Q)$ is a *key*, and $T \in 2^{\mathit{PTypes}(Q)}$ is a *dependency set*, such that the following holds:

1. $(\tau, T) \in \mathcal{S}$ and $(\tau, T') \in \mathcal{S}$ implies $T = T'$;
2. $\tau, \tau' \in \mathit{Keys}(\mathcal{S})$ implies $\mathit{Types}(\tau) \cap \mathit{Types}(\tau') = \emptyset$; and,
3. $(\tau, T) \in \mathcal{S}$ implies $\mathit{Vars}(\tau') \subseteq_{\tau'} \mathit{Vars}(\tau)$ for every $\tau' \in T$.

The above definition states that (1) every key can have at most one value in $\mathcal{S}$; (2) every complete type implies at most one partial type $\tau \in \mathit{Keys}(\mathcal{S})$; and, (3) the set of variables of $\mathit{atom}(\tau')$ is included in the set of variables of $\mathit{atom}(\tau)$ taking into account the equalities in $E_{\tau'}$. We first explain informally how a BDS represents an OBF. Let $\mathbf{f}$ be a fact in the local instance at a computing node. When $\mathit{type}(\mathbf{f})$ is undefined, then $\mathbf{f}$ is static as $\mathbf{f}$ can never participate in any satisfying valuation. For instance this happens when $\mathbf{f} = R(a, b)$ with $a \neq b$ and Q contains the atom $R(x, x)$. Every pair $(\tau, T) \in \mathcal{S}$ now specifies a condition on facts: when $\mathbf{f} \models \tau$ then $\mathbf{f}$ is broadcast only if a set of facts implied by T (to be formalized below) is not present at the local instance. Furthermore, when there is no $\tau \in \mathit{Keys}(\mathcal{S})$ for which $\mathbf{f} \models \tau$, $\mathbf{f}$ is broadcast as well. In this light, conditions (1) and (2) ensure that every

local fact $\mathbf{f}$ is matched by at most one partial type $\tau \in \mathit{Keys}(S)$; and, condition (3) ensures that when $\mathbf{f} \models \tau$ then $V_{\mathbf{f}}$ can be extended in a unique way to a valuation for every $\tau' \in T$ that is consistent with $\mathbf{f}$, that is, for which $\mathit{type}(\mathbf{f}) \sim_Q \tau'$.

Next, we formally define how every BDS $\mathcal{S}$ implies an OBF $f_{\mathcal{S}}$. Given a fact $\mathbf{f}$, if there is no $\tau \in \mathit{Keys}(\mathcal{S})$ for which $\mathbf{f} \models \tau$ then $\mathbf{f}$ is always broadcast. Otherwise, by condition (1) and (2) of Definition 10, there is exactly one $\tau \in \mathit{Keys}(\mathcal{S})$ such that $\mathbf{f} \models \tau$. Recall that $V_{\mathbf{f}}$ is the valuation (defined above) such that $V_{\mathbf{f}}(\mathit{atom}(\tau)) = \mathbf{f}$. Then, by condition (3) of Definition 10, $V_{\mathbf{f}}$ can also be interpreted as a valuation for every $\mathit{atom}(\tau')$ for every $\tau' \in T$ for which $\mathit{type}(\mathbf{f}) \sim_Q \tau'$. Indeed, for every $y \in \mathit{Vars}(\tau') \setminus \mathit{Vars}(\tau)$ there is a variable $x \in \mathit{Vars}(\tau)$ for which $(x, y) \in E_{\tau'}$. Therefore, define for every $y \in \mathit{Vars}(\tau')$,

$$V_{\mathbf{f},\tau'}(y) = \begin{cases} V_{\mathbf{f}}(y) & \text{if } y \in \mathit{Vars}(\tau); \text{ and,} \\ V_{\mathbf{f}}(x) & \text{if } y \notin \mathit{Vars}(\tau) \text{ and } (x, y) \in E_{\tau'}. \end{cases}$$

As we only consider $V_{\mathbf{f},\tau'}$ for which $\mathit{type}(\mathbf{f}) \sim_Q \tau'$, the above is well-defined.

Now, $\mathbf{f}$ is broadcast when the local instance does not contain all the facts $V_{\mathbf{f},\tau'}(\mathit{atom}(\tau'))$ for which $\tau' \in T$ and $\mathit{type}(\mathbf{f}) \sim_Q \tau'$. We refer to these facts as the *dependency fact set*. To formally define $f_{\mathcal{S}}$, we set $\mathit{Dep}(\mathbf{f}, T) = \{V_{\mathbf{f},\tau'}(\mathit{atom}(\tau')) \mid \tau' \in T \text{ and } \mathit{type}(\mathbf{f}) \sim_Q \tau'\}$. Then, define $\mathit{Dep}(\mathbf{f}, \mathcal{S})$ as $\mathit{Dep}(\mathbf{f}, T)$ when there is a $(\tau, T) \in \mathcal{S}$ for which $\mathbf{f} \models \tau$. Otherwise, $\mathit{Dep}(\mathbf{f}, \mathcal{S})$ is undefined.

▶ **Definition 11.** For a CQ Q and a BDS $\mathcal{S}$ for Q, define $f_{\mathcal{S}}$ as the function that maps every instance J to the set $f_{\mathcal{S}}(J)$ of those facts $\mathbf{f} \in J$ for which (1) $\mathit{type}(\mathbf{f}) \in \mathit{Types}(Q)$; and, (2) $\mathit{Dep}(\mathbf{f}, \mathcal{S})$ is undefined or $\mathit{Dep}(\mathbf{f}, \mathcal{S}) \not\subseteq J$.

Intuitively, $\mathbf{f}$ is static only when $\mathit{type}(\mathbf{f}) \notin \mathit{Types}(Q)$ ($\mathbf{f}$ can not participate in any satisfying valuation) or the dependency fact set $\mathit{Dep}(\mathbf{f}, \mathcal{S})$ is present at the local instance.

▶ **Example 12.** (1) For a simple example of a BDS $\mathcal{S}$ and OBF $f_{\mathcal{S}}$, recall query Q_1 from Example 1, being $Q_1(x, y, z) \leftarrow A(x, y), B(y, x), C(x, z)$. Let $\varphi = (\emptyset, \emptyset)$, that is, φ imposes no restrictions. Let $\tau_A = (A, \varphi)$ and $\tau_B = (B, \varphi)$. Then, $\mathcal{S} = \{(\tau_B, \{\tau_A\}), (\tau_A, \emptyset)\}$ is a BDS for Q_1. Indeed, every partial atomic type occurs at most once as a key. There is no (complete) atomic type that implies both τ_A and τ_B. Furthermore, the variable containment condition between τ_A and τ_B is satisfied. Notice that $f_{\mathcal{S}}$ simulates exactly the broadcast dependency function which is described in Example 1.

(2) Consider the query $Q_2(x, y, z) \leftarrow A(x, y, z), B(x, y, z), C(z, z)$. For simplicity, we define partial types through formulas. Then, define $\tau_B = (B, \text{true})$, $\tau_A^{x=y} = (A, x = y)$, $\tau_A^{y=z} = (A, y = z)$, $\tau_A^{\neq} = (A, x \neq y \wedge y \neq z)$, $\tau_B^{\neq} = (B, x \neq y \wedge y \neq z)$. Then, $\mathcal{S} = \{(\tau_B, \{\tau_A^{x=y}, \tau_A^{y=z}\}), (\tau_A^{\neq}, \{\tau_B^{\neq}\})\}$ is a BDS for Q_2. To illustrate how OBF $f_{\mathcal{S}}$ works, let $I = \{A(1,2,3), B(1,2,3), A(1,1,2), B(1,1,2), A(1,2,2), B(1,2,2), C(3,4), C(3,3)\}$ be a database instance. Then, $f_{\mathcal{S}}(I) = \{A(1,1,2), A(1,2,2), C(3,3)\}$. Indeed, the facts $A(1,1,2)$, $A(1,2,2)$, $C(3,3)$ do not match a key in $\mathcal{S}$ and their type occurs in $\mathit{Types}(Q)$. So they are broadcast. The fact $C(3,4)$ is not broadcast as its type does not occur in $\mathit{Types}(Q)$ ($C(3,4)$ does not match $C(z,z)$). The fact $\mathbf{f}_1 = B(1,1,2)$ matches τ_B and $\mathit{Dep}(\mathbf{f}_1, \{\tau_A^{x=y}, \tau_A^{y=z}\}) = \{A(1,1,2)\} \subseteq I$. Therefore, $B(1,1,2)$ is static. Similarly, the fact $\mathbf{f}_2 = B(1,2,2)$ matches τ_B and $\mathit{Dep}(f_2, \{\tau_A^{x=y}, \tau_A^{y=z}\}) = \{A(1,2,2)\} \subseteq I$. Therefore, $B(1,2,2)$ is static as well. The fact $\mathbf{f}_3 = A(1,2,3)$ is static as it matches τ_A and $\mathit{Dep}(\mathbf{f}_3, \{\tau_b^{\neq}\}) = \{B(1,2,3)\} \subseteq I$. The fact $\mathbf{f}_4 = B(1,2,3)$ is static as it matches τ_B and $\mathit{Dep}(\mathbf{f}_4, \{\tau_A^{x=y}, \tau_A^{y=z}\}) = \emptyset$.

(3) For an example where condition (3) of Definition 10 does not reduce to ordinary variable containment, consider again query Q_1 from Example 1. Let $\tau_C = (C, x = z)$, and

$\tau_A = (A, \text{true})$. Then, $\mathcal{S} = \{(\tau_A, \{\tau_C\}), (\tau_C, \emptyset)\}$ is a BDS for Q_1. Notice that condition $Vars(C) \not\subseteq Vars(A)$ but $Vars(\tau_C) \subseteq_{\tau_C} Vars(\tau_A)$.

(4) Our final example shows that dependencies can be circular. Let $Q_3(x, y, z) \leftarrow A(x, y), B(y, z), C(z, x)$. Let $\tau_A = (A, x = y)$, $\tau_B = (B, x = y)$, and $\tau_C = (C, x = y)$. Then, $\mathcal{S} = \{(\tau_A, \{\tau_B\}), (\tau_B, \{\tau_C\}), (\tau_C, \{\tau_A\})\}$ is an OBF for Q_1. Though correctness of $\mathcal{S}$ for Q follows from Lemma 13, we provide some intuition. Let $I = \{A(1,1), B(1,1), C(1,1)\}$ be a database instance. Consider a network containg the nodes c_1, c_2, and c_3. When $I(c_1) = \{A(1,1)\}$, $I(c_2) = \{B(1,1)\}$, and $I(c_3) = \{C(1,1)\}$, then all three facts will be broadcast. Now, assume one of the nodes contains two of the facts in I, w.l.o.g., say $I(c_1) = \{A(1,1), B(1,1)\}$. Then, exactly one of the facts in $I(c_1)$ is broadcast; i.e., $B(1,1)$. Now, suppose that $C(1,1)$ is mapped on some node, say c_2, but that $C(1,1)$ is not broadcast. Then it must be that $A(1,1)$ is mapped on c_2 as well. So, broadcasting $B(1,1)$ indeed suffices to guarantee correctness. ◀

Note that not every BDS for Q induces an OBF which is correct for Q. Indeed, the following lemma provides equivalent semantic and syntactic conditions for an OBF $f_{\mathcal{S}}$ to be correct for a query.

▶ **Lemma 13.** *Let Q be a CQ and let $\mathcal{S}$ be a BDS for Q. Then the following are equivalent:*

1. *$f_{\mathcal{S}}$ is an OBF for Q;*
2. *there are no instances I, J, and facts $\boldsymbol{f}, \boldsymbol{g}$, with $\boldsymbol{f} \sim_Q \boldsymbol{g}$, $\boldsymbol{g} \notin I, \boldsymbol{f} \notin J$ such that $\boldsymbol{f} \notin f_{\mathcal{S}}(I \cup \{\boldsymbol{f}\})$ and $\boldsymbol{g} \notin f_{\mathcal{S}}(J \cup \{\boldsymbol{g}\})$; and*
3. *there are no (complete) atomic types ω_1, ω_2, and pairs $(\tau_1, T_1), (\tau_2, T_2) \in \mathcal{S}$, with $\omega_1 \sim_Q \omega_2$, $\omega_1 \models \tau_1$, $\omega_2 \models \tau_2$ such that $\omega_1 \notin Types(T_2)$ and $\omega_2 \notin Types(T_1)$.*

Proof. (1)⇔(2) Because $f_{\mathcal{S}}$ is an OBF, the equivalence follows immediately from Lemma 4.

(2)⇒(3) The proof is by contraposition. So, assume that there are two (complete) atomic types ω_1, ω_2, and pairs $(\tau_1, T_1), (\tau_2, T_2) \in \mathcal{S}$, with $\omega_1 \sim_Q \omega_2$, $\omega_1 \in Types(\tau_1)$, $\omega_2 \in Types(\tau_2)$ such that $\omega_1 \notin Types(T_2)$ and $\omega_2 \notin Types(T_1)$. Now, because $\omega_1 \sim_Q \omega_2$, there are facts $\mathbf{f}$ and $\mathbf{g}$, with $\mathbf{f} \sim_Q \mathbf{g}$, $type(\mathbf{f}) = \omega_1$, and $type(\mathbf{g}) = \omega_2$. Define $I = Dep(\mathbf{f}, \mathcal{S})$ and $J = Dep(\mathbf{g}, \mathcal{S})$. Observe that by definition of Dep, $\omega_1 \notin Types(T_2)$ implies $\mathbf{f} \notin Dep(\mathbf{g}, \mathcal{S})$ and $\omega_2 \notin Types(T_1)$ implies $\mathbf{g} \notin Dep(\mathbf{f}, \mathcal{S})$. Hence, $\mathbf{f} \notin J$ and $\mathbf{g} \notin I$. Moreover, by definition of $f_{\mathcal{S}}$, it is always the case that $\mathbf{f} \notin f_{\mathcal{S}}(Dep(\mathbf{f}, \mathcal{S}) \cup \{\mathbf{f}\})$ and $\mathbf{g} \notin f_{\mathcal{S}}(Dep(\mathbf{g}, \mathcal{S}) \cup \{\mathbf{g}\})$. Therefore, $\mathbf{f} \notin f_{\mathcal{S}}(I \cup \{\mathbf{f}\})$ and $\mathbf{g} \notin f_{\mathcal{S}}(J \cup \{\mathbf{g}\})$, which contradicts condition (2).

(3)⇒(2) Again, the proof is by contraposition. So, assume that there is an instance I and J and facts $\mathbf{f}$ and $\mathbf{g}$ where $\mathbf{f} \sim_Q \mathbf{g}$, $\mathbf{g} \notin I$ and $\mathbf{f} \notin J$, but $\mathbf{f} \notin f_{\mathcal{S}}(I \cup \{\mathbf{f}\})$ and $\mathbf{g} \notin f_{\mathcal{S}}(J \cup \{\mathbf{g}\})$. As $\mathbf{f} \sim_Q \mathbf{g}$, we have $\omega_1 \sim_Q \omega_2$ for $\omega_1 = type(\mathbf{f})$ and $\omega_2 = type(\mathbf{g})$. Then, by construction of $f_{\mathcal{S}}$ there are $(\tau_1, T_1), (\tau_2, T_2) \in \mathcal{S}$ with $type(\mathbf{f}) \in Types(\tau_1)$ and $type(\mathbf{g}) \in Types(\tau_2)$. Now, $\mathbf{f} \notin f_{\mathcal{S}}(I \cup \{\mathbf{f}\})$ and $\mathbf{g} \notin f_{\mathcal{S}}(J \cup \{\mathbf{g}\})$ implies $Dep(\mathbf{f}, \mathcal{S}) \subseteq I$ and $Dep(\mathbf{g}, \mathcal{S}) \subseteq J$. If we assume that $type(\mathbf{g}) \in Types(T_1)$ then $\mathbf{g} \in Dep(\mathbf{f}, \mathcal{S})$ (as $\mathbf{g} = V_{\mathbf{f}, type(\mathbf{g})}(atom(type(\mathbf{f})))$), and therefore $\mathbf{g} \in I$ which leads to a contradiction. Hence, $type(\mathbf{g}) \notin Types(T_1)$. A similar argument shows that $type(\mathbf{f}) \notin Types(T_2)$. So, we have found ω_1, ω_2, (τ_1, T_2), and (τ_2, T_2) contradicting condition (3). ◀

Notice that the OBFs of Example 12 are all correct for the given query.

Two partial atomic types τ_1, τ_2 are said to be *equal*, denoted $\tau_1 = \tau_2$, when $Types(\tau_1) = Types(\tau_2)$. We say that a BDS $\mathcal{S}$ is *harmonious* when every two partial types in $\mathcal{S}$ are either disjoint or equal. That is, when for every two partial atomic types $\tau_1, \tau_2 \in Keys(\mathcal{S}) \cup \{\tau' \in T \mid T \in Values(\mathcal{S})\}$, either $\tau_1 = \tau_2$ or $Types(\tau_1) \cap Types(\tau_2) = \emptyset$.

▶ **Theorem 14.** *Let Q be a CQ and let $\mathcal{S}$ be a BDS for Q. Deciding whether $f_{\mathcal{S}}$ is correct for Q is* CONP*-complete and in* PTIME *when $\mathcal{S}$ is harmonious.*

5.2 Local optimality

Next, we turn to local optimal OBFs. The following lemma provides equivalent semantic and syntactic conditions for an OBF to be local optimal. Regarding condition (3), the intuition is as follows. While condition (3c) is the syntactic counterpart of condition (2), conditions (3a) and (3b) specify optimality requirements which are inherent to the formalism of BDS. More specifically, condition (3a) specifies that every atomic type implying a partial type in a dependency set in $\mathcal{S}$ must also imply a key in $\mathcal{S}$. Indeed, when an atomic type does not imply a key, every local fact of this type is always broadcast and therefore present at every computing node. The atomic type can therefore be removed from every dependency set it occurs in. When Condition (3b) fails for an atomic type ω, $\mathcal{S}$ can be adapted to broadcast less while preserving correctness for Q by adding the pair $(\omega, \{\tau \mid \tau \sim_Q \omega, \tau \in \mathit{Types}(\mathit{Keys}(\mathcal{S}))\})$.

▶ **Lemma 15.** *Let Q be a CQ, $\mathcal{S}$ a BDS for Q, and $f_{\mathcal{S}}$ an OBF for Q. The following are equivalent:*

1. *$f_{\mathcal{S}}$ is local optimal;*
2. *for every instance I and fact $\boldsymbol{f}$ for which $\boldsymbol{f} \in f_{\mathcal{S}}(I \cup \{\boldsymbol{f}\})$, there is an instance J and a fact $\boldsymbol{g}$ such that $\boldsymbol{f} \sim_Q \boldsymbol{g}$, $\boldsymbol{g} \notin I$, $\boldsymbol{f} \notin J$, and $\boldsymbol{g} \notin f_{\mathcal{S}}(J \cup \{\boldsymbol{g}\})$; and,*
3. *$\mathcal{S}$ satisfies the following conditions:*
   - **(a)** *for $(\tau, T) \in \mathcal{S}$ and $\omega \in \mathit{Types}(T)$, $\omega \sim_Q \tau$ implies $\omega \models \tau'$ for some $\tau' \in \mathit{Keys}(\mathcal{S})$;*
   - **(b)** *for every $\omega \in \mathit{Types}(Q) \setminus \mathit{Types}(\mathit{Keys}(\mathcal{S}))$, there is a partial atomic type $\tau_1 \in \mathit{Keys}(\mathcal{S})$ and a $\omega_1 \in \mathit{Types}(\tau_1)$ such that $\omega \sim_Q \omega_1$ and $\mathit{Vars}(\omega_1) \not\subseteq_{\omega_1} \mathit{Vars}(\omega)$; and*
   - **(c)** *for $(\tau_1, T_1), (\tau_2, T_2) \in \mathcal{S}$, where $\omega_1 \in \mathit{Types}(\tau_1)$, $\omega_2 \in \mathit{Types}(\tau_2)$, and $\omega_1 \sim_Q \omega_2$: $\omega_1 \in \mathit{Types}(T_2)$ implies $\omega_2 \notin \mathit{Types}(T_1)$.*

Deciding whether $f_{\mathcal{S}}$ is local optimal for arbitrarily given BDS $\mathcal{S}$ turns out to be hard (c.f., Theorem 16). Therefore, we also consider the special case of open BDSs. We say that a partial type $\varphi = (E, I)$ is *open* when it enforces no restrictions. That is, when $E = I = \emptyset$. A partial atomic type (R, φ) is *open* when φ is. We say that a BDS $\mathcal{S}$ is *open* when it only contains open partial atomic types. Notice that a BDS that is open is also harmonious (but not vice versa).

Similarly to Theorem 14, we have the following decidability result for local optimal OBFs.

▶ **Theorem 16.** *Let Q be a CQ and let $\mathcal{S}$ be a BDS for Q for which $f_{\mathcal{S}}$ is correct for Q. Deciding whether $f_{\mathcal{S}}$ is local optimal is in* CONP *and in* PTIME *when $\mathcal{S}$ is open.*

It remains open though whether deciding local optimality is CONP-complete or in PTIME (even for harmonious BDS). For harmonious BDS, condition (1) and (3) of Lemma 15 are verifiable in polynomial time.

Next, we show that every local optimal OBF can be represented by a BDS thereby obtaining that BDSs (satisfying the conditions in Lemma 15) are a complete representation of local optimal OBFs. Let Q be a CQ and let f be an OBF for Q. We call a fact $\mathbf{f}$ *semi-static* for f when there is an atomic type ω and an instance I such that $\mathbf{f} \notin f(I \cup \{\mathbf{f}\})$ and $\mathit{type}(\mathbf{f}) = \omega$. That is, $\mathbf{f}$ has an atomic type and there is an instance for which $\mathbf{f}$ is not broadcast. We say that a semi-static fact $\mathbf{f}$ (for f) *depends* on a fact $\mathbf{g}$, when $\mathbf{f} \notin f(I \cup \{\mathbf{f}\})$ implies $\mathbf{g} \in I$ for every instance I. With every semi-static fact $\mathbf{f}$, we associate the set $D_{\mathbf{f}}$ containing exactly all facts on which $\mathbf{f}$ depends. Thus, $\mathbf{f} \notin f(I \cup \{\mathbf{f}\})$ implies $D_{\mathbf{f}} \subseteq I$.

We make use of the following lemma in the proof of Theorem 18.

▶ **Lemma 17.** *Let Q be a CQ, and f be a local optimal OBF for Q. Let $\boldsymbol{f}$ be semi-static for f. Then, $\boldsymbol{f} \notin f(D_{\boldsymbol{f}} \cup \{\boldsymbol{f}\})$. Furthermore, $\boldsymbol{g} \in D_{\boldsymbol{f}}$ implies*

1. *$\boldsymbol{g}$ is semi-static and $\boldsymbol{g} \sim_Q \boldsymbol{f}$;*
2. *$Adom(\boldsymbol{g}) \subseteq Adom(\boldsymbol{f})$;*
3. *$Vars(atom(\boldsymbol{g})) \subseteq_{type(\boldsymbol{g})} Vars(atom(\boldsymbol{f}))$; and*
4. *$\boldsymbol{g} = V_{\boldsymbol{f},type(\boldsymbol{g})}(atom(\boldsymbol{g}))$;*

We are now ready to prove completeness. The proof of the following theorem shows that the formalism of BDS that only uses complete atomic types can already represent every local optimal OBF.

▶ **Theorem 18** (Completeness). *Let Q be a CQ and f a local optimal OBF for Q. Then, there is a BDS $\mathcal{S}$ for Q such that $f = f_{\mathcal{S}}$.*

Proof. We start by noting that if $\mathbf{f}$ is semi-static for f, then every $\mathbf{g}$ with $type(\mathbf{f}) = type(\mathbf{g})$ is semi-static for f as well. Therefore, we say that an atomic type τ is semi-static for f when there is a semi-static fact $\mathbf{f}$ with $type(\mathbf{f}) = \tau$. The proof is by construction. Let $\mathcal{S}$ be the set of pairs (τ, D_τ) where τ is semi-static for f and $D_\tau = Types(D_{\mathbf{f}})$, where $\mathbf{f}$ is a fact with atomic type τ.

We first show that $\mathcal{S}$ is a BDS and then that $f = f_{\mathcal{S}}$. Notice that, $\mathcal{S}$ has only finitely many pairs, because there are only finitely many distinct atomic-types, and every set in $Values(\mathcal{S})$ is finite by construction. Let $(\tau, T) \in \mathcal{S}$, and $\tau' \in T$. By construction of $\mathcal{S}$, τ is a semi-static atomic type for f and for every atomic type τ there is at most one pair $(\tau, T) \in \mathcal{S}$. Furthermore, $T = D_\tau$. Let $\mathbf{f}$ be a fact of type τ. Then, $\mathbf{f}$ is a semi-static fact for f and there is a $\mathbf{g} \in D_{\mathbf{f}}$, such that $type(\mathbf{g}) = \tau'$. By Lemma 17(3), $Vars(atom(\tau')) = Vars(atom(\mathbf{g})) \subseteq_{type(\mathbf{g})} Vars(atom(\mathbf{f})) = Vars(atom(\tau))$. So, $\mathcal{S}$ is a broadcast dependency set for query Q.

Next, we show that $f = f_{\mathcal{S}}$. For this, we assume $D_{\mathbf{f}} = Dep(\mathbf{f}, D_{type(\mathbf{f})})$ (which is argued below) and show that $\mathbf{f} \notin f(I \cup \{\mathbf{f}\})$ iff $\mathbf{f} \notin f_{\mathcal{S}}(I \cup \{\mathbf{f}\})$.

Let $\mathbf{f}$ be a fact and I an instance, such that $\mathbf{f} \notin f(I \cup \{\mathbf{f}\})$. If $\mathbf{f}$ has no atomic type, then it is never broadcast by $f_{\mathcal{S}}$. So, assume $\mathbf{f}$ has an atomic type. Then it must be that $D_{\mathbf{f}} \subseteq I$. However, because $(type(\mathbf{f}), D_{type(\mathbf{f})}) \in \mathcal{S}$ and $D_{\mathbf{f}} = Dep(\mathbf{f}, D_{type(\mathbf{f})})$, $Dep(\mathbf{f}, \mathcal{S}) \subseteq I$. Hence, by definition of $f_{\mathcal{S}}$, $\mathbf{f} \notin f_{\mathcal{S}}(I \cup \{\mathbf{f}\})$.

For fact $\mathbf{f}$ and instance I, where $\mathbf{f} \in f(I \cup \{\mathbf{f}\})$, Lemma 9 implies that $\mathbf{f}$ has an atomic type. Either, $\mathbf{f}$ is always broadcast by f, or it is semi-static for $\mathbf{f}$. The former implies that there is no pair in $\mathcal{S}$ of the form $(type(\mathbf{f}), T)$. So, $\mathbf{f}$ is broadcast by $f_{\mathcal{S}}$ as well. The latter implies by Lemma 17 that $D_{\mathbf{f}} \not\subseteq I$ and there is a pair $(type(\mathbf{f}), D_{type(\mathbf{f})}) \in \mathcal{S}$. In particular, because $Dep(\mathbf{f}, D_{type(\mathbf{f})}) = D_{\mathbf{f}}$, $Dep(\mathbf{f}, D_{type(\mathbf{f})}) \not\subseteq I$, which implies that $\mathbf{f} \notin f_{\mathcal{S}}(I \cup \{\mathbf{f}\})$.

It remains to show that $D_{\mathbf{f}} = Dep(\mathbf{f}, D_{type(\mathbf{f})})$. Because $\mathbf{g} \in D_{\mathbf{f}}$, implying $type(\mathbf{g}) \in D_{type(\mathbf{f})}$, it follows by Lemma 17(4) that $\mathbf{g} \in Dep(\mathbf{f}, D_{type(\mathbf{f})})$. For the reverse direction, let $\mathbf{g} \in Dep(\mathbf{f}, D_{type(\mathbf{f})})$, which implies $type(\mathbf{g}) \in D_{type(\mathbf{f})}$. So, there must be some fact $\mathbf{g}'$, which is of the same type as $\mathbf{g}$, in $D_{\mathbf{f}}$. In particular, because $D_{\mathbf{f}} \subseteq Dep(\mathbf{f}, D_{type(\mathbf{f})})$, $\mathbf{g}' = V_{\mathbf{f},type(\mathbf{g}')}(atom(\mathbf{g}'))$. However, because $\mathbf{g} = V_{\mathbf{f},type(\mathbf{g})}(atom(\mathbf{g}))$, $atom(\mathbf{g}) = atom(\mathbf{g}')$, and $type(\mathbf{g}') = type(\mathbf{g})$, it must be that $\mathbf{g} = \mathbf{g}'$. So, indeed $\mathbf{g} \in D_{\mathbf{f}}$. ◀

6 Algorithms for constructing a BDS

Lemma 13 and Lemma 15 yield a natural algorithm for constructing a local optimal OBF for a given conjunctive query Q by simply starting from $\mathcal{S} = \emptyset$ and adding new pairs in a one by one fashion till no more pairs can be added. More formally, we introduce the algorithm

Input: conjunctive query Q
Param: sequence of partial types $\mathcal{R}$
$\mathcal{S} = \emptyset$;
foreach $\tau \in \mathcal{R}$ **do**
  addPair = *true*;
  Values = $\emptyset$;
  foreach $\tau' \in \mathit{Keys}(\mathcal{S})$, *where* $\tau' \sim_Q \tau$ **do**
    Values = Values $\cup\{\tau'\}$;
    if $\mathit{Vars}(\tau') \not\subseteq_{\tau'} \mathit{Vars}(\tau)$ **then**
      addPair = *false*;
    end
  end
  if addPair **then**
    $\mathcal{S} = \mathcal{S} \cup \{(\tau, \text{Values})\}$;
  end
end
return $\mathcal{S}$

Algorithm 1: Algorithm BDS-BUILD.

BDS-BUILD, given in Algorithm 1. As there are exponentially many (in the size of Q) partial atomic types, we parameterize BDS-BUILD by a sequence $\mathcal{R}$ of partial atomic types.[4] The algorithm then produces a set of pairs $(\tau, T) \in \mathit{PTypes}(Q) \times 2^{\mathit{PTypes}(Q)}$.

The following theorem obtains the correctness of BDS-BUILD. The complexity follows directly from the size of $\mathcal{R}$ which is polynomial in the size of Q for open types and exponential for complete types.

▶ **Theorem 19.** *For a conjunctive query Q and a sequence $\mathcal{R}$ consisting of exactly the complete (respectively, open) types,* BDS-BUILD(Q) *computes a BDS $\mathcal{S}$ for Q in time exponential (respectively, polynomial) in the size of Q such that $f_{\mathcal{S}}$ is correct for Q and local optimal.*

▶ **Example 20.** We illustrate BDS-BUILD by means of an example.
Consider the conjunctive query $Q(x, y, z, w) \leftarrow A(x, y, z), B(x, y, z), C(z, w)$.

1. **Open types.** Observe that query Q has three open types, being $\tau_A = (A, \mathit{true})$, $\tau_B = (B, \mathit{true})$, and $\tau_C = (C, \mathit{true})$. Let $\mathcal{R} = (\tau_A, \tau_B, \tau_C)$. Then, BDS-BUILD computes a BDS by starting from $\mathcal{S} = \emptyset$, expanding $\mathcal{S}$ to $\{(\tau_A, \emptyset)\}$ in the first iteration and to $\{(\tau_A, \emptyset), (\tau_B, \{\tau_A\})\}$ in the second iteration. During the last iteration, $\mathcal{S}$ is not changed anymore, because $\mathit{Vars}(\tau_A) \not\subseteq_{\tau_A} \mathit{Vars}(\tau_C)$.
2. **Complete types.** The (complete) atomic types for Q are

$$\tau_X^{\neq} = (X, x \neq y \wedge y \neq z \wedge x \neq z), \qquad \tau_X^{x=z} = (X, x = z \wedge z \neq y \wedge y \neq z),$$
$$\tau_X^{x=y} = (X, x = y \wedge x \neq z \wedge y \neq z), \qquad \tau_X^{y=z} = (X, x \neq y \wedge y = z \wedge z \neq x),$$
$$\tau_X^{=} = (X, x = y \wedge x = z \wedge y = z), \qquad \tau_C^{=} = (C, z = w), \text{ and } \tau_C^{\neq} = (C, z \neq w),$$

where $X \in \{A, B\}$.[5] Let $\mathcal{R} = (\tau_B^{\neq}, \tau_C^{=}, \tau_C^{\neq}, \tau_B^{x=z}, \tau_A^{x=y}, \tau_A^{\neq}, \tau_A^{x=z}, \tau_A^{=}, \tau_B^{=}, \tau_A^{y=z}, \tau_B^{x=y}, \tau_B^{y=z})$.

[4] We use a sequence rather than a set $\mathcal{R}$ to keep BDS-BUILD deterministic.
[5] For convenience we represent atomic types here by partial atomic types with sufficient (but not complete) conditions; e.g., we write $(C, x = y)$ to denote $(C, x = y \wedge y = x)$. Nevertheless, all of the listed pairs indeed correspond to a single (complete) atomic type.

Then, the output of algorithm BDS-BUILD(Q) is the BDS $\mathcal{S} = \{(\tau_B^{\neq}, \emptyset), (\tau_B^{x=z}, \emptyset), (\tau_A^{x=y}, \emptyset), (\tau_A^{\neq}, \{\tau_B^{\neq}\}), (\tau_A^{x=z}, \{\tau_B^{x=z}\}), (\tau_A^{=}, \emptyset), (\tau_B^{=}, \{\tau_A^{=}\}), (\tau_A^{y=z}, \emptyset), (\tau_B^{x=y}, \{\tau_A^{x=y}\}), (\tau_B^{y=z}, \{\tau_A^{y=z}\})\}$. Observe that the atomic types $\tau_C^{=}$ and $\tau_C^{\neq}$ are not part of $\mathcal{S}$ because the variable containment condition is not satisfied by the earlier included atomic type $\tau_B^{\neq}$.
Observe that the constructed BDS $\mathcal{S}$ can be simplified by merging multiple atomic types into partial atomic types; e.g., for $\mathcal{S}' = \{(\tau_A, \{\tau_B^{\neq}, \tau_B^{x=z}\}), (\tau_B, \{\tau_A^{x=y}, \tau_A^{=}, \tau_A^{y=z}\})\}$, we have $f_{\mathcal{S}} = f_{\mathcal{S}'}$. ◀

Notice that when $\mathcal{R}$ consists of the complete or open atomic types, adding pairs to a given BDS $\mathcal{S}$ as is done by BDS-BUILD(Q) results in a BDS $\mathcal{S}'$ that describes an OBF that broadcasts strictly less facts, i.e., $f_{\mathcal{S}'} \subsetneq f_{\mathcal{S}}$. That is, adding pairs optimizes the OBF.

▶ Remark. By construction, BDS-BUILD(Q) prevents any circular dependencies by stratifying the construction of $\mathcal{S}$ so that partial atomic types can only depend on partial atomic types that where added before. As illustrated in Example 12(4), dependencies in a BDS can also be circular. To allow for these BDS-BUILD can be modified as follows: as an alternative for adding pairs (τ, T) where every existing key that is compatible with τ is included in T, we can allow adding pairs where some keys that are compatible with τ are in T, and for every other compatible key, their respective value set is expandend to contain τ; i.e., allowing pairs of the form (τ, D), where D is a subset of $C = \{\omega' \in \mathit{Keys}(\mathcal{S}) \mid \omega' \sim_Q \omega\}$ satisfying $\mathit{Vars}(\omega') \subseteq_{\omega'} \mathit{Vars}(\omega)$ for every $\omega' \in D$, and where every existing pair (ω', T), where $\omega' \in C \setminus D$, is expanded to $(\omega', T \cup \{\omega\})$. Particularly notice that when a given BDS $\mathcal{S}$ is changed to $\mathcal{S}'$ by adding a pair and expanding at least one of the existing pairs as described above, the inherent nature of the described OBF changes, so that not necessarily $f_{\mathcal{S}'} \subsetneq f_{\mathcal{S}}$.

▶ Remark. Although the machinery developed throughout this paper is motivated by gaining a better understanding of the spectrum of local optimal OBFs, the reader may notice that when no (statistical) information on the actual distribution of the data is available, there is no basis to favor one local optimal OBF over another.

In fact, there is already a very simple algorithm to find an arbitrary local optimal OBF for given CQ Q which is as good as any local optimal one (when no additional information on the distribution of the data is available). Indeed, consider an arbitrary order on the predicates of Q:

> For every local fact $\mathbf{f}$, with predicate R, if there is an earlier predicate S such that some variable in $\mathit{Vars}(S)$ is not in $\mathit{Vars}(R)$, $\mathbf{f}$ is broadcast; otherwise, $\mathbf{f}$ is broadcast only if all the facts induced by $V_{\mathbf{f}}$ on query Q are in the local instance.

Of course, not every local optimal OBF can take this form.

7 Discussion

We investigated local optimal oblivious broadcasting functions represented by the formalism of broadcast dependency sets. We obtained semantical and syntactical characterizations, showed completeness of BDSs for representing local optimal OBFs, and gave an algorithm for constructing local optimal OBFs for a given conjunctive query. We present several directions for future work: more expressive query languages, incorporating background knowledge, and non-oblivious broadcast functions.

An obvious question is how to generalize our results to the class of all conjunctive queries (possibly extended with negation) or even to (subsets of) Datalog. Of course, to evaluate non-

monotonic queries in a coordination-free manner, computing nodes need more information on how data is distributed (c.f., [6]).

We only discussed how to build a BDS when no information about the way data is distributed is available. Indeed, the best one can do is to let a BDS cover as much types as possible, but at the same time introduce as little dependencies as possible, as these are likely to fail when data is arbitrarily distributed. It would be interesting to devise optimal broadcasting algorithms taking more background knowledge into account like information about clustering of attributes, foreign keys, or cardinality of relations.

Another interesting direction for future work is to investigate non-oblivious broadcasting functions where over time, when new messages arrive, static facts can become broadcast facts (but not vice versa). Such functions are initially more conservative keeping more facts static and only broadcast facts when there is some evidence that they can be used at another computing node. For instance, consider the setting of Example 1. Rather than immediately sending $B(i, j)$ whenever $A(j, i)$ is locally absent, broadcasting is suspended until a C-fact of the form $C(j, k)$ is received. The rationale is that a B-fact that can not contribute to a locally satisfying valuation, should only be broadcast when some evidence is received that it could potentially contribute to a satisfying valuation on a remote node. For our example this means that c waits to send $B(2, 1)$ until $C(1, 3)$ arrives. Moreover, $B(4, 4)$ is never sent. While non-oblivious strategies might seem more attractive as they transmit fewer tuples, such strategies, while remaining coordination-free, can increase the overall evaluation time.

Acknowledgment. We thank Phokion Kolaitis for raising the question whether it is always necessary to broadcast all the data in the context of the work in [5]. We thank the reviewers for their in-depth comments and numerous suggestions for improving the presentation of the results.

References

1 F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *International Conference on Extending Database Technology (EDBT)*, pages 99–110, 2010.

2 Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. Parallel skyline queries. In *International Conference on Database Theory (ICDT)*, pages 274–284, 2012.

3 Peter Alvaro, Neil Conway, Joe Hellerstein, and William R. Marczak. Consistency analysis in bloom: a CALM and collected approach. In *Conference on Innovative Data Systems Research (CIDR)*, pages 249–260, 2011.

4 Peter Alvaro, Neil Conway, Joseph M. Hellerstein, and David Maier. Blazes: Coordination analysis for distributed programs. In *International Conference on Data Engineering (ICDE)*, pages 52–63. IEEE, 2014.

5 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. In *Symposium on Principles of Database Systems (PODS)*, pages 64–75. ACM, 2014.

6 Tom J. Ameloot, Frank Neven, and Jan Van den Bussche. Relational transducers for declarative networking. *Journal of the ACM*, 60(2):15, 2013.

7 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Symposium on Principles of Database Systems (PODS)*, pages 273–284, 2013.

8 Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *Symposium on Principles of Database Systems (PODS)*, pages 212–223, 2014.

9 Peter Buneman, James Cheney, Wang Chiew Tan, and Stijn Vansummeren. Curated databases. In *Symposium on Principles of Database Systems (PODS)*, pages 1–12. ACM, 2008.

10 Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *International Conference on Database Theory (ICDT)*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001.

11 Neil Conway, William R. Marczak, Peter Alvaro, Joseph M. Hellerstein, and David Maier. Logic and lattices for distributed programming. In *Symposium on Cloud Computing (SoCC)*, page 1. ACM, 2012.

12 Wenfei Fan, Floris Geerts, and Leonid Libkin. On scale independence for querying big data. In *Symposium on Principles of Database Systems (PODS)*, pages 51–62. ACM, 2014.

13 Sumit Ganguly, Abraham Silberschatz, and Shalom Tsur. Parallel bottom-up processing of datalog queries. *Journal of Logic Programming*, 14(1&2):101–126, 1992.

14 Joseph M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010.

15 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *Symposium on Principles of Database Systems (PODS)*, pages 223–234, 2011.

16 Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y. Halpern, Christoph Koch, Katherine F. Moore, and Dan Suciu. Causality in databases. *IEEE Data Engineering Bulletin*, 33(3):59–67, 2010.

17 Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proceedings of the VLDB Endowmen (PVLDB)*, 4(1):34–45, 2010.

18 Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28. USENIX Association, 2012.

19 Daniel Zinn, Todd J. Green, and Bertram Ludäscher. Win-move is coordination-free (sometimes). In *International Conference on Database Theory (ICDT)*, pages 99–113, 2012.

Datalog Queries Distributing over Components

Tom J. Ameloot[*1], Bas Ketsman[†1], Frank Neven[1], and Daniel Zinn[2]

1 **Hasselt University & transnational University of Limburg**
Martelarenlaan 42, Hasselt, Belgium
`firstname.lastname@uhasselt.be`

2 **LogicBlox, Inc**
1416 NW 46th St., Suite 301, Seattle, WA 98103, USA
`daniel.zinn@logicblox.com`

Abstract

We investigate the class $\mathcal{D}$ of queries that distribute over components. These are the queries that can be evaluated by taking the union of the query results over the connected components of the database instance. We show that it is undecidable whether a (positive) Datalog program distributes over components. Additionally, we show that connected Datalog$^\neg$ (the fragment of Datalog$^\neg$ where all rules are connected) provides an effective syntax for Datalog$^\neg$ programs that distribute over components under the stratified as well as under the well-founded semantics. As a corollary, we obtain a simple proof for one of the main results in previous work [19], namely, that the classic win-move query is in $\mathcal{F}_2$ (a particular class of coordination-free queries).

1998 ACM Subject Classification H.2.3 Query Languages, H.2.4 Distributed databases

Keywords and phrases Datalog, stratified semantics, well-founded semantics, coordination-free evaluation, distributed databases

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.308

1 Introduction

A Datalog program is called connected when the graph of every rule is connected; here, the graph of a rule views the variables of the rule as vertices and each positive body atom as a hyperedge. For instance, the canonical program computing the transitive closure of a binary relation

$$TC(x,y) \leftarrow E(x,y)$$
$$TC(x,y) \leftarrow E(x,z), TC(z,y)$$

is connected, while the program

$$A(x,y) \leftarrow E(x,z), E(y,z')$$

is not as $E(x,z)$ and $E(y,z')$ do not share a common variable. The definition of connectedness can also be extended to Datalog$^\neg$ (with negation), where the negative body atoms of a rule do not contribute to the graph of this rule. While connected Datalog programs are very natural, as a logic they are not well-understood. The earliest reference to connected Datalog is by Guessarian [11] who obtained a decidability result for boundedness of a subclass of

[*] Postdoctoral Fellow of the Research Foundation – Flanders (FWO)
[†] PhD Fellow of the Research Foundation – Flanders (FWO)

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 308–323

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

connected Datalog programs. Ameloot et al. [4] obtained that every connected stratified Datalog$^\neg$ program distributes over components, that is, the program can be evaluated by taking the union of the query results over the connected components of the database instance (cf. Section 3 for a formal definition). We denote by $\boldsymbol{\mathcal{D}}$ the class of queries that distribute over components.

In this paper, we investigate the relationship between connected Datalog$^\neg$ and the class $\boldsymbol{\mathcal{D}}$. This investigation is motivated by a general theme in model theory that considers the relationship between syntactic and semantic properties of logic. In the context of Datalog, results of this type have for example been obtained for the class of queries preserved under homomorphisms, denoted by $\mathcal{H}$. For instance, Feder and Vardi [10] showed that all queries in semi-positive Datalog$^\neg$ that are preserved under homomorphisms can already be expressed in Datalog itself. That is, semi-positive Datalog$^\neg \cap \mathcal{H}$ = Datalog. Dawar and Kreutzer [9] showed that the latter result can not be extended to least fixed-point logic (LFP): LFP $\cap \mathcal{H} \not\subseteq$ Datalog. The main result of this paper is that both under the stratified as well as under the well-founded semantics, we have connected Datalog$^\neg$ = Datalog$^\neg \cap \boldsymbol{\mathcal{D}}$. Additionally, we show that, even when we forbid negation in rules, it is undecidable whether a given (positive) Datalog program is in $\boldsymbol{\mathcal{D}}$. Our main result therefore shows that connected Datalog$^\neg$ is an effective syntax for queries in Datalog$^\neg \cap \boldsymbol{\mathcal{D}}$ (both under the stratified and under the well-founded semantics).

Apart from the model theoretic motivation mentioned above, the results in this paper also provide more insight in some of the recent results concerning coordination-free evaluation. Datalog has attracted quite a bit of attention as a declarative programming language for distributed systems, see e.g. [14, 1, 16]. In fact, Hellerstein [12] argues that the theory of declarative database query languages can provide a foundation for the next generation of parallel and distributed programming languages. In this respect, programs (queries) are specified on a logical level over a global schema and are computed by multiple computing nodes over which the input database is distributed. These nodes can perform local computations and communicate asynchronously with each other via messages. The model operates under the assumption that messages can never be lost but can be arbitrarily delayed. As the global barriers raised by the need for synchronization are an inherent source of inefficiency in such systems, a number of researchers started investigating classes of queries that can be evaluated in a coordination-free manner [8, 19, 5, 3, 4]. In a coordination-free evaluation, communication between nodes can only transfer data and can not be used to coordinate.[1] Zinn, Green, and Ludäscher [19] introduced various classes of coordination-free queries: $\boldsymbol{\mathcal{F}}_0$, $\boldsymbol{\mathcal{F}}_1$, and $\boldsymbol{\mathcal{F}}_2$. Membership of the classical non-monotonic win-move query in $\boldsymbol{\mathcal{F}}_2$ is one of the main results in [19], where the authors describe a distributed query evaluation strategy that is specific for the win-move query. The results in this paper provide a more in-depth explanation of that result. Indeed, letting $\boldsymbol{\mathcal{V}}$ denote the class of so-called *value-driven* queries that have nonempty output only on inputs containing values, we explain that every query in $\boldsymbol{\mathcal{D}} \cap \boldsymbol{\mathcal{V}}$ is also in the class $\boldsymbol{\mathcal{F}}_2$. This implies that every connected Datalog$^\neg$ program in $\boldsymbol{\mathcal{V}}$ is in $\boldsymbol{\mathcal{F}}_2$ as well. Since win-move is a connected Datalog$^\neg$ program, and is value-driven, it then follows immediately that win-move is in $\boldsymbol{\mathcal{F}}_2$.

In this paper, we also briefly discuss *semi-connected* Datalog$^\neg$ programs, a relaxation of connected Datalog$^\neg$ introduced in previous work [4]: under the well-founded semantics, the queries expressible by semi-connected Datalog$^\neg$ programs remain in the class $\boldsymbol{\mathcal{F}}_2$.

[1] We refer to [5, 4] for a formal definition of coordination-freeness.

Outline. This paper is organized as follows. Section 2 presents preliminaries on databases, and on Datalog$^\neg$ together with its stratified and well-founded semantics. Section 3 recalls distribution over components; additionally, we present an undecidability result and we discuss the relationship with weaker forms of monotonicity. Next, Section 4 discusses a syntactic restriction of Datalog$^\neg$, called *connected Datalog*$^\neg$; and, we show that under the stratified semantics, this restriction captures the queries that both distribute over components and that are expressible in Datalog$^\neg$. This capturing result is subsequently generalized in Section 5 to the well-founded semantics. Section 6 briefly discusses how a relaxation of the connectedness restriction behaves under the well-founded semantics. We conclude in Section 7. Many of the technical proofs are moved to the appendix due to space limitations. Proof sketches are given in the main body.

2 Preliminaries

2.1 Database Basics

A *(database) schema* σ is a finite set of pairs (R, k), also denoted as $R^{(k)}$, where R is a relation name and $k \in \mathbb{N}$ its associated arity. We assume an infinite universe **dom** of atomic data values. A *fact* is a pair $(R, \bar{a})$, also denoted as $R(\bar{a})$, where R is a relation name and $\bar{a}$ is a (possibly empty) tuple of values over **dom**. We say that fact $R(a_1, \ldots, a_k)$ is *over* database schema σ if $R^{(k)} \in \sigma$. If $k = 0$ then the fact is called *nullary*. A *(database) instance* I over σ is a finite set of facts over σ. The *active domain* of a fact $\boldsymbol{f}$, denoted $adom(\boldsymbol{f})$, is the set of values occurring in $\boldsymbol{f}$. For an instance I, we define $adom(I) = \bigcup_{\boldsymbol{f} \in I} adom(\boldsymbol{f})$. For a subset $\sigma' \subseteq \sigma$, we write $I|_{\sigma'}$ to denote the maximal subset of I that is over σ'.

A *query* $\mathcal{Q}$ *over input schema* σ_1 *and output schema* σ_2 is a function that maps instances over σ_1 to instances over σ_2. We consider only queries that are *generic*: these queries are independent of the concrete data values. More formally, a query $\mathcal{Q}$ is generic if for all inputs I, and all permutations ρ of **dom**, we have $\rho(\mathcal{Q}(I)) = \mathcal{Q}(\rho(I))$.

2.2 Datalog with Negation

We recall here the language Datalog with negation [2], denoted Datalog$^\neg$.

Atoms and rules

We assume a separate universe **var** of variables. An *atom* is a pair $(R, \bar{u})$, also denoted as $R(\bar{u})$, where R is a relation name and $\bar{u}$ is a (possibly empty) tuple of variables over **var**. We say that an atom $R(u_1, \ldots, u_k)$ is *over* a database schema σ if $R^{(k)} \in \sigma$. A *rule* φ is a tuple $(head_\varphi, pos_\varphi, neg_\varphi)$ where $head_\varphi$ is an atom, and pos_φ and neg_φ are both sets of atoms. We call $head_\varphi$ the *head*; and we call pos_φ and neg_φ respectively the *positive body atoms* and the *negative body atoms*. We only consider rules φ where each variable in $head_\varphi$ and neg_φ also occurs in pos_φ. We say that φ is *over* a database schema σ when all its atoms are over σ. A rule φ may also be written in the conventional syntax, e.g., when $head_\varphi = T(\bar{u})$, $pos_\varphi = \{R_1(\overline{u_1}), \ldots, R_m(\overline{u_m})\}$ and $neg_\varphi = \{S_1(\overline{v_1}), \ldots, S_n(\overline{v_n})\}$ then we may write φ as:

$$T(\bar{u}) \leftarrow R_1(\overline{u_1}), \ldots, R_m(\overline{u_m}), \neg S_1(\overline{v_1}), \ldots, \neg S_n(\overline{v_n}).$$

The ordering of atoms to the right of the arrow is arbitrary.

A *valuation* for rule φ is a function V that maps each variable in φ to a value in **dom**. Applying V to atoms of φ results in facts: we substitute each variable u by $V(u)$. We say

that V is *satisfying for φ on an instance I*, when $V(pos_\varphi) \subseteq I$ and $V(neg_\varphi) \cap I = \emptyset$. In that case, the pair (φ, V) is said to *derive* the fact $V(head_\varphi)$ on instance I.

Programs

A *Datalog program with negation over a schema σ* is a set P of rules over σ. The class of such programs is denoted by Datalog$^\neg$. For a Datalog$^\neg$ program P, we also write $sch(P)$ to denote the (minimal) schema that P is over. We define $idb(P) \subseteq sch(P)$ as the relations of $sch(P)$ that appear in rule heads. We also define $edb(P) = sch(P) \setminus idb(P)$.[2] Intuitively, $edb(P)$ can be seen as the input relations for P. Various semantics can be given to Datalog$^\neg$ programs. In this paper we use the stratified semantics and the well-founded semantics.

2.3 Stratified Semantics

We recall the stratified semantics of Datalog$^\neg$ [2].

Semi-positive programs

We call a Datalog$^\neg$ program P *semi-positive* when its rules only apply negation to relations in $edb(P)$. More formally, for all rules φ in P, the set neg_φ is over $edb(P)$. The semantics of such a program P can be defined as follows. Consider the following function T_P, called the *(inflationary) immediate consequence operator* of P: T_P maps any instance J over $sch(P)$ to $J \cup A$ where $A = \{V(head_\varphi) \mid \varphi \in P, V \text{ is a satisfying valuation for } \varphi \text{ on } J\}$. Now, for an input I over $edb(P)$, consider the following infinite sequence of instances: $I_0, I_1, I_2, \ldots$ where $I_0 = I$ and $I_i = T_P(I_{i-1})$ for all $i \geq 1$. Because T_P only adds facts, and is limited to $adom(I)$, there is an index k such that $I_k = I_{k+1}$, i.e., there is a fixpoint. The output of P on I is defined as this fixpoint.

Syntactic stratification

Let P be a Datalog$^\neg$ program. We call P *syntactically stratifiable* (or simply *stratified*) if we can partition the rules of P into a sequence of Datalog$^\neg$ subprograms $P_1, \ldots, P_n$ such that:

- Rules with the same head relation occur in the same subprogram;
- In each subprogram P_i, relations R of positive body atoms either belong to $edb(P)$ or all rules computing R must be in some subprogram P_j with $j \leq i$; and,
- In each subprogram P_i, relations R of negative body atoms either belong to $edb(P)$ or all rules computing R must be in some subprogram P_j with $j < i$.

Each subprogram is also called a *stratum*. Note that negation is only applied to relations computed in strictly lower strata. So, each stratum by itself is a semi-positive program. Given a syntactic stratification $P_1, \ldots, P_n$, the output of P on an input I over $edb(P)$, denoted $P(I)$, is defined as $P_n(P_{n-1}(\ldots(P_1(I))\ldots))$, i.e., we first apply stratum P_1, then stratum P_2, etc. All syntactic stratifications give the same result [2].

We say that a query $\mathcal{Q}$ with input schema σ_1 and output schema σ_2 is *computed by a stratified Datalog$^\neg$ program* P if for all inputs I for $\mathcal{Q}$, we have $\mathcal{Q}(I) = P(I)|_{\sigma_2}$ using the stratified semantics of P.

[2] The abbreviations "idb" and "edb" respectively stand for *intensional schema* and *extensional schema*.

2.4 Well-founded Semantics

Let P be a Datalog$^\neg$ program. We define the well-founded semantics of P using the *alternating fixpoint computation* [18].

Negation on assumptions

Let φ be a rule in P, and let J be an instance over $sch(P)$. A valuation V for φ is said to be *J-neg-satisfying for φ on an instance I* if $V(pos_\varphi) \subseteq I$ and $V(neg_\varphi) \cap J = \emptyset$. In contrast to the semantics of semi-positive programs from above, J-neg-satisfaction tests negative body atoms against the fixed database instance J. Next, consider the following function T_P^J, called the *(inflationary) immediate consequence operator of P with assumptions J*: function T_P^J maps each instance K over $sch(P)$ to $K \cup A$ where $A = \{V(head_\varphi) \mid \varphi \in P,\ V$ is a J-neg-satisfying valuation for φ on $K\}$. Now, for an instance I over $sch(P)$, consider the following infinite sequence of instances: $I_0, I_1, I_2, \ldots$ where $I_0 = I$ and $I_i = T_P^J(I_{i-1})$ for all $i \geq 1$. Because T_P^J only adds facts, and is limited to $adom(I)$, there is an index k such that $I_k = I_{k+1}$, i.e., there is a fixpoint, denoted as $\hat{T}_{P,I}^J$.

Antimonotone operator

Let I be an input over $edb(P)$. Let $\Gamma_{P,I}$ denote the function that maps each instance J over $sch(P)$ to $\hat{T}_{P,I}^J$. Note that for any two instances J_1 and J_2 with $J_1 \subseteq J_2$, the J_2-neg-satisfying valuations are also J_1-neg-satisfying, so $\Gamma_{P,I}(J_2) \subseteq \Gamma_{P,I}(J_1)$. For this reason, we call $\Gamma_{P,I}$ *antimonotone*. We similarly see that $\Gamma_{P,I}(\Gamma_{P,I}(J_1)) \subseteq \Gamma_{P,I}(\Gamma_{P,I}(J_2))$, so function $\Gamma_{P,I} \circ \Gamma_{P,I}$ is monotone. Next, consider the following infinite sequence of instances: $I_0, I_1, I_2, \ldots$ where $I_0 = \emptyset$ and $I_i = \Gamma_{P,I}(I_{i-1})$ for all $i \geq 1$.[3] Since $\Gamma_{P,I} \circ \Gamma_{P,I}$ is monotone, the subsequence $I_0, I_2, I_4, \ldots$ converges to a fixpoint, i.e., there is a number $k \geq 0$ such that $I_{2k} = I_{2k+2}$.[4] This implies that index $2k+1$ is a fixpoint for the subsequence with uneven indices, i.e., $I_{2k+1} = I_{2k+3}$. Now, the *well-founded semantics of P on I* produces both a set of *true* facts and a set of *true or undefined* facts, denoted as $P_{\mathrm{t}}(I)$ and $P_{\mathrm{t}\vee\mathrm{u}}(I)$ respectively. Formally, they are defined as $P_{\mathrm{t}}(I) = I_{2k}$ and $P_{\mathrm{t}\vee\mathrm{u}}(I) = I_{2k+1}$. Intuitively, for the facts in $P_{\mathrm{t}\vee\mathrm{u}}(I) \setminus P_{\mathrm{t}}(I)$, we can not compute whether they are definitely present in the output or definitely absent from the output. So, the well-founded semantics essentially assigns one of three truth values to facts over $sch(P)$: true, false, and undefined.

We say that a query $\mathcal{Q}$ with input schema σ_1 and output schema σ_2 is *computed by P under the well-founded semantics* if for all inputs I for $\mathcal{Q}$, we have $\mathcal{Q}(I) = P_{\mathrm{t}}(I)|_{\sigma_2}$.

▶ **Example 1.** We recall the well-known *win-move* Datalog$^\neg$ program P [2]:

$$win(x) \leftarrow move(x, y), \neg win(y).$$

The win-move program represents a game as follows. The input relation *move* is viewed as a graph. A game on this graph starts with one node x of the graph marked with a flag. Next, two players, called 1 and 2, take turns to move the flag from the currently flagged node to one of its successor nodes, and player 1 always gets the first turn. A player looses when

[3] There is an alternating fixpoint: the inner fixpoint is given by $\Gamma_{P,I}(J) = \hat{T}_{P,I}^J$, the outer fixpoint is obtained by iterating $\Gamma_{P,I}$. Applying $\Gamma_{P,I}$ to an underestimate yields an overestimate and vice versa.

[4] To see this, we start with $I_0 = \emptyset \subseteq I_2$. Next, since $\Gamma_{P,I} \circ \Gamma_{P,I}$ is monotone, we have $I_2 = \Gamma_{P,I} \circ \Gamma_{P,I}(I_0) \subseteq \Gamma_{P,I} \circ \Gamma_{P,I}(I_2) = I_4$. This reasoning can be repeated to see $I_4 \subseteq I_6$, etc. Since derived facts are restricted to $adom(I)$, we eventually arrive at a fixpoint.

there are no successor nodes during his or her turn. Now, we say that player 1 *has a winning strategy at node* x, if player 1 can always force a win when starting at node x. That is, no matter how player 2 moves, eventually, player 1 will move the flag to a node where player 2 cannot move anymore.

The relation *win* computes the nodes for which player 1 has a winning strategy. For example, letting $\sigma = \{win^{(1)}\}$, on the input $I = \{move(a,b), move(b,a), move(a,c)\}$, we have $P_{\mathrm{t}}(I)|_\sigma = P_{\mathrm{t}\vee\mathrm{u}}(I)|_\sigma = \{win(a)\}$; the winning strategy for player 1 is to move the flag from a to c. As another example, consider the instance $J = I \cup \{move(c,d)\}$. We have $P_{\mathrm{t}}(J)|_\sigma = \{win(c)\}$ and $P_{\mathrm{t}\vee\mathrm{u}}(J)|_\sigma = \{win(a), win(b), win(c)\}$. Facts in $P_{\mathrm{t}\vee\mathrm{u}}(J) \setminus P_{\mathrm{t}}(J)$ represent drawn positions, that is, neither player can force a win and the game goes on indefinitely. The absence of $win(d)$ indicates that player 1 has no winning strategy at node d (because player 1 can not make a move there).

The win-move program is non-monotone: $win(a) \in P_{\mathrm{t}}(I)$ but $win(a) \notin P_{\mathrm{t}}(J)$. ◀

3 Distribution over Components

We recall distribution over components [4]. We call an instance J *connected* if for all values $a, b \in adom(J)$, there exists a sequence $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n$ of facts in J such that $a \in adom(\boldsymbol{f}_1)$, $b \in adom(\boldsymbol{f}_n)$, and $adom(\boldsymbol{f}_i) \cap adom(\boldsymbol{f}_{i-1}) \neq \emptyset$ for all $i \in \{2, \ldots, n\}$. Possibly $n = 1$. Intuitively, any two values are connected by at least one chain of facts, where subsequent facts share at least one value.[5]

Now, for an instance I, we call a subinstance $J \subseteq I$ a *component* of I if *(i)* J includes all nullary facts of I; *(ii)* J is connected and is maximal with this property in I. This implies that $adom(J) \cap adom(I \setminus J) = \emptyset$. We write $co(I)$ to denote the set of components of I.[6] For example, the components of $I = \{R(a,b), R(b,c), S(c), T(d), U(\,)\}$ are $\{R(a,b), R(b,c), S(c), U(\,)\}$ and $\{T(d), U(\,)\}$.

We say that a query $\mathcal{Q}$ *distributes over components* if for all inputs I for $\mathcal{Q}$ we have $\mathcal{Q}(I) = \bigcup_{J \in co(I)} \mathcal{Q}(J)$, i.e., the centralized output of $\mathcal{Q}$ on I is precisely obtained when we parallelize $\mathcal{Q}$ over the components of I. Let $\boldsymbol{\mathcal{D}}$ denote the class of queries that distribute over components.

3.1 Undecidability

To gain additional insight into distribution over components, we consider decidability of this semantical property for the concrete setting of Datalog$^\neg$. First, we call a Datalog$^\neg$ program *positive* if its rules contain no negative body atoms. To denote the class of such programs, we simply write 'Datalog'. For evaluating Datalog programs, we will assume the intuitive semantics of semi-positive Datalog$^\neg$ programs (cf. Section 2.3). Interestingly, despite the restriction,

▶ **Theorem 2.** *Membership in $\boldsymbol{\mathcal{D}}$ is undecidable for queries computable by Datalog programs.*

Proof. First, from previous work by Shmueli [17], we know that it is impossible to decide whether two Datalog programs P_1 and P_2, each with a single non-nullary output relation, are equivalent. This problem was shown to be undecidable by a reduction from equivalence

[5] Equivalently, one could demand that the Gaifman graph of J is connected, where we view $adom(J)$ as the set of vertices and each fact of J as a hyperedge.

[6] The nullary facts are given to each component because there is no natural preference for how to distribute these facts to any particular component.

of context-free grammars. We point out that this reduction actually constructs *connected* programs (see Section 4.1 for a formal definition). So, equivalence, and thus containment, of two connected Datalog programs, each with a single non-nullary output relation, is undecidable. Our proof below reduces this latter containment problem to deciding whether a Datalog program distributes over components.

Let P_1 and P_2 be two connected Datalog programs with the same *edb* schema σ_1 and each having one k-ary intended output relation, denoted A_1 and A_2 respectively, where $k \geq 1$. Both programs may use auxiliary *idb* relations, but for convenience we assume that $idb(P_1)$ and $idb(P_2)$ have no relation names in common. We define the following auxiliary program P', where T and S are relation names not yet used in P_1 and P_2, and all variables are assumed to be pairwise different:

$$T(\bar{u}) \leftarrow A_1(\bar{u}), S(z).$$
$$T(\bar{u}) \leftarrow A_2(\bar{u}).$$

Now consider the program $P^* = P_1 \cup P_2 \cup P'$. Note that $edb(P^*) = \sigma_1 \cup \{S^{(1)}\}$. Although program P^* is positive, it is *not* connected due to the first rule of P'. The S-atom plays the role of a guard: relation A_1 flows into relation T if S is nonempty. Let $\mathcal{Q}$ be the query computed by P^* over output schema $\sigma_2 = \{T^{(k)}\}$. To finish the proof, we show that $\mathcal{Q} \in \boldsymbol{\mathcal{D}}$ if and only if P_1 is contained in P_2.

Suppose that P_1 is contained in P_2. First, we define program P^c as P^* but without the first rule of P'. Note that P^c is connected. For any input I over $edb(P^*)$, since $A_1(\bar{a}) \in P_1(I)$ implies $A_2(\bar{a}) \in P_2(I)$ by containment, we have $P^c(I)|_{\sigma_2} = P^*(I)|_{\sigma_2} = \mathcal{Q}(I)$. And since P^c is a connected program, we can apply Proposition 6 (in Section 4.2), to know $\mathcal{Q} \in \boldsymbol{\mathcal{D}}$.

Suppose that P_1 is not contained in P_2. In particular, there is some input I over σ_1 for which there is a tuple $\bar{a}$ with $A_1(\bar{a}) \in P_1(I)$ and $A_2(\bar{a}) \notin P_2(I)$. Letting d be a new value outside $adom(I)$, we define the instance $I' = I \cup \{S(d)\}$. During the computation of $P^*(I')$, the first rule of subprogram P' has access to $A_1(\bar{a})$ and $S(d)$, giving $T(\bar{a}) \in P^*(I')$. But, the first rule of P' can never be satisfied on any $J \in co(I')$, because by choice of value d, component J does not simultaneously contain non-nullary facts over σ_1 and $\{S^{(1)}\}$. Moreover, for any $J \in co(I')$, we have $A_2(\bar{a}) \notin P_2(J)$: since program P_2 is unaware of S-facts, we have $A_2(\bar{a}) \notin P_2(I')$ and monotonicity of P_2 implies that $A_2(\bar{a})$ can not be produced on any subset of I'. Overall, we have $T(\bar{a}) \notin P^*(J)$ for all $J \in co(I')$. Hence, $\mathcal{Q} \notin \boldsymbol{\mathcal{D}}$. ◀

We now readily observe that:

▶ **Corollary 3.** *Membership in $\boldsymbol{\mathcal{D}}$ is undecidable for queries computable by stratified Datalog$^\neg$ programs.*

3.2 Weaker Forms of Monotonicity

We briefly relate $\boldsymbol{\mathcal{D}}$ to the classes $\boldsymbol{\mathcal{M}}_{\text{distinct}}$ and $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$ [4]. The class $\boldsymbol{\mathcal{M}}_{\text{distinct}}$ consists of the *domain-distinct-monotone* queries: for such queries $\mathcal{Q}$, we have $\mathcal{Q}(I) \subseteq \mathcal{Q}(I \cup J)$ for all instances I and J where each $\boldsymbol{f} \in J$ satisfies $adom(\boldsymbol{f}) \nsubseteq adom(I)$.[7] Intuitively, $\mathcal{Q}$ behaves monotonically when adding facts that contain at least one new value.

We observe that $\boldsymbol{\mathcal{D}} \nsubseteq \boldsymbol{\mathcal{M}}_{\text{distinct}}$: over a schema $\{R^{(1)}, S^{(2)}\}$, consider the query $\mathcal{Q}_1 = R - \pi_1(S)$, where π_1 projects onto the first component. To see $\mathcal{Q}_1 \in \boldsymbol{\mathcal{D}}$, note that when

[7] This implies that J contains no nullary facts.

forming components, the S-facts are always grouped together with those R-facts they subtract from. To see $\mathcal{Q}_1 \notin \boldsymbol{\mathcal{M}}_{\text{distinct}}$, consider the instances $I = \{R(a)\}$ and $J = \{S(a,b)\}$; note that $\mathcal{Q}_1(I) \not\subseteq \mathcal{Q}_1(I \cup J)$.

We also observe that $\boldsymbol{\mathcal{M}}_{\text{distinct}} \not\subseteq \boldsymbol{\mathcal{D}}$: over a schema $\{R^{(1)}, S^{(1)}\}$, consider the query $\mathcal{Q}_2$ that computes the cross product $T = R \times S$. Query $\mathcal{Q}_2$ is monotone, hence $\mathcal{Q}_2 \in \boldsymbol{\mathcal{M}}_{\text{distinct}}$. To see $\mathcal{Q}_2 \notin \boldsymbol{\mathcal{D}}$, consider the instance $I = \{R(a), S(b)\}$. On the full input I, query $\mathcal{Q}_2$ produces $T(a,b)$, but this fact is not produced on component $\{R(a)\}$ nor on component $\{S(b)\}$.

Next, the class $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$ consists of the *domain-disjoint-monotone* queries: for such queries $\mathcal{Q}$, we have $\mathcal{Q}(I) \subseteq \mathcal{Q}(I \cup J)$ for all instances I and J where J contains no nullary facts and $adom(I) \cap adom(J) = \emptyset$.[8]

We observe that $\boldsymbol{\mathcal{D}} \not\subseteq \boldsymbol{\mathcal{M}}_{\text{disjoint}}$: over a schema $\{R^{(1)}\}$, take the query $\mathcal{Q}_3$ that outputs true (in a nullary relation T) if $R = \emptyset$. We see that $\mathcal{Q}_3 \in \boldsymbol{\mathcal{D}}$: if there are multiple components in an input I then each component contains one R-fact, implying that relation T remains empty on each component, giving the same result as the centralized execution $\mathcal{Q}_3(I)$. Also, $\mathcal{Q}_3 \notin \boldsymbol{\mathcal{M}}_{\text{disjoint}}$, because on the instances $I = \emptyset$ and $J = \{R(a)\}$, we have $\mathcal{Q}(I) = \{T(\,)\} \not\subseteq \mathcal{Q}(I \cup J) = \emptyset$.

We also observe that $\boldsymbol{\mathcal{M}}_{\text{disjoint}} \not\subseteq \boldsymbol{\mathcal{D}}$: take the same query $\mathcal{Q}_2$ from above. Since $\boldsymbol{\mathcal{M}}_{\text{distinct}} \subseteq \boldsymbol{\mathcal{M}}_{\text{disjoint}}$, we have $\mathcal{Q}_2 \in \boldsymbol{\mathcal{M}}_{\text{disjoint}}$. But $\mathcal{Q}_2 \notin \boldsymbol{\mathcal{D}}$ as shown above.

When we exclude queries like $\mathcal{Q}_3$, the remaining queries of $\boldsymbol{\mathcal{D}}$ are included in $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$. Formally, we call a query $\mathcal{Q}$ *value-driven* if for all inputs I for $\mathcal{Q}$ with $adom(I) = \emptyset$, we have $\mathcal{Q}(I) = \emptyset$. Intuitively, the query produces nothing in absence of values. Let $\boldsymbol{\mathcal{V}}$ denote the class of such queries. We observe that $\boldsymbol{\mathcal{D}} \cap \boldsymbol{\mathcal{V}} \subseteq \boldsymbol{\mathcal{M}}_{\text{disjoint}}$: for a query $\mathcal{Q} \in \boldsymbol{\mathcal{D}} \cap \boldsymbol{\mathcal{V}}$, *(i)* for an input I with $adom(I) = \emptyset$, we have $\mathcal{Q}(I) = \emptyset \subseteq \mathcal{Q}(I \cup J)$ for all instances J; and *(ii)* for an input I with $adom(I) \neq \emptyset$, and an instance J without nullary facts and with $adom(I) \cap adom(J) = \emptyset$, we have $co(I) \subseteq co(I \cup J)$, so using $\mathcal{Q} \in \boldsymbol{\mathcal{D}}$, we see $\mathcal{Q}(I) = \bigcup_{K \in co(I)} \mathcal{Q}(K) \subseteq \bigcup_{L \in co(I \cup J)} \mathcal{Q}(L) = \mathcal{Q}(I \cup J)$.

4 Connected Datalog

We can not decide for queries computed by stratified Datalog$^\neg$ programs whether they distribute over components (Corollary 3). However, in this section we show there is a fragment of stratified Datalog$^\neg$ that captures precisely the queries of $\boldsymbol{\mathcal{D}}$ expressible in stratified Datalog$^\neg$.

4.1 Connected Syntax

We recall the language of *connected Datalog*$^\neg$, denoted con-Datalog$^\neg$ [4]. We extend the definition in this previous work, however, to explicitly deal with nullary relations. Nullary relations allow more programming flexibility, and they allow boolean computation in absence of input values, e.g., when only a set of nullary facts is given.

As notational convenience, for an atom $\boldsymbol{a}$ we write $var(\boldsymbol{a})$ to denote the set of variables occurring in $\boldsymbol{a}$. Also, for a rule φ we write $var(\varphi)$ to denote the set of variables in φ. Now, very similarly to connected database instances, we say that a rule φ is *connected* when for any two variables $u, v \in var(\varphi)$ there is a sequence of atoms $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ in pos_φ such that

[8] The queries in $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$ are conceptually similar to the first order sentences preserved under closed extensions, studied by Compton [7].

$u \in var(\boldsymbol{a}_1)$, $v \in var(\boldsymbol{a}_n)$, and $var(\boldsymbol{a}_i) \cap var(\boldsymbol{a}_{i-1}) \neq \emptyset$ for all $i \in \{2, \ldots, n\}$. Possibly $n = 1$. Negative body atoms do not contribute to the connectedness of a rule. Note that rules without variables are always connected.

Next, for a Datalog$^\neg$ program P, we say that nullary relations of $edb(P)$ are *global (for all components)* because the nullary input facts are given to all components by definition. Similarly, we say a nullary relation of $idb(P)$ is *global* if all its rules, and the rules of the idb-relations it depends on, do not use variables.[9] So, the term "global" means that these nullary relations will have the same contents on every component. Also, we say that a nullary relation $S^{(0)} \in idb(P)$ is *value-detecting* when *(i)* for each non-nullary relation $R^{(k)} \in edb(P)$, program P contains a rule isomorphic to '$S(\,) \leftarrow R(u_1, \ldots, u_k)$', using pairwise distinct variables; and, *(ii)* there are no other rules for S in P. Such value-detecting relations can only be used to see if the input contains values.

Now, we say that a Datalog$^\neg$ program P is *connected* when

1. every rule of P is connected by itself; and,
2. the only nullary relations used in rule bodies are global or value-detecting.

Nullary relations that are neither global nor value-detecting may still be used for directly representing output.

Note that the win-move program from Example 1 is connected. Below we consider other examples of connected programs.

▶ **Example 4.** Consider the following semi-positive con-Datalog$^\neg$ program P with $edb(P) = \{A^{(1)}, B^{(1)}, R^{(2)}\}$:

$$reach(x) \leftarrow A(x).$$
$$reach(y) \leftarrow reach(x), R(x, y).$$
$$T(x) \leftarrow reach(x), \neg B(x).$$

Thinking of relation R as edges of a graph, this program computes all nodes reachable from set A but outside set B. ◀

▶ **Example 5.** As an example using nullary relations, here is a stratified con-Datalog$^\neg$ program P, with $edb(P) = \{R^{(2)}, S^{(0)}, T^{(0)}, U^{(1)}\}$, whose meaning is discussed below:

$$xor(\,) \leftarrow S(\,), \neg T(\,).$$
$$xor(\,) \leftarrow \neg S(\,), T(\,).$$
$$V(x) \leftarrow xor(\,), U(x).$$
$$V(y) \leftarrow V(x), R(x, y).$$
$$values(\,) \leftarrow R(x, y).$$
$$values(\,) \leftarrow U(x).$$
$$W(\,) \leftarrow \neg values(\,), xor(\,).$$

Suppose that V and W are the output relations. For the nullary relations of $idb(P)$, note that *xor* is global, *values* is value-detecting, and W is neither global nor value-detecting (and hence may not be used in rule bodies). In the presence of values, again thinking of relation R as edges of a graph, program P finds in relation V the nodes reachable from U on condition that the exclusive or $S \oplus T$ is true. In absence of values, P outputs $S \oplus T$ in relation W.

[9] Focusing on $idb(P)$, a relation R *depends on* another relation S if there is a path from R to S in the so-called *dependency graph* of P, where the relations of $idb(P)$ are the vertices and there is an edge from a relation A to a relation B if a rule with head relation A uses B in its body (either positively or negatively).

Note that V and W are never simultaneously nonempty (although they can be simultaneously empty). The output behavior strongly depends on the presence or absence of values; value-detecting relations are needed to achieve this effect. ◀

4.2 Results

We recall the following result [4]:

▶ **Proposition 6.** Every query computable by a stratified con-Datalog$^\neg$ program distributes over components.

The sketch below provides an intuitive understanding.

Proof (sketch). The positive body atoms in connected rules are all strung together. This way, connected rules can only combine facts from the same component, causing derived non-nullary facts to be connected to their originating component. Essentially, a con-Datalog$^\neg$ program derives facts "inside" components. So, the program does not notice when we separate the components.

For completeness, we also discuss the details of nullary relations. First, note that a value-detecting relation S is nonempty on the entire (nondistributed) input if and only if relation S is nonempty on *all* individual components: this property is trivially true when there is only one component; and, when there is more than one component, they each have non-nullary facts.

Next, since each component contains by definition all nullary input facts, nullary facts derived purely from nullary input facts can be seen as "global flags". These global flags may be injected into per-component computations (as represented by rules with variables).[10]

Lastly, nullary relations that are neither global nor value-detecting, can be seen as per-component flags. The syntactic restriction prevents using such flags in further computation. Without this restriction, per-component flags could be combined in a cross-component fashion, preventing distribution over components. ◀

Within stratified Datalog$^\neg$, a new result is that the converse direction also holds:

▶ **Proposition 7.** Every query computable by a stratified Datalog$^\neg$ program, and distributing over components, can be computed by a stratified con-Datalog$^\neg$ program.

Proof (sketch). Let $\mathcal{Q}$ be a query computable by a stratified Datalog$^\neg$ program P, with the additional assumption that $\mathcal{Q}$ distributes over components. Let σ_1 and σ_2 denote the input and output schema of $\mathcal{Q}$. The proof is constructive: we transform P into a stratified con-Datalog$^\neg$ program $\alpha(P)$ such that for all inputs I over σ_1, we have $\alpha(P)(I)|_{\sigma_2} = \mathcal{Q}(I)$, i.e., $\alpha(P)$ also computes $\mathcal{Q}$. The main idea behind $\alpha(P)$ is that it uses connected rules to separate the original computation over the components (as sketched for Proposition 6). Concretely, $\alpha(P)$ is defined as a union of four subprograms: $\alpha(P) = P^{\uparrow} \cup P^{\#} \cup P^{\downarrow} \cup P^{\text{null}}$. In particular, there are two parts: subprogram $(P^{\uparrow} \cup P^{\#} \cup P^{\downarrow})$ is executed in case there are values in the input, and otherwise the subprogram P^{null} is executed (on just the nullary input facts). Roughly speaking, the order $P^{\uparrow}$, $P^{\#}$, P^{null}, $P^{\downarrow}$ aligns with a syntactic stratification for $\alpha(P)$. Below we explain each subprogram in turn. We also provide an illustration in Example 8. As notation, for any schema σ, we define the extended schema $\#(\sigma) = \{R^{(k+1)}_{\#} \mid R^{(k)} \in \sigma\}$.

[10] This usage is illustrated by Example 5.

First, $P^{\uparrow}$ transforms the input instance I over σ_1 to its *component-extended version* over $\#(\sigma_1)$, denoted $\#(I)$: each original input fact is tagged at the front with the "identifier" of the component it belongs to.[11] However, we have no choice-mechanism that allows us to pick just one value for this identifier; hence, each fact is tagged with *all* values occurring in its component. To illustrate, if $I = \{R(a), R(c), S(a,b), T(\,)\}$, having two components $\{R(a), S(a,b), T(\,)\}$ and $\{R(c), T(\,)\}$, then $P^{\uparrow}$ produces $\#(I) = \{R_{\#}(a,a), R_{\#}(b,a), R_{\#}(c,c), S_{\#}(a,a,b), S_{\#}(b,a,b), T_{\#}(a), T_{\#}(b), T_{\#}(c)\}$.

Next, letting $\boldsymbol{A}$ be a variable not yet used in P, the program $P^{\#}$ is obtained from P by changing each atom $R(\bar{u})$ (including head atoms) to $R_{\#}(\boldsymbol{A}, \bar{u})$. Note that $P^{\#}$ is over $\#(sch(P))$. The presence of variable $\boldsymbol{A}$ guarantees that all rules in $P^{\#}$ are connected. Moreover, satisfying valuations now only use sets of facts whose first value is the same, i.e., the facts share the same component-identifier. Hence, when we execute $P^{\#}$ over $\#(I)$, the computation proceeds in a per-component fashion. Because the original program P distributes over components, program $P^{\#}$ correctly simulates P when the input contains values (see below for more discussion). To obtain output over σ_2, the third program $P^{\downarrow}$ projects the relations of $\#(\sigma_2)$ back to σ_2.

The subprogram $(P^{\uparrow} \cup P^{\#} \cup P^{\downarrow})$ will only do something if there are values in the input: at the very least, variable $\boldsymbol{A}$ needs to be assigned a value when evaluating $P^{\#}$. But even if $adom(I) = \emptyset$, in which case there is only one component, the original program P could still do useful boolean operations (e.g., as in Example 5). This computation is preserved by program P^{null}, that contains only the rules of P without variables, after extending the output rules with an additional negative body atom $\neg values(\,)$, where *values* is a value-detecting nullary relation outside $sch(P)$. The atom $\neg values(\,)$ acts as a guard, so the output rules will not fire when there are values. ◀

▶ **Example 8.** We illustrate the construction used in the proof of Proposition 7. Consider the following Datalog$^{\neg}$ program P with $edb(P) = \{R^{(1)}, S^{(1)}, T^{(2)}\}$ and $idb(P) = \{U^{(2)}, V^{(2)}\}$:

$$U(x,y) \leftarrow R(x), S(y).$$
$$V(x,y) \leftarrow U(x,y), T(x,y).$$

Assuming that V is the output relation, although the first rule of P is not connected, P distributes over components because of the join with input relation T. The transformed version of P is $\alpha(P) = P^{\uparrow} \cup P^{\#} \cup P^{\downarrow} \cup P^{\text{null}}$. Note that $P^{\text{null}} = \emptyset$ as P contains no rules without variables. Next, the program $P^{\uparrow}$ that tags input facts with their component values, contains the following rules, where '*con*' is an auxiliary relation to detect which values are connected:

$$con(x,x) \leftarrow R(x). \qquad R_{\#}(\boldsymbol{A},x) \leftarrow R(x), con(x,\boldsymbol{A}).$$
$$con(x,x) \leftarrow S(x). \qquad S_{\#}(\boldsymbol{A},x) \leftarrow S(x), con(x,\boldsymbol{A}).$$
$$con(x,y) \leftarrow T(x,y). \qquad T_{\#}(\boldsymbol{A},x,y) \leftarrow T(x,y), con(x,\boldsymbol{A}).$$
$$con(x,y) \leftarrow con(y,x).$$
$$con(x,y) \leftarrow con(x,z), con(z,y).$$

Next, program $P^{\#}$ is as follows:

$$U_{\#}(\boldsymbol{A},x,y) \leftarrow R_{\#}(\boldsymbol{A},x), S_{\#}(\boldsymbol{A},y).$$
$$V_{\#}(\boldsymbol{A},x,y) \leftarrow U_{\#}(\boldsymbol{A},x,y), T_{\#}(\boldsymbol{A},x,y).$$

[11] Nullary facts are tagged with all identifiers, since by definition they belong to all components.

Lastly, to project output back to V, program $P^{\downarrow}$ contains the rule: $V(x, y) \leftarrow V_{\#}(\boldsymbol{A}, x, y)$.

Intuitively, whenever the rule for relation $U_{\#}$ combines a fact $R_{\#}(c, a)$ and a fact $S_{\#}(c, b)$ where $a \neq b$, the shared tag c implies (through program $P^{\uparrow}$) that there is some T-fact connecting values a and b in the input, i.e., $R(a)$ and $S(b)$ belong to the same component. So, the rule for relation $U_{\#}$ works "inside" components, considering fewer pairs of R-facts and S-facts compared to the original rule for relation U. But, since P distributes over components (assuming output relation V), the output of $\alpha(P)$ is the same as P for all inputs. ◀

Let Datalog$^{\neg s}$ and con-Datalog$^{\neg s}$ denote the classes of queries computable by respectively stratified Datalog$^{\neg}$ programs and stratified con-Datalog$^{\neg}$ programs. By combining Proposition 6 and Proposition 7, we may write:

▶ **Theorem 9.** *Datalog*$^{\neg s} \cap \boldsymbol{\mathcal{D}} =$ *con-Datalog*$^{\neg s}$.

5 Connected Well-founded Datalog

In the following, we extend our results on class $\boldsymbol{\mathcal{D}}$ and stratified Datalog$^{\neg}$ to the well-founded semantics. The proofs for the well-founded semantics build upon the results for the stratified semantics by constructing, for each Datalog$^{\neg}$ program P and an input instance I, a stratified Datalog$^{\neg}$ program that simulates the well-founded semantics of P for the specific instance I. We start with the following result:

▶ **Proposition 10.** Every query computable by a con-Datalog$^{\neg}$ program under the well-founded semantics distributes over components.

Proof (sketch). Let $\mathcal{Q}$ be a query computable by a con-Datalog$^{\neg}$ program P under the well-founded semantics. Let I be an input for $\mathcal{Q}$. We have to show $\mathcal{Q}(I) = \bigcup_{J \in co(I)} \mathcal{Q}(J)$. We first transform P to a *stratified* con-Datalog$^{\neg}$ program $u_k(P)$, where each successive stratum simulates an outer step in the alternating fixpoint computation of P, where k indicates that $2k$ steps are simulated in total.[12] This technique is inspired by the *doubled program* construction [15]. Although only a constant number of steps can be simulated this way, for the specific instance I, we can choose k sufficiently large so that $u_k(P)$ simulates $P_{\mathrm{t}}(I)$ and $P_{\mathrm{t}}(J)$ for each $J \in co(I)$. Letting σ denote the output schema of $\mathcal{Q}$, we have $u_k(P)(I)|_{\sigma} = \mathcal{Q}(I)$ and $u_k(P)(J)|_{\sigma} = \mathcal{Q}(J)$ for each $J \in co(I)$. Next, because $u_k(P)$ is a stratified con-Datalog$^{\neg}$ program, we can apply Proposition 6 to know $u_k(P)(I)|_{\sigma} = \bigcup_{J \in co(I)} u_k(P)(J)|_{\sigma}$, resulting in $\mathcal{Q}(I) = \bigcup_{J \in co(I)} \mathcal{Q}(J)$, as desired. ◀

Now, Proposition 10 combined with the inclusion $\boldsymbol{\mathcal{D}} \cap \boldsymbol{\mathcal{V}} \subseteq \boldsymbol{\mathcal{M}}_{\text{disjoint}}$ from Section 3.2, gives rise to the following corollary:

▶ **Corollary 11.** *Every query computable by a con-Datalog*$^{\neg}$ *program under the well-founded semantics, and being value-driven, is domain-disjoint-monotone.*

Corollary 11 can be used to obtain an alternative proof for one of the main results in previous work by Zinn et al. [19], namely, that the win-move query (Example 1) is in the class $\boldsymbol{\mathcal{F}}_2$, as we now explain. First, $\boldsymbol{\mathcal{F}}_2$ is the class of queries that can be computed in a coordination-free manner under so-called *domain-guided distribution policies*: here, nodes of a network are made responsible for values of **dom**, and each input fact $\boldsymbol{f}$ is distributed to all those nodes responsible for at least one value of $adom(\boldsymbol{f})$. Coordination-freeness means that

[12] The 'u' in $u_k(P)$ stands for *unrolling*.

for any input, there exists a domain-guided distribution policy under which the nodes do not have to share input facts in order to compute the query. Now, since the win-move Datalog$^\neg$ program is connected, and this program is value-driven, Corollary 11 gives us that win-move is in $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$. Further applying the result $\boldsymbol{\mathcal{M}}_{\text{disjoint}} = \boldsymbol{\mathcal{F}}_2$ [4], we obtain that win-move is in $\boldsymbol{\mathcal{F}}_2$.

We also have the converse result of Proposition 10:

▶ **Proposition 12.** Every query computable by a Datalog$^\neg$ program under the well-founded semantics, and distributing over components, can be computed by a con-Datalog$^\neg$ program under the well-founded semantics.

Proof (sketch). Let $\mathcal{Q}$ be a query computable by a Datalog$^\neg$ program P under the well-founded semantics. Let $\alpha(P) = P^\uparrow \cup P^\# \cup P^\downarrow \cup P^{\text{null}}$ be the con-Datalog$^\neg$ program as defined in the proof for Proposition 7. If P is not stratified then $P^\#$, and by extension $\alpha(P)$, is also not stratified. We now outline the main arguments to demonstrate that $\alpha(P)$ computes the query $\mathcal{Q}$ under the well-founded semantics. Let I be an input for $\mathcal{Q}$. If I contains no values then the output of $\alpha(P)$ on I is just the output of P^{null} on I, and P^{null} correctly simulates P on such inputs. We also sketch the main steps for the case that I contains values. Let σ denote the output schema of $\mathcal{Q}$. First, because P computes $\mathcal{Q}$ under the well-founded semantics, we have $\mathcal{Q}(I) = P_{\text{t}}(I)|_\sigma$. Next, as in the proof of Proposition 10, we convert P to a stratified Datalog$^\neg$ program $u_k(P)$, with sufficiently large $k \in \mathbb{N}$ such that: $\mathcal{Q}(I) = u_k(P)\,(I)|_\sigma$. Now, considering the transformation $\bar{\alpha}(D) = D^\uparrow \cup D^\# \cup D^\downarrow$ for any Datalog$^\neg$ program D, it can be shown that $\bar{\alpha}$ may be applied to the right hand side, to obtain: $\mathcal{Q}(I) = \bar{\alpha}(u_k(P))\,(I)|_\sigma$. Subsequently, we can use a technical lemma to know that operations u_k and $\bar{\alpha}$ commute, i.e., $\bar{\alpha}(u_k(P))\,(I)|_\sigma = u_k(\bar{\alpha}(P))\,(I)|_\sigma$. We can up front also choose k large enough to correctly simulate the well-founded semantics of $\bar{\alpha}(P)$ on I, so that $u_k(\bar{\alpha}(P))\,(I)|_\sigma = \bar{\alpha}(P)_{\text{t}}\,(I)|_\sigma$. Finally, since P^{null} is constructed to output nothing when $adom(I) \neq \emptyset$, we have $\mathcal{Q}(I) = \bar{\alpha}(P)_{\text{t}}\,(I)|_\sigma = \alpha(P)_{\text{t}}\,(I)|_\sigma$, as desired. ◀

Let Datalog$^{\neg wf}$ and con-Datalog$^{\neg wf}$ denote the classes of queries computable under the well-founded semantics by respectively Datalog$^\neg$ programs and con-Datalog$^\neg$ programs. By combining Proposition 10 and Proposition 12, we may write:

▶ **Theorem 13.** *Datalog*$^{\neg wf} \cap \boldsymbol{\mathcal{D}} =$ *con-Datalog*$^{\neg wf}$.

6 Semi-connected Well-founded Datalog

Previous work has considered a relaxation of connected Datalog$^\neg$, called *semi-connected* [4]. We refer to Section 4.1 for the definition of connected Datalog$^\neg$, including the notions of *global* and *value-detecting* nullary relations. Now, we say that a Datalog$^\neg$ program P is *semi-connected* if we can partition the rules of P into two subprograms P_1 and P_2 such that:

1. P_1 is a con-Datalog$^\neg$ program;[13]
2. P_2 is a semi-positive program satisfying the following conditions: *(i)* $idb(P_2) \cap sch(P_1) = \emptyset$, and *(ii)* nullary relations occurring in rule bodies of P_2 are either global or value-detecting within the entire program P.

Note that the entire schema of P_1 can be used as input for P_2. So, P_2 can negate relations of $idb(P_1)$, as demonstrated by Example 15. Subprogram P_1 or P_2 could be empty. Subprogram

[13] Note that this includes restrictions on nullary atoms in rule bodies.

P_1 is not necessarily stratified, but we may view P_2 as a last computation step of P that possibly uses non-connected rules. If P is stratified then we may view P_2 as the last stratum. We denote the language of semi-connected Datalog$^\neg$ programs as semicon-Datalog$^\neg$.

Recall the query classes $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$ and $\boldsymbol{\mathcal{V}}$ from Section 3.2. Queries of $\boldsymbol{\mathcal{V}}$ computable by stratified semicon-Datalog$^\neg$ programs are in $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$ [4]. We can now confirm that this result is maintained under the well-founded semantics:

▶ **Theorem 14.** *Every query computable by a semicon-Datalog$^\neg$ program under the well-founded semantics, and being value-driven, is in $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$.*

Proof (sketch). Let $\mathcal{Q}$ be a query that is computed by a semicon-Datalog$^\neg$ program P under the well-founded semantics, and being value-driven. Let I and J be two inputs for $\mathcal{Q}$ such that J contains no nullary facts and $adom(I) \cap adom(J) = \emptyset$. We show that $\mathcal{Q}(I) \subseteq \mathcal{Q}(I \cup J)$. Following the proof idea for Proposition 10, we convert P to a stratified program $u_k(P)$ with k sufficiently large to correctly simulate the alternating fixpoint computation of P on the instances I and $I \cup J$. Importantly, if P is semi-connected then $u_k(P)$ is also semi-connected. Letting σ be the output schema of $\mathcal{Q}$, it can be shown that $u_k(P)(I)|_\sigma \subseteq u_k(P)(I \cup J)|_\sigma$, using similar techniques as in previous work [4]. Next, because $u_k(P)$ correctly simulates the well-founded semantics of P on instances I and $I \cup J$, and $\mathcal{Q}$ is computed by P, we obtain $\mathcal{Q}(I) \subseteq \mathcal{Q}(I \cup J)$, as desired. ◀

We illustrate the use of Theorem 14 with the following example.

▶ **Example 15.** Building upon the win-move example (Example 1), the following program outputs *true* (in nullary relation T) if there are at least two nodes at which player 1 has a winning strategy:

$$win(x) \leftarrow move(x, y), \neg win(y).$$
$$same(x, x) \leftarrow move(x, y).$$
$$same(y, y) \leftarrow move(x, y).$$
$$T() \leftarrow win(x), win(y), \neg same(x, y).$$

Note that negation on relation *same* simulates nonequality. This program is not stratified due to the embedding of the win-move program; so, we apply the well-founded semantics. Moreover, this program is not connected due to the last rule. But, the program is still semi-connected. It is also value-driven. Hence we can apply Theorem 14 to know that the query expressed by this program is in $\boldsymbol{\mathcal{M}}_{\text{disjoint}}$. Next, using $\boldsymbol{\mathcal{M}}_{\text{disjoint}} = \boldsymbol{\mathcal{F}}_2$ [4], we know this query can be computed in a coordination-free manner under domain-guided distribution policies. ◀

Relating to the class $\boldsymbol{\mathcal{D}}$, the following simple example shows that not all queries computable by semicon-Datalog$^\neg$ programs under the well-founded semantics distribute over components:[14]

▶ **Example 16.** Consider the following semicon-Datalog$^\neg$ program P, with $edb(P) = \{R^{(1)}\}$ and $idb(P) = \{S^{(1)}, T^{(2)}\}$, where T is the intended output relation:

$$S(x) \leftarrow R(x).$$
$$T(x, y) \leftarrow S(x), S(y).$$

[14] The query of Example 15 also does not distribute over components, e.g., on an input consisting of two disjoint *move*-subgraphs, in each of which there is precisely one winning node.

The first rule is connected, whereas the second rule is not. For the input $I = \{R(a), R(b)\}$, the output of P on I under the well-founded semantics (or stratified semantics) is $\{T(a,a), T(b,b), T(a,b), T(b,a)\}$. The facts $T(a,b)$ and $T(b,a)$, however, can not be computed when we distribute P over the components $\{R(a)\}$ and $\{R(b)\}$. ◀

7 Discussion

In this paper, we have shown that although membership of positive Datalog programs in $\boldsymbol{\mathcal{D}}$ is undecidable, connected Datalog$^\neg$ provides an effective syntax for Datalog$^\neg \cap \boldsymbol{\mathcal{D}}$ both under the stratified as well as under the well-founded semantics. In addition, the latter result provides an alternative explanation for why the non-monotonic win-move query is in $\boldsymbol{\mathcal{F}}_2$.

In theory, any query in $\boldsymbol{\mathcal{D}}$ (and therefore any query in connected Datalog$^\neg$) can be evaluated without any communication over a network using a distribution where every computing node is assigned, as a local instance, a union of connected components of the global database instance (and every connected component is assigned to at least one computing node). However, as finding connected components is expensive, it is unlikely that there are many datasets for which such a distribution of data is practical. Still, it would be interesting to investigate properties of Datalog$^\neg$ programs that imply distributions of data that give rise to communication-free evaluation.

Hull and Yoshikawa [13] introduced a declarative formalism in the style of stratified Datalog$^\neg$ in the context of object databases. Using their formalism, Cabibbo [6] showed, among other things, that semi-positive Datalog$^\neg$ extended with value invention captures the class of all queries preserved under extensions. The latter type of result can be seen as evidence that semi-positive Datalog$^\neg$ is a core fragment of Datalog$^\neg$ for the class of queries preserved under extensions. In analogy, we expect that con-Datalog$^\neg$ is somehow the right Datalog$^\neg$ fragment for $\boldsymbol{\mathcal{D}}$ and conjecture that con-Datalog$^\neg$ extended with value invention captures the class $\boldsymbol{\mathcal{D}}$.

References

1 S. Abiteboul, Z. Abrams, S. Haar, and T. Milo. Diagnosis of asynchronous discrete event systems: Datalog to the rescue! In *PODS*, pages 358–367. ACM Press, 2005.

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

3 P. Alvaro, N. Conway, J.M. Hellerstein, and D. Maier. Blazes: Coordination analysis for distributed programs. In *IEEE 30th International Conference on Data Engineering*, pages 52–63. IEEE, 2014.

4 T.J. Ameloot, B. Ketsman, F. Neven, and D. Zinn. Weaker forms of monotonicity for declarative networking: A more fine-grained answer to the CALM-conjecture. In *PODS*, pages 64–75. ACM Press, 2014.

5 T.J. Ameloot, F. Neven, and J. Van den Bussche. Relational transducers for declarative networking. *J. ACM*, 60(2):15:1–15:38, 2013.

6 L. Cabibbo. The expressive power of stratified logic programs with value invention. *Information and Computation*, 147(1):22–56, 1998.

7 K.J. Compton. Some useful preservation theorems. *Journal of Symbolic Logic*, 48:427–440, 1983.

8 N. Conway, W.R. Marczak, P. Alvaro, J.M. Hellerstein, and D. Maier. Logic and lattices for distributed programming. In *Proceedings of the Third ACM Symposium on Cloud Computing*, pages 1:1–1:14. ACM Press, 2012.

9 A. Dawar and S. Kreutzer. On Datalog vs. LFP. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 160–171. Springer, 2008.

10 T. Feder and M.Y. Vardi. Homomorphism closed vs. existential positive. In *LICS*, pages 311–320. IEEE Computer Society, 2003.

11 I. Guessarian. Deciding boundedness for uniformly connected datalog programs. In S. Abiteboul and P.C. Kanellakis, editors, *ICDT*, volume 470 of *Lecture Notes in Computer Science*, pages 395–405. Springer, 1990.

12 J.M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010.

13 R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In *VLDB*, pages 455–468. Morgan Kaufmann Publishers Inc., 1990.

14 T. Jim and D. Suciu. Dynamically distributed query evaluation. In *PODS*, pages 28–39. ACM Press, 2001.

15 D.B. Kemp, D. Srivastava, and P.J. Stuckey. Bottom-up evaluation and query optimization of well-founded models. *Theor. Comput. Sci.*, 146(1&2):145–184, 1995.

16 B.T. Loo, T. Condie, M. Garofalakis, D.E. Gay, J.M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: Language, execution and optimization. In *SIGMOD*, pages 97–108. ACM Press, 2006.

17 O. Shmueli. Equivalence of Datalog queries is undecidable. *The Journal of Logic Programming*, 15(3):231–241, 1993.

18 A. Van Gelder. The alternating fixpoint of logic programs with negation. *J. Comput. Syst. Sci.*, 47(1):185–221, 1993.

19 D. Zinn, T.J. Green, and B. Ludäscher. Win-move is coordination-free (sometimes). In *ICDT*, pages 99–113. ACM Press, 2012.

Distributed Streaming with Finite Memory

Frank Neven[1], Nicole Schweikardt[2], Frédéric Servais[1], and Tony Tan[1]

1 **Hasselt University and Transnational University of Limburg**
2 **Humbold-University Berlin**

Abstract

We introduce three formal models of distributed systems for query evaluation on massive databases: Distributed Streaming with Register Automata (DSAs), Distributed Streaming with Register Transducers (DSTs), and Distributed Streaming with Register Transducers and Joins (DSTJs). These models are based on the key-value paradigm where the input is transformed into a dataset of key-value pairs, and on each key a local computation is performed on the values associated with that key resulting in another set of key-value pairs. Computation proceeds in a constant number of rounds, where the result of the last round is the input to the next round, and transformation to key-value pairs is required to be generic. The difference between the three models is in the local computation part. In DSAs it is limited to making one pass over its input using a register automaton, while in DSTs it can make two passes: in the first pass it uses a finite-state automaton and in the second it uses a register transducer. The third model DSTJs is an extension of DSTs, where local computations are capable of constructing the Cartesian product of two sets. We obtain the following results: (1) DSAs can evaluate first-order queries over bounded degree databases; (2) DSTs can evaluate semijoin algebra queries over arbitrary databases; (3) DSTJs can evaluate the whole relational algebra over arbitrary databases; (4) DSTJs are strictly stronger than DSTs, which in turn, are strictly stronger than DSAs; (5) within DSAs, DSTs and DSTJs there is a strict hierarchy w.r.t. the number of rounds.

1998 ACM Subject Classification C.2.4 Distributed Systems, H.2.4 Systems, H.2.6 Database Machines

Keywords and phrases Distributed systems, relational algebra, semijoin algebra, register automata, register transducers

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.324

1 Introduction

Recent years have seen a massive growth in parallel and distributed computations based on the key-value paradigm. This was fostered by the emergence of popular systems such as Hadoop [31] and Spark [25], which support this paradigm, as well as by many specialised systems built on top of them such as Hive [29], Pig [15], Shark [32], etc.

In brief, the key-value paradigm works as follows. An input dataset D is first transformed into another dataset D' of key-value pairs which is then distributed across a cluster of machines, where values with the same key are sent to the same server. The main computation is performed on D', where values in different servers can be processed in parallel. Take, for example, the Pig Latin[1] script below for computing the query $A(x, y) \wedge \neg B(y)$:

[1] See [24, 23, 15] and the references therein for more details about Pig Latin and the Pig system.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 324–341

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
1.  A  =  load 'A.txt' as (x,y);
2.  B  =  load 'B.txt' as (y);
3.  C  =  cogroup A by y, B by y;
4.  D  =  filter C by IsEmpty(B);
5.  E  =  foreach D generate flatten(A);  // E is the result of A(x,y) ∧ ¬B(y)
```

In brief, the Pig system converts this script into a Hadoop's MapReduce program that does the following. The mapper maps each tuple $A(a, b)$ into a key-value pair $(\mathsf{key} = b, \mathsf{val} = A(a, b))$; and each tuple $B(b)$ into $(\mathsf{key} = b, \mathsf{val} = B(b))$. This is done in the script's step 3. For each key c, it checks whether there is an A-tuple and a B-tuple. It collects only those keys in which there is A-tuple, but no B-tuple. This is done in step 4. It will then store only the A-tuples from the collected keys. This is done in step 5.

This example highlights one of the most appealing features of the key-value paradigm: ease of parallelisation. Since computations for different keys are independent, they can be computed in parallel by assigning each key to a server that is responsible for its computation. Typically one server can be assigned with many keys, and in Hadoop such assignments are done using random hash function by default.

We are aware that the key-value paradigm is often called the map-reduce paradigm, and rightly so. However, in many systems communities the name map-reduce refers to Hadoop's MapReduce and excludes Spark, even though Spark does support map-reduce like computations. The difference between map-reduce in Hadoop and Spark lies in, among many other aspects, the implementation of fault tolerance and data storage [25, 33]. Since our focus is on the theory, and to avoid confusion, we opt for the name key-value paradigm.

Many algorithms and systems have been built based on the key-value paradigm. We will discuss some of them in the related work section at the end of Section 1. In the database setting, SQL-queries are *the* standard class of queries. Recently, systems such as Pig, Hive, and Shark have been built to support SQL-like queries on massive datasets, and have been widely used in both academia and industry. However, still lacking is a detailed study of their theoretical foundations.

In this paper we aim to contribute to filling this gap. Our goal is to determine computing mechanisms that are necessary and sufficient for evaluating relational algebra, which is the foundation of the SQL query language. To this end, we introduce three models for distributed computations based on the key-value paradigm and compare their expressiveness with relational algebra: *Distributed Streaming with register Automata* (DSAs), *Distributed Streaming with register Transducers* (DSTs), and *Distributed Streaming with register Transducers and Joins* (DSTJs). In introducing new models, we must be aware that systems like Pig, Hive, or Shark are fully automated in the sense that an input query is automatically converted into a program in Hadoop (for Pig and Hive) or Spark (in the case of Shark). The models must be simple enough to allow for such automation, while still being strong enough to capture a useful class of queries (in our case, relational algebra or suitable fragments thereof).

Brief description of our models. To avoid clutter, we start by defining our models for Boolean queries over directed finite graphs, which is the simplest form of database.

For Boolean queries each model consists of three components: (1) a *mapper* that maps each element in the input to a bag of key-value pairs; (2) a *reducer* that computes for each key separately on the bag of values associated to that key and outputs a bag of values; (3) an *aggregator* that determines the final output yes/no from a bag of values. They can perform multiple rounds of computation, where the output of the reducer is passed as input to the next mapper. The aggregator is consequently only applied at the very end to determine the

result. For non-Boolean queries, we discard the aggregator, and set the output of the last reducer as the output of the computation.[2]

The difference between DSAs/DSTs/DSTJs and the general key-value paradigm lies on the specific, concrete models of computations assigned to the map, reduce, and aggregator functions. In fact, the models assigned are very simple as we will briefly explain below.

In DSAs the mappers are *generic* functions that map *deterministically* a tuple to a bag of key-value pairs based on the equality type of the input tuple. They are essentially functions that neither can invent values nor interpret values, except for the equality test among the data values. The reducers and aggregators in DSAs are *commutative*[3] finite memory automata [17], also called register automata [22]. These are finite automata extended with a fixed number of registers where each register can hold a data value. The automata change states depending on the current state and equality tests among the values currently stored in the registers and those in the input tuple. In the reduce phase, the input values are fed to the automaton one by one (hence, the name "streaming"). After having read the last input item, it outputs a finite bag of values of constant size depending on the final configuration. The automaton from the aggregator component is used to pass through the output of the last reduce phase to determine the end result.

Note that the computation performed by a DSA's mapper, reducer, or aggregator process the input only once while using at most logarithmic space. Furthermore, the number of elements in the output of reducers within the DSA model does not depend on the length of its input but only on the reducer itself. Hence, DSAs are rather limited as they need to summarise an input stream by a fixed number of output values. In particular, DSAs cannot transform a stream of values into another stream of values. To allow for this, we introduce the second model, DSTs. DSTs use the same mappers and aggregators as DSAs, but it has available more powerful reducers. In DSTs, a reducer makes two passes over the input: in the first pass it uses a commutative finite state automaton to gather some finite information on the input, and in the second pass it uses a commutative *register transducer* that for each input value outputs a bag of values. A register transducer works essentially like a register automaton. It has a fixed number of registers where each register can hold a data value. Depending on the current state and the equality tests among the values in the registers and in the input tuple, it can change its state and at the same time output a bag of values. Hence, a register transducer can transform a stream of values into another stream of values.

Note that it seems very unlikely that DSAs or DSTs can compute Boolean queries that involve join operations, where there can be a quadratic blow-up in the size of intermediate results. In fact, as we will show later, both DSAs and DSTs cannot detect the existence of a triangle in a given graph, and hence, cannot perform join operations. This motivates us to introduce the third model called DSTJs. Again, the only difference between DSTJs and DSTs lies on the reducers. In DSTJs, the reducers can be of two types: a register transducer (as used by DSTs), or an abstract function that performs a Cartesian product between two subsets of the input values; the latter is a natural abstraction of the `join` transformation supported by the RDD data structure in Spark [26, 25, 33]. By definition, DSTJs hence can perform join operations. We will show later that DSTJs can evaluate the whole class of relational algebra queries.

---

[2] We note that although the aggregator component is not common in the key-value paradigm, it does exist. See, for example, the system Bagel [8]. For Boolean queries such as *"Are there at least 1000 triangles?"*, it is more convenient and efficient to add an aggregator component that aggregates all the output, rather than adding an extra round to simulate the aggregator.

[3] Commutativity is necessary to ensure that the output is independent of the order in which the input tuples are processed.

Main results. The main results in this paper are the following:

1. DSTJs are strictly stronger than DSTs, which are strictly stronger than DSAs; and within each of the 3 models there is a strict expressiveness hierarchy w.r.t. the number of rounds;
2. neither DSAs nor DSTs can detect the presence of a triangle in a graph (hence, neither DSAs nor DSTs can do joins);
3. when restricting attention to bounded degree databases, DSAs can evaluate relational algebra (and, even more, first-order sentences with modulo counting quantifiers)
4. over arbitrary databases, DSTs can evaluate the semijoin algebra while DSAs can not;
5. over arbitrary databases, DSTJs can evaluate the relational algebra while DSTs can not.

The relations among DSAs, DSTs and DSTJs with the classical database queries are illustrated as follows.[4]

DSA	$\subsetneq$	DST	$\subsetneq$	DSTJ
$\cup\mid$		$\cup\mid$		$\cup\mid$
FO (= rel. alg.) over bounded degree databases		semijoin algebra		relational algebra

These results emphasise that, albeit simple, DSAs, DSTs, and DSTJs are pretty expressive. In fact, they also highlight that the power of the key-value paradigm here lies within the ability to group values according to a common key.

Related Work. The key-value paradigm, or map-reduce paradigm, attracted a lot of attention since its inception into Google in the mid 2000s [13, 14]. Arguably it can be viewed as a subclass of the BSP model introduced by Valiant back in 1990 [30], in which the keys play a special role in determining the distribution of the data. We discuss the work most related to the setting of the present paper. We are aware of [5, 1, 4, 2, 9, 10, 11, 19, 20, 21, 27, 28]. Karloff et al. [18] introduce a rigorous computation model for the MapReduce programming paradigm where (randomised) mappers and reducers are implemented by a RAM with sublinear space and polynomial time. It is typical that in the map-reduce computation the reducers considered so far in the literature, such as [2, 4, 27], while limited in the number of data it can access, can be arbitrarily strong, typically polynomial time machines in the number of original input items. This is obviously orthogonal with our models here, where the power of the reducers are limited.

Map-reduce as a framework for the evaluation of special classes of queries, especially the join queries, has been considered by a number of articles. However, it is not that clear how to extend them to full relational algebra. We mention here some of the work along this line. Afrati and Ullman [5] study the evaluation of join queries and take the amount of communication, calculated as the sum of the sizes of the input to reducers, as a complexity measure. Evaluation of transitive closure and datalog queries in MapReduce has been investigated in [1, 6]. Afrati et al. [4] study the tradeoff between parallelism and communication cost in a map-reduce setting. In particular, the authors established lower and upper bounds on communication costs for a number of typical problems in databases. All the lower bounds are established only for one round computation.

Most of the existing MapReduce algorithms assume the number of keys generated is bounded by a constant, equating the number of keys with the number of available servers.

[4] It is a classic result by Codd [12] that first-order logic and relational algebra are equivalent in terms of expressiveness.

See, for example, the algorithm for enumerating the triangles in [27] and arbitrary sample subgraphs in [2, 4]. This is orthogonal to our approach, where the number of generated keys can be proportional to the number of vertices in the input graph, and parallelisation can be achieved by automatically hashing the keys to the available servers. A more thorough discussion on generic mappers is provided in Section 3.

Koutris and Suciu [19] introduce the massively parallel (MP) model of computation, where computations proceed in a sequence of parallel steps, each followed by a global synchronisation of all servers. In this model, evaluation of conjunctive queries [9, 19] as well as skyline queries [3] have been considered. The MP model can be implemented in the map-reduce setting, with the hash functions fully specified. Again, the bounds, especially the lower bounds, are established mainly for one round of computation.

Another setting, but orthogonal to the MapReduce framework, is that of declarative networking where distributed computations and networking protocols are modeled and programmed using formalisms based on Datalog [7, 16].

Outline. We give a formal definition of the key-value paradigm in Section 2. In Section 3 we present the notion of generic mappers. Then, in Sections 4–6 we provide the formal definitions of DSAs, DSTs, and DSTJs, respectively, and study their expressiveness. In Section 7 we establish the relations between our DSA/DST/DSTJ models and the classical semijoin algebra and relational algebra. We conclude in Section 8.

2 The key-value paradigm

We start by introducing some notations. Let $\mathbb{N}$ be the set of natural numbers $\{1, 2, \ldots\}$. For $m \in \mathbb{N}$, we let $[m] = \{1, \ldots, m\}$. Let S and T be sets. We write $\mathrm{Pow}(S)$ or 2^S to denote the set of all finite subsets of S, and we write $\mathcal{P}(S, T)$ and $\mathcal{F}(S, T)$ to denote the class of all partial functions and all functions from S to T, respectively. We write $\mathrm{Bags}(S)$ to denote the set of all finite *bags* over S (i.e., all finite multisets built from elements in S). Instead of $B \in \mathrm{Bags}(S)$, we sometimes write $B \sqsubseteq S$. We write $\boldsymbol{\chi}_B$ to denote the characteristic function of the bag B. That is, for every $x \in S$, $\boldsymbol{\chi}_B(x)$ returns the multiplicity of x in B. We say that A is a *subbag* of B, if $\boldsymbol{\chi}_A(x) \leq \boldsymbol{\chi}_B(x)$, for every $x \in S$.

We fix an infinite set $\mathbf{D}$ of *data values*. In this paper, we are mostly concerned with (finite, directed) graphs $G = (V, E)$, where $V \subseteq \mathbf{D}$ and $E \subseteq V \times V$. Such graphs are always presented in the form of a sequence of pairs $(a_1, b_1), \ldots, (a_n, b_n)$ where each pair (a_i, b_i) indicates that there is an edge from vertex a_i to vertex b_i. In view of this, elements of $\mathbf{D}$ will also be called vertices, or nodes. We use the words vertex and node interchangeably, and we write $V(G)$ and $E(G)$ to denote G's set of vertices and edges, respectively. We refer to Section 7 for a generalisation to relations of higher arity, where the input is a stream of facts of the form $R(a_1, \ldots, a_m)$, where R is an arbitrary relation symbol.

We assume we are given two sets $\mathbf{K}$ and $\mathbf{V}$ denoting the domain of *keys* and *values*, respectively. We call an element $(\mathsf{key}, \mathsf{val}) \in \mathbf{K} \times \mathbf{V}$ a *key-value pair* and an element $(\mathsf{key}, B) \in \mathbf{K} \times \mathrm{Bags}(\mathbf{V})$ a *key-bag-value pair*. To differentiate them from the standard tuple, we will write $\langle \mathsf{key} : \mathsf{val} \rangle$ and $\langle \mathsf{key} : B \rangle$ to denote key-value and key-bag-value pairs, respectively.

A *key-value paradigm* (KVP) instance is a tuple $\mathcal{M} = (map_1, red_1, \ldots, map_\ell, red_\ell, agg)$ where $\ell \in \mathbb{N}$. We say that $\mathcal{M}$ has ℓ *rounds*. The components of $\mathcal{M}$ are defined as follows:

- map_1 is an *initial mapper* which maps an edge $e \in \mathbf{D} \times \mathbf{D}$ to a finite bag over $\mathbf{K} \times \mathbf{V}$;
- for each $i \geq 2$, map_i is a *mapper* which maps a value in $\mathbf{V}$ to a finite bag over $\mathbf{K} \times \mathbf{V}$;

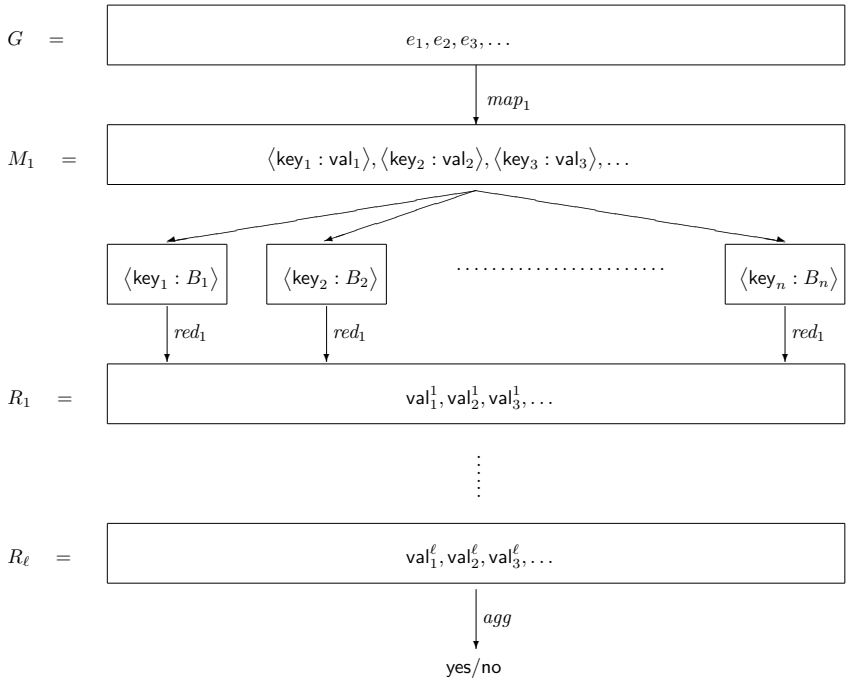


Figure 1 The flow of computation in an ℓ round key-value paradigm computation.

- for each $i \in [\ell]$, red_i is a *reducer* which maps a key $\mathsf{key} \in \mathbf{K}$ and finite bag $B \sqsubseteq \mathbf{V}$ of values to a finite bag $B' \sqsubseteq \mathbf{V}$ of values; and,
- agg is an *aggregator* which determines the output of $\mathcal{M}$; agg is a function mapping a finite bag over $\mathbf{V}$ to the value yes or no.

For a bag $B \sqsubseteq \mathbf{K} \times \mathbf{V}$, define $\text{keys}(B) = \{\mathsf{key} \mid \exists\, \mathsf{val} \in \mathbf{V}, \langle \mathsf{key} : \mathsf{val} \rangle \in B\}$ as the set of keys occurring in B, and $\text{values}(\mathsf{key}, B) = \{\!\{\mathsf{val} \mid \langle \mathsf{key} : \mathsf{val} \rangle \in B\}\!\}$ as the *bag* of values occurring in B with key key. Here, we use double braces $\{\!\{...\}\!\}$ to indicate bags, i.e., if B contains i copies of tuple $\langle \mathsf{key} : \mathsf{val} \rangle$, then $\text{values}(\mathsf{key}, B)$ contains i copies of val.

On input $G = (V, E)$, the output $\mathcal{M}(G) \in \{\mathsf{yes}, \mathsf{no}\}$ is computed as follows:

- $M_1 = \bigcup_{e \in E} map_1(e)$ and $R_1 = \bigcup_{\mathsf{key} \in \text{keys}(M_1)} red_1(\mathsf{key}, \text{values}(\mathsf{key}, M_1))$.
- For each $i \in \{2, \ldots, \ell\}$,

$$M_i = \bigcup_{\mathsf{val} \in R_{i-1}} map_i(\mathsf{val}) \quad \text{and} \quad R_i = \bigcup_{\mathsf{key} \in \text{keys}(M_i)} red_i(\mathsf{key}, \text{values}(\mathsf{key}, M_i))$$

- Finally, $\mathcal{M}(G) = agg(R_\ell)$.

We write $M_i(G)$ and $R_i(G)$ to indicate that the bags M_i and R_i are obtained when the input graph is G. We say that $\mathcal{M}(G)$ is the output of $\mathcal{M}$ on input graph G.

Figure 1 illustrates the flow of computation in an ℓ-round KVP instance. As mentioned in Section 1, the models DSA/DST/DSTJ introduced in this paper follow the key-value paradigm, where the mappers are required to be generic (see Section 3), and the reducers and aggregators are specified by extensions of finite automata (see Sections 4–6).

3 Generic mappers

In this section we instantiate the key and value sets $\mathbf{K}$ and $\mathbf{V}$, and define formally the notion of generic mappers. We fix a finite alphabet Σ and a number $k \in \mathbb{N}$. We reserve $\#$ to be

a special symbol not in $\mathbf{D}$, intended to represent an empty spot or an empty register. $\mathbf{D}_\#$ denotes the set $\mathbf{D} \cup \{\#\}$. We usually write $a, b, c, \ldots$ to denote elements of $\mathbf{D}_\#$ and $\bar{a}, \bar{b}, \bar{c}, \ldots$ for elements of $\mathbf{D}_\#^k$ with $k \in \mathbb{N}$. When $\bar{a} \in \mathbf{D}_\#^k$, we tacitly assume that $\bar{a} = a_1, \ldots, a_k$.

Define $\mathbf{A}_k$ as $\Sigma \times \mathbf{D}_\#^k$. Both $\mathbf{K}$ and $\mathbf{V}$ will be interpreted as $\mathbf{A}_k$. The purpose of σ in $(\sigma, \bar{a}) \in \Sigma \times \mathbf{D}_\#^k$ is to encode a finite amount of information about the vertices in $\bar{a}$. For $t = (\sigma, \bar{a}) \in \mathbf{A}_k$, we call σ the label of t.

A $\mathbf{D}$-*bijection* is a 1-1 mapping $\pi : \mathbf{D}_\# \to \mathbf{D}_\#$, where $\pi(\#) = \#$. We extend π to tuples in the canonical way. Let R and S be finite sets and let f be a function from $R \times \mathbf{D}_\#^m$ to $\mathrm{Bags}(S \times \mathbf{D}_\#^n)$ for some $m, n \in \mathbb{N}$. The function f is *generic* if the following two conditions hold: (1) For all $(r, \bar{c}) \in R \times \mathbf{D}_\#^m$, if $(s, \bar{d}) \in f(r, \bar{c})$, then all non-# values in $\bar{d}$ are from $\bar{c}$; i.e., f cannot invent new values. (2) For every $\mathbf{D}$-bijection π, $\boldsymbol{\chi}_{f(r,\bar{c})}(s, \bar{d}) = \boldsymbol{\chi}_{f(r,\pi(\bar{c}))}(s, \pi(\bar{d}))$; i.e., f cannot interpret values in $\mathbf{D}$.

Let us briefly comment on our choice of generic mappers. In the theoretical studies of MapReduce computations, a mapper is typically a hash function, which maps the data items to the available machines; see, for example, [5, 2, 9, 10, 19, 21, 27]. This is different to our model here, where the mappers are generic functions that map a value deterministically to a set of key-value pairs. Such mappers are not uncommon. For example, the mappers generated by the Pig system [15] are essentially generic mappers similar to the ones studied in this paper; see [23, Section 4.2]. We will give a more detailed comparison between our model and the Pig system at the end of Section 7.

Obviously, the generic mappers can generate as many keys as the number of tuples in the input database. However, this does not mean that the system needs one machine for one key. In the classic example of a MapReduce program for "word count" [13], the mapper is a generic function and the number of keys produced equals the number of different words in the input text. But one would hardly insist that it requires one machine for each key. Rather, to achieve parallelisation, the system automatically hashes the keys to the available machines[5], and the processor evaluates the values for each key separately, one key at a time. Of course, specific hash functions may be desirable to achieve optimisation in some settings, say when the input datasets have been preprocessed, or when some statistics about the input are known. This is out of the scope of our paper. Our goal is to study the sufficient and necessary computation mechanism to evaluate relational algebra in a general setting, where nothing is known about the data or the available machines.

To end this section, let us describe how generic mappers can be specified. We let $[k]_\# := [k] \cup \{\#\}$. The *equality type* τ of a tuple $(d_1, \ldots, d_k)$ is the undirected graph with vertex set $[k]_\#$, where for $i, j \in [k]$ there is an edge between vertices i and j iff $d_i = d_j$, and there is an edge between vertices i and # iff $d_i = \#$. A generic mapper can be specified by a table that assigns to each *equality type* τ over $[k]_\#$ a list $p_1, \ldots, p_s$ of *patterns*, each of the form $\langle k_i : v_i \rangle$, where $k_i = (\sigma_i, j_1, \ldots, j_k)$ and $v_i = (\sigma_i', j_1', \ldots, j_k')$ with $\sigma_i, \sigma_i' \in \Sigma$ and $j_1, \ldots, j_k, j_1', \ldots, j_k' \in [k]_\#$. On input of a tuple $(\sigma, d_1, \ldots, d_k) \in \mathbf{A}_k$, the mapper then determines the equality type τ of $(d_1, \ldots, d_k)$, looks up the according patterns $p_1, \ldots, p_s$, and for each such p_i outputs the key-value pair $\langle \mathsf{key}_i : \mathsf{val}_i \rangle$ with $\mathsf{key}_i = (\sigma_i, d_{j_1}, \ldots, d_{j_k})$ and $\mathsf{val}_i = (\sigma_i', d_{j_1'}, \ldots, d_{j_k'})$, where $d_\#$ is defined to be the value #. Generic *initial* mappers are specified accordingly, where only equality types over $\{1, 2, \#\}$ for input tuples $(d_1, d_2) \in \mathbf{D} \times \mathbf{D}$ are considered.

[5] By default, the Hadoop system [31] takes a random hash function to hash the keys, which in practice works well. Theoretically this is not surprising. A standard application of Chernoff bounds guarantees that the keys are assigned to all machines uniformly (up to a small constant factor). Nevertheless, Hadoop also provides a platform for the user to specify his/her own hash functions.

4 Distributed streaming with register automata (DSA)

In this section we introduce DSAs and study their expressiveness. We start with RA-reducers and RA-aggregators, which are reducers and aggregators instantiated with register automata. Following this, we present the formal definition of DSAs, and establish their expressiveness, as well as a hierarchy on the number of rounds.

RA-reducers. We start with the notion of register transition systems, which are essentially register automata [17, 22]. Intuitively, they work as follows. The input is a sequence of elements of $\mathbf{A}_k$, and each register can hold an element of $\mathbf{D}_\#$. For every input $(\sigma, \bar{a}) \in \mathbf{A}_k$, the system changes its state depending on σ and equality tests among the vertices in $\bar{a}$ and the vertices currently stored in the registers. The formal definition reads as follows.

▶ **Definition 1.** For $r \in \mathbb{N}$, an *r-register transition system over* $\mathbf{A}_k$ is a tuple $\mathcal{S} = \langle Q, \delta \rangle$, where $r \geq k$, Q is a finite set of states, and δ is a transition function from $Q \times \Sigma \times \mathcal{F}([k], 2^{[r]})$ to $\mathcal{P}([r], [k]) \times Q$.[6]

The intuitive meaning of a transition in δ is as follows. If on input $(\sigma, \bar{a})$ the system is in state q, and the data value a_i appears in exactly the registers in $f(i)$ for each $i \in [k]$, and $\delta(q, \sigma, f) = (g, q')$, then the system can enter state q' and replace the content of each register j with $a_{g(j)}$ for each $j \in [r]$.

A *configuration of* $\mathcal{S}$ is an element of $Q \times \mathbf{D}_\#^r$. An element $(\sigma, \bar{a}) \in \mathbf{A}_k$ induces a relation $\vdash_{(\sigma,\bar{a})}$ on the configurations of $\mathcal{S}$ defined as follows: $(q, \bar{u}) \vdash_{(\sigma,\bar{a})} (q', \bar{v})$, if $\delta(q, \sigma, f) = (g, q')$ and

- $f(i) = \{j \mid u_j = a_i\}$ for each $i \in [k]$, and
- for each $i \in [r]$, if $g(i)$ is defined, then $v_i = a_{g(i)}$ and if $g(i)$ is undefined, then $v_i = u_i$.

Let $t = t_1 \cdots t_n$ be a sequence of elements of $\mathbf{A}_k$. A *run of $\mathcal{S}$ on t starting from a configuration* $(q, \bar{u})$ is a sequence $(q_0, \bar{u}_0), \ldots, (q_n, \bar{u}_n)$ of configurations, where $(q_0, \bar{u}_0) = (q, \bar{u})$ and $(q_{i-1}, \bar{u}_{i-1}) \vdash_{t_i} (q_i, \bar{u}_i)$ for each $i \in [n]$.

We now define reducers in terms of transition systems.

▶ **Definition 2.** An *RA-reducer over* $\mathbf{A}_k$ is a tuple $red = (\mathcal{S}, \rho_{in}, \rho_{out})$, where $\mathcal{S} = \langle Q, \delta \rangle$ is an r-register transition system over $\mathbf{A}_k$ and $r \geq k$; ρ_{in} is a function that maps an element of $\mathbf{A}_k$ to a configuration of $\mathcal{S}$; and, ρ_{out} is a function that maps a configuration of $\mathcal{S}$ to a finite bag over $\mathbf{A}_k$. Both ρ_{in} and ρ_{out} are required to be generic.

Intuitively, each reducer gets as input a key-bag-value pair $\langle \mathsf{key} : B \rangle$ where $\rho_{in}(\mathsf{key})$ identifies the initial configuration from which the run of $\mathcal{S}$ is started. The output then is $\rho_{out}(c)$, where c is the last configuration of the run.

Formally, let $\langle \mathsf{key} : B \rangle \in \mathbf{A}_k \times \mathrm{Bags}(\mathbf{A}_k)$ be a key-bag-value pair, and let $t_1, \ldots, t_m$ be an enumeration of the elements in B.[7] The output $red(\mathsf{key}, B)$ is defined as $\rho_{out}(q_m, \bar{u}_m)$ for the run $(q_0, \bar{u}_0), \ldots, (q_m, \bar{u}_m)$ of $\mathcal{S}$ on $t_1 t_2 \cdots t_m$ with $(q_0, \bar{u}_0) = \rho_{in}(\mathsf{key})$.

Obviously, the run of $\mathcal{S}$ on B depends on the order in which $t_1, \ldots, t_m$ are presented. However, we want to insist that the output $red(\mathsf{key}, B)$ is the same regardless of the order in which the elements in B are arranged. Therefore, we require RA-reducers to be *commutative*

[6] Note that unlike the definition of register automata in [17] and [22], in a transition system we do not specify the initial state, the final states and the initial content of the registers. We will, however, use the standard register automata to define the aggregator.

[7] Since B is a bag, some elements can appear multiple times in the enumeration.

in the following sense: If $t = t_1 \cdots t_m$ and $t' = t_{\pi(1)} \cdots t_{\pi(m)}$ are two enumerations of the elements of B (for some permutation π of $[m]$), and $(q_m, \bar{u}_m)$ and $(q'_m, \bar{u}'_m)$ are the final configurations of the runs of $\mathcal{S}$ on t and t', respectively, starting in configuration $\rho_{in}(\mathsf{key})$, then $\rho_{out}(q_m, \bar{u}_m) = \rho_{out}(q'_m, \bar{u}'_m)$.

Note that by definition of a transition system, an RA-reducer can never get stuck and always processes the complete input. The output of an RA-reducer is therefore well-defined.

RA-aggregator. An *r-register automaton over* $\mathbf{A}_k$ is an r-register transition system $\mathcal{S} = \langle Q, \delta \rangle$ together with a designated initial state q_0, a set of final states $F \subseteq Q$ and an initial content of the registers $\bar{u}_0$. We will write $\mathcal{A} = \langle Q, \delta, q_0, F, \bar{u}_0 \rangle$ to denote an r-register automaton.

The configurations of $\mathcal{A}$ and the relations $\vdash_{(\sigma,\bar{a})}$ are defined similarly as for a transition system. The only difference is that in a register automaton, we insist that the run should start from the configuration $(q_0, \bar{u}_0)$.

Formally, let $t = t_1 \cdots t_n$ be a sequence of elements of $\mathbf{A}_k$. The run $(q_0, \bar{u}_0), \cdots, (q_n, \bar{u}_n)$ of $\mathcal{A}$ on t is *accepting* (and $\mathcal{A}$ *accepts* t) iff $q_n \in F$. The automaton is *commutative* when $\mathcal{A}$ accepts $t_1 \cdots t_n$ if and only if $\mathcal{A}$ accepts $t_{\pi(1)} \cdots t_{\pi(n)}$ for every sequence $t = t_1 \cdots t_n$ of elements of $\mathbf{A}_k$ and for every permutation π on $[n]$. For commutative register automata we can safely regard the input sequence as a finite bag B, where we consider an arbitrary enumeration of the elements in B and in which context we simply say that either $\mathcal{A}$ accepts B or not.

▶ **Definition 3.** An *RA-aggregator* is a commutative r-register automaton $\mathcal{A}$ over $\mathbf{A}_k$ with $r \geq k$.

Obviously, an *RA-aggregator* $\mathcal{A}$ can be viewed as a function from finite bags of $\mathbf{A}_k$ to $\{\mathsf{yes}, \mathsf{no}\}$, where $\mathcal{A}(B) = \mathsf{yes}$, if $\mathcal{A}$ accepts B, and $\mathcal{A}(B) = \mathsf{no}$, otherwise.

Definition of DSA. An ℓ-round DSA is a tuple $\mathcal{M} = (map_1, red_1, \ldots, map_\ell, red_\ell, agg)$, where each map_i is a generic mapper, each red_i is an RA-reducer, and agg is an RA-aggregator.

We say that $\mathcal{M}$ *accepts* a graph G, if $agg(R_\ell(G)) = \mathsf{yes}$, in which case, we write $\mathcal{M}(G) = \mathsf{yes}$. Here, R_ℓ is as defined in Section 2. By $\mathcal{G}(\mathcal{M})$ we denote the class of all graphs accepted by $\mathcal{M}$, and we say that $\mathcal{G}(\mathcal{M})$ is the class of graphs *recognised* by $\mathcal{M}$.

▶ **Example 4.** Consider inputs of the form $(d_1, s_1), \ldots, (d_n, s_n)$, where each tuple (d_i, s_i) indicates that data value d_i is stored on server s_i. Let INTERSECT be the problem to decide whether there is a data value that is stored on more than one server. It can easily be formalised as a 1-round DSA $\mathcal{M} = (map_1, red_1, agg)$ over $\mathbf{A}_k$ for $k = 1$ and $\Sigma = \{\sigma_{\text{blank}}, \sigma_{\text{disj}}, \sigma_{\text{ndisj}}\}$.

The initial mapper map_1 assigns to each input tuple $(d_i, s_i) \in \mathbf{D} \times \mathbf{D}$ a single key-value pair $\langle \mathsf{key} : \mathsf{val} \rangle$ with $\mathsf{key} = (\sigma_{\text{blank}}, d_i)$ and $\mathsf{val} = (\sigma_{\text{blank}}, s_i)$. Thus, the initial mapper can be specified by a table which assigns to each equality type τ the single pattern $p = \langle k : v \rangle$ with $k = (\sigma_{\text{blank}}, 1)$ and $v = (\sigma_{\text{blank}}, 2)$.

The reducer red_1 is an RA-reducer over $\mathbf{A}_k$ (for $k = 1$), with a single register, with state set $Q = \{q_0, q_1, q_2\}$, and with $\rho_{in}(\mathsf{key}) = (q_0, \#)$ for all $\mathsf{key} \in \mathbf{A}_k$. The transition function δ ensures that when reading a symbol $(\sigma, s) \in \mathbf{A}_k$, the RA-reducer proceeds as follows: If the current state is q_0 (i.e., the automaton performs its first step), then the automaton stores the value s in its register and changes to state q_1. If the current state is q_1, and the value s is different from the value stored in the register, then the automaton changes to state q_2; otherwise (i.e., s coincides with the value stored in the register), the automaton remains in state q_1. If the current state is q_2, then the automaton simply remains in this state.

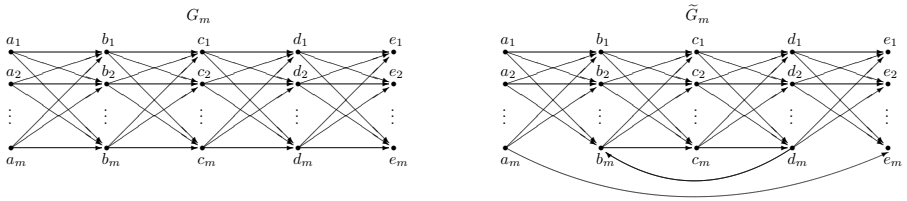


Figure 2 DSAs cannot differentiate between G_m on the left and $\widetilde{G}_m$ on the right.

The function ρ_{out} maps the final configuration (q, v) to $(\sigma_{\text{ndisj}}, \#)$ if $q = q_2$, and to $(\sigma_{\text{disj}}, \#)$ otherwise. Finally, the aggregator agg is a simple finite automaton which receives as input a list of items in $\mathbf{A}_k$ and accepts if, and only if, at least one these items is of the form $(\sigma_{\text{ndisj}}, \#)$. This completes the description of a 1-round DSA which solves the INTERSECT problem. □

Expressiveness of DSAs and a hierarchy on the number of rounds. The rest of this section is devoted to our study of the expressiveness of DSAs.

We start by showing that on general graphs DSAs cannot compute joins; in fact, they cannot even test if an input graph contains a triangle. Let TRIANGLE be the class of all graphs G that contain a directed triangle.

▶ **Theorem 5.** *There is no DSA that recognises* TRIANGLE.

Proof (sketch). Consider the graphs G_m and $\widetilde{G}_m$ depicted in Figure 2. While $\widetilde{G}_m$ contains a triangle, G_m does not. We show that for every DSA $\mathcal{M}$ there is an $m \in \mathbb{N}$ such that $\mathcal{M}$ cannot distinguish between G_m and $\widetilde{G}_m$, i.e., $\mathcal{M}(G_m) = \mathcal{M}(\widetilde{G}_m)$. The number m we choose here is bigger than the number r of registers of $\mathcal{M}$, and the proof relies on a careful analysis of the computation of $\mathcal{M}$, utilising the fact that mappers of $\mathcal{M}$ are generic and reducers of $\mathcal{M}$ are generic and commutative. Briefly, it is based on the fact that for every vertex u, its neighbourhoods in both G_m and $\tilde{G}_m$ are "the same". Moreover, since $m \geq r + 1$, by just looking at the u and its neighbourhood, the DSA $\mathcal{M}$ cannot differentiate whether u is a vertex in G_m or $\tilde{G}_m$. This holds for every vertex u in G_m and $\tilde{G}_m$ (both have the same set of vertices), and implies that $\mathcal{M}$ cannot differentiate G_m and $\tilde{G}_m$. ◀

Concerning the graphs G_m and $\widetilde{G}_m$ used in the above proof, note that the maximum length of a walk[8] in G_m is 4, while $\widetilde{G}_m$ contains walks of arbitrary lengths. Thus, we obtain the following where, for $\ell \in \mathbb{N}$, we define ℓ-WALK as the class of all graphs that contain a walk of length ℓ.

▶ **Corollary 6.** *Let $\ell \geq 5$. There is no DSA that recognises ℓ-WALK.*

However, when restricting attention to bounded degree graphs, DSAs are quite powerful: they can recognise all properties definable in first-order logic with modulo counting quantifiers. That is, first-order logic enriched by quantifiers of the form $\exists^{i \bmod m} x\, \psi$, stating that the number of nodes x satisfying ψ is congruent i modulo m, for integers $m \geq 1$ and $i \in \{0, \ldots, m-1\}$.

[8] A walk of length ℓ is a sequence of ℓ edges (a_0, a_1), (a_1, a_2), $\ldots$, $(a_{\ell-1}, a_\ell)$ in which repetition of vertices/edges is allowed.

For a vertex u in a graph G, define in-deg(u) and out-deg(u) as the in-degree and the out-degree of u, respectively, and let deg(u) = in-deg(u) + out-deg(u), and let deg(G) = $\max_{u \in V(G)}(\deg(u))$ be the degree of G.

▶ **Theorem 7.** *Let $d \geq 2$ and let φ be a sentence of first-order logic with modulo counting quantifiers. There is a DSA $\mathcal{M}_{\varphi,d}$ such that $\mathcal{G}(\mathcal{M}_{\varphi,d}) = \{G : \deg(G) \leq d \text{ and } G \models \varphi\}$.*

For $d, \ell \geq 0$, define 2^ℓ-WALK$_d$ to be the class of all graphs G such that $\deg(G) \leq d$ and there is a walk of length 2^ℓ in G.

▶ **Theorem 8.**
1. *For every $d, \ell \geq 0$, there is an ℓ-round DSA $\mathcal{M}$ such that $\mathcal{G}(\mathcal{M}) = (2^\ell)$-WALK$_d$.*
2. *For every $\ell \geq 0$, there is no ℓ-round DSA that recognises $(2^{\ell+1})$-WALK$_2$.*
3. *For every $\ell \in \mathbb{N}$, $(\ell{+}1)$-round DSAs are strictly more expressive than ℓ-round DSAs.*

5 Distributed streaming with register transducers (DST)

In this section we introduce the model DST, which is stronger than the DSA-model. As mentioned earlier, the only difference between DSTs and DSAs is on the reducer level. Within a DSA, a reducer is a register automaton that makes one pass over its input, and upon finishing this pass, it outputs a finite bag of values determined by its final configuration. In contrast, within a DST, a reducer is an *RT-reducer* which consists of two components: a finite-state automaton and a transducer system; and makes *two* passes over the input. In the first pass, it uses its finite-state automaton to read the input, but does not produce any output. The final state of the first pass serves as the initial state for the transducer system to make another pass on the input. During this second pass, the transducer outputs a bag of values for each input value (hence the name transducer).

In the next few paragraphs we present the formal definition of DSTs. We start by extending Definition 1 to transducer systems.

▶ **Definition 9.** For $r \in \mathbb{N}$, an *r-register transducer system over* $\mathbf{A}_k$ is a tuple $\mathcal{T} = \langle Q, \delta, \mu \rangle$, where $r \geq k$, Q is a finite set of states, δ is a transition function from $Q \times \Sigma \times \mathcal{F}([k], 2^{[r]})$ to $\mathcal{P}([r], [k]) \times Q$, and μ is a transducer function from $Q \times \Sigma \times \mathcal{F}([k], 2^{[r]})$ to Bags$(\Sigma \times \mathcal{F}([k], [r{+}k]))$.

Thus, an r-register transducer system $\mathcal{T} = \langle Q, \delta, \mu \rangle$ is a transition system $\langle Q, \delta \rangle$ extended with a transducer function μ. The meaning of δ is the same as before, while the meaning of μ is as follows. If on input $(\sigma, \bar{a})$ the automaton is in configuration $(q, \bar{u})$, and for each $i \in [k]$, the data value a_i appears in exactly the registers in $f(i)$, then the transducer function outputs the finite bag $C \sqsubseteq \mathbf{A}_k$ which is obtained from $\tilde{C} := \mu(q, \sigma, f)$ by replacing every $(\sigma', h) \in \tilde{C}$ with the value $(\sigma', \bar{v})$ where, for each $i \in [k]$,

$$v_i \quad = \quad \begin{cases} u_{h(i)} & \text{if } h(i) \leq r \\ a_{h(i)-r} & \text{if } h(i) \geq r{+}1 \end{cases}$$

(i.e., the function h tells us for each of the k positions i of $\bar{v}$, that the value at this position should be the value at the $h(i)$-th position of the tuple $\bar{u}\bar{a}$). We say that C is the output of μ from $(\sigma, \bar{a})$ and $(q, \bar{u})$.

Let $t = t_1 \cdots t_n$ be a sequence of elements of $\mathbf{A}_k$. When starting with a configuration $(q, \bar{u})$, the transducer system $\mathcal{T} = \langle Q, \delta, \mu \rangle$ processes t as follows: It runs the transition system $\langle Q, \delta \rangle$ on t starting with configuration $(p_0, \bar{v}_0) := (q, \bar{u})$, resulting in a run $(p_0, \bar{v}_0), \ldots, (p_n, \bar{v}_n)$.

During this run, on reading each t_i it outputs the bag C_i, defined as the output of μ from t_i and $(p_{i-1}, \bar{v}_{i-1})$.

The union C of the bags $C_1, \ldots, C_n$ is the output of the transducer system $\mathcal{T}$ on t from the configuration $(q, \bar{u})$.[9]

▶ **Definition 10.** An *RT-reducer over* $\mathbf{A}_k$ is a tuple $red = (\mathcal{A}, \mathcal{T}, \rho_{in})$, where $\mathcal{A}$ is a commutative finite-state automaton[10] over the alphabet Σ and $\mathcal{T}$ is an r-register transducer system over $\mathbf{A}_k$ for $r \geq k$, and ρ_{in} is a function that maps an element of $\mathbf{A}_k$ to a state of $\mathcal{A}$. As before, ρ_{in} is required to be generic.

As input, an RT-reducer $red = (\mathcal{A}, \mathcal{T}, \rho_{in})$ receives a key-bag-value pair $\langle \mathsf{key} : B \rangle \in \mathbf{A}_k \times \mathrm{Bags}(\mathbf{A}_k)$. Let $t = t_1 \cdots t_m$ be an enumeration of the elements in B. First, the finite state automaton $\mathcal{A}$ reads only the labels in t starting from the state $\rho_{in}(\mathsf{key})$, and ends in a configuration, say q. Then, the transducer system $\mathcal{T}$ reads t starting from the configuration $(q, \bar{a})$, where $\bar{a}$ is the data values component in key. The output of *red* on $\langle \mathsf{key} : B \rangle$ is the output of $\mathcal{T}$ on t. As in the case of RA-reducers, we want to insist that the output $red(\mathsf{key}, B)$ is independent of the order of elements in B read by $\mathcal{T}$. Therefore, we require RT-reducers to be *commutative*.

Finally, we are ready to define DST.

▶ **Definition 11.** An ℓ-round DST is a tuple $\mathcal{M} = (map_1, red_1, \ldots, map_\ell, red_\ell, agg)$, where each map_i is a generic mapper, each red_i is an RT-reducer, and agg is an RA-aggregator.

The notion of acceptance, along with the notions $\mathcal{M}(G)$ (for a graph G) and $\mathcal{G}(\mathcal{M})$, are defined in the same way as for DSAs. Note that we require the reducer to make two passes on the values, where a finite state automaton is making the first pass, and a transducer is making the second pass. Without two passes, semijoin algebra cannot be captured. The rest of this section is devoted to our study of the expressiveness of DSTs.

Our first result states that for DSTs, ℓ rounds are sufficient and necessary to recognise the existence of a walk of length 2ℓ. Recall that ℓ-WALK (for $\ell \in \mathbb{N}$) is the class of all graphs that contain a walk of length ℓ.

▶ **Theorem 12.**

1. *For each $\ell \in \mathbb{N}$ there is an ℓ-round DST $\mathcal{M}$ such that $\mathcal{G}(\mathcal{M}) = (2\ell)$-WALK.*
2. *For each $\ell \in \mathbb{N}$, there is no ℓ-round DST that recognises $(2\ell{+}2)$-WALK.*
3. *For every $\ell \in \mathbb{N}$, $(\ell{+}1)$-round DSTs are strictly more expressive than ℓ-round DSTs.*

In particular, 6-WALK can be recognised by a DST. From Corollary 6, we know that no DSA can recognise 6-WALK. Furthermore, by modifying the proof of Theorem 5, we can also show that DSTs are still not powerful enough to solve the TRIANGLE problem. These two facts are stated formally as follows:

▶ **Theorem 13.**

- *DSTs are strictly stronger than DSAs.*
- *There is no DST that recognises* TRIANGLE.

[9] We should remark that although register transducers are very natural extension of register automata, we are not aware of any literature where they have been studied previously.

[10] A finite state automaton $\mathcal{A}$ is commutative, if for every sequence $\sigma_1 \cdots \sigma_m$, for every permutation π on $[m]$, on reading the sequence $\sigma_1 \cdots \sigma_m$ and $\sigma_{\pi(1)} \cdots \sigma_{\pi(m)}$, the automaton ends in the same state.

6 Distributed streaming with register transducers and joins

In this section we introduce the strongest model of this paper, called *Distributed streaming with register transducers and joins* (DSTJ). It is designed specifically to capture relational algebra. The difference between DSTJs and DSTs is again on the reducer level. In DSTJs, a reducer can be of two types: an RT-reducer or a joiner, which is simply an abstract function that performs the Cartesian product between two sets. Its formal definition is as follows.

A *joiner* is a triplet $\mathcal{J} = (\alpha, \beta, \gamma)$, where α, β, γ are symbols from Σ. A joiner $\mathcal{J} = (\alpha, \beta, \gamma)$ works as follows. The input is a key-bag-value pair $\langle \mathsf{key} : \mathsf{VAL} \rangle$. Let $\mathsf{key} = (\zeta, \bar{a})$. The joiner $\mathcal{J}$ outputs the bag $\{\!\{(\alpha, \bar{a}\bar{b}\bar{c}) \mid (\beta, \bar{b}) \in \mathsf{VAL} \text{ and } (\gamma, \bar{c}) \in \mathsf{VAL}\}\!\}$.

Next, we define a *relational reducer* as a reducer that can choose either an RT-reducer or a joiner to process its values.

▶ **Definition 14.** A *relational reducer* is a tuple $\mathcal{R} = (F, \mathcal{J}, \mathcal{T})$, where $F : \Sigma \to \{C, T\}$ is a function that maps $\sigma \in \Sigma$ to either C or T, $\mathcal{J}$ is a joiner, and $\mathcal{T}$ is an RT-reducer.

On input of a key-bag-value pair $\langle \mathsf{key} : \mathsf{VAL} \rangle$, a relational reducer does the following: Let $\mathsf{key} = (\sigma, t)$. If $F(\sigma) = C$, the relational reducer runs the joiner $\mathcal{J}$ on $\langle \mathsf{key} : \mathsf{VAL} \rangle$. If $F(\sigma) = T$, it runs the RT-reducer $\mathcal{T}$ on $\langle \mathsf{key} : \mathsf{VAL} \rangle$.

▶ **Definition 15.** An ℓ-round DSTJ is a tuple $\mathcal{M} = (map_1, red_1, \ldots, map_\ell, red_\ell, agg)$, where each map_i is a generic mapper, each red_i is a relational reducer, and agg is an RA-aggregator.

The notion of acceptance, along with the notions $\mathcal{M}(G)$ (for a graph G) and $\mathcal{G}(\mathcal{M})$, are defined in the same way as for DSTs. The rest of this section is devoted to the expressiveness of DSTJs. Our first expressiveness result states that DSTJ can recognise the existence of a triangle.

▶ **Lemma 16.** *There is a 2-round DSTJ $\mathcal{M}$ such that $\mathcal{G}(\mathcal{M}) =$ TRIANGLE.*

Proof (sketch). Intuitively, in the first round $\mathcal{M}$ collects all pairs (u, v) where there is path of length 2 from u to v. In the second round on each pair (u, v) output in the first round, it checks whether there is an edge from v to u. If so, it outputs a special symbol γ. The aggregator simply checks whether γ appears among the values output by the reducer in the second round. We note that this algorithm is very similar to the algorithm MR-Node-Iterator++ in [27]. ◀

Combining Lemma 16 and Theorem 13, we obtain:

▶ **Theorem 17.** *DSTJs are strictly stronger than DSTs.*

In a graph G, a cycle of length m is sequence of edges $(u_1, u_2), \ldots, (u_{m-1}, u_m), (u_m, u_1) \in E(G)$. It is not necessary that the vertices $u_1, \ldots, u_m$ are pairwise different. For $m \geq 3$, define the class m-CYCLE where a graph $G \in m$-CYCLE if and only if G contains a cycle of length m.

▶ **Theorem 18.** *For each positive integer $\ell \geq 1$, the following holds.*

1. *There is an ℓ-round DSTJ $\mathcal{M}$ such that $\mathcal{G}(\mathcal{M}) = 2^\ell$-CYCLE.*
2. *For each $\ell \in \mathbb{N}$, there is no ℓ-round DSTJ $\mathcal{M}$ such that $\mathcal{G}(\mathcal{M}) = 2^{\ell+1}$-CYCLE.*
3. *$(\ell{+}1)$-round DSTJs are strictly more expressive than ℓ-round DSTJs.*

7 Semijoin algebra and relational algebra

In this section we study the connections between the models DSA/DST/DSTJ and the semijoin algebra and the relational algebra. To this end, we define the corresponding model for DSA/DST/DSTJ for non-Boolean queries on general databases.

We fix a finite vocabulary τ of relation symbols with associated arities and assume every database DB to be over τ. For a relation symbol R and a tuple of values $\bar{a}$ whose arity matches the arity of R, we call $R(\bar{a})$ a *fact*. Clearly, a database is just a finite set of facts. The initial mapper will now receive as input an enumeration of all the facts in the database.

Here we assume that Σ contains τ, and as before, $\mathbf{A}_k$ denotes $\Sigma \times \mathbf{D}^k_\#$. A fact $R(\bar{a})$ can then be viewed as an element of $\mathbf{A}_k$ by padding an appropriate number of #'s at the end of $\bar{a}$. Similarly, an element $(R, \bar{a}) \in \mathbf{D}^k_\#$ can be viewed as an R-fact by discarding the # components. To avoid being pedantic, we will view elements of $\mathbf{A}_k$ as facts, and vice versa.

▶ **Definition 19.** For every $X \in \{\text{DSA}, \text{DST}, \text{DSTJ}\}$, an ℓ-round *DB-X* over $\mathbf{A}_k$ is a tuple $\mathcal{M} = (map_1, red_1, \ldots, map_\ell, red_\ell)$, where each map_i is a generic mapper, and each red_i is an RA-reducer, an RT-reducer, and relational reducer, when X is DSA, DST and DSTJ, respectively.

On an input database DB, for each $i \in [\ell]$, the bags $M_i(\mathsf{DB})$ of key-value pairs and the bags $R_i(\mathsf{DB})$ of values are defined as in Section 2. For every $X \in \{\text{DSA,DST,DSTJ}\}$, on input of a database DB, the output of a DB-X $\mathcal{M}$ is defined as the tuples from $R_\ell(\mathsf{DB})$.

Note that in the lower bounds proved in the previous sections are for models with aggregator components, which the non-Boolean models do not have. Obviously, all the lower bounds for the Boolean queries carry over to their non-Boolean counterparts. Next, we are going to show that on classes of bounded degree databases, DB-DSA can evaluate the relational algebra; while over general databases, DB-DST and DB-DSTJ can evaluate the semijoin algebra and the relational algebra, respectively. In the following $e(\mathsf{DB})$ denotes the result of evaluating the expression e on the database DB.

▶ **Theorem 20.**

1. *For every relational algebra expression e and an integer $d > 0$, there is a DB-DSA $\mathcal{M}_e$ such that for every database DB of degree at most d, $\mathcal{M}_e(\mathsf{DB}) = e(\mathsf{DB})$.*
2. *For every semijoin algebra expression e, there is a DB-DST $\mathcal{M}_e$ such that for every database DB, $\mathcal{M}_e(\mathsf{DB}) = e(\mathsf{DB})$.*
3. *For every relational algebra expression e, there is a DB-DSTJ $\mathcal{M}_e$ such that for every database DB, $\mathcal{M}_e(\mathsf{DB}) = e(\mathsf{DB})$.*

Moreover, each $\mathcal{M}_e$ can be constructed effectively.

Proof. Proof of (1). We are going to show that on bounded degree databases, each RA operation can be simulated by one round DSA $\mathcal{M} = (map, red)$, in which the tuples output by the reducer has the same label T. Its generalisation for arbitrary RA-expression can be established via straightforward induction. Note that the bounded degree is only needed for the semijoin and join operations.

- Union: $R \cup S$.
  The mapper works as follows. On input t, if t is $R(\bar{a})$, it outputs $\langle T(\bar{a}) : R(\bar{a}) \rangle$; if t is $S(\bar{a})$, it outputs $\langle T(\bar{a}) : S(\bar{a}) \rangle$; otherwise, it outputs nothing. The reducer *red* works as follows. On key t, it outputs t itself.
- Intersection: $R \cup S$.

The mapper works like in the case $R \cup S$. The reducer *red* works as follows. On key t, it checks whether there are two tuples, one with label R and another with label S. If so, it outputs t itself. Otherwise, it outputs nothing.

- Difference: $R - S$.
  The mapper works like in the case $R \cup S$. The reducer *red* works as follows. On key t, it checks whether there is a tuple with label R and there is no tuple with label S. If so, it outputs t itself. Otherwise, it outputs nothing.
- Selection: $\sigma_{i=j}(R)$.
  The mapper works as follows. On input t, if t is $R(\bar{a})$ and $a_i = a_j$ it outputs $\langle T(\bar{a}) : T(\bar{a}) \rangle$; otherwise, it outputs nothing. The reducer *red* works as follows. On key t, it outputs t itself.
- Projection: $\pi_{i_1,\ldots,i_m}(R)$.
  The mapper works as follows. On input t, it outputs $\langle T(a_{i_1}, \ldots, a_{i_m}) : T(a_{i_1}, \ldots, a_{i_m}) \rangle$, if t is $R(\bar{a})$. Otherwise, it outputs nothing. The reducer *red* works as follows. On key t, it outputs t itself.
- Semijoin: $R \ltimes_\theta S$.
  Let I and J be the projection of θ to its first and second coordinates. The mapper works as follows. On input t, if t is $R(\bar{a})$, it outputs $\langle T(\pi_I(\bar{a})) : R(\bar{a})) \rangle$; if t is $S(\bar{a})$, it outputs $\langle T(\pi_J(\bar{a})) : S(\pi_J(\bar{a}))) \rangle$; otherwise, it outputs nothing. The reducer *red* works as follows. On key t, it passes through its input, while remembering all the R-facts in its registers. Since the input dataabse DB is of bounded degree, say $\leq d$, the number R-facts associated with one particular key is also bounded by d. So we can choose the number of registers in *red* to be kd, where k is the arity of R, to accommodate all the R-facts and S-facts. If there is at least an S-fact among its input, it outputs all the R-facts. Otherwise, if there is no S-fact among its input, it outputs nothing.
- Join: $R \bowtie_\theta S$.
  As in the semijoin case, let I and J be the projection of θ to its first and second coordinates. The mapper works in the same manner as in the semijoin case. The reducer *red* works as follows. On key t, it passes through its input, while remembering all the R-facts and all its S-facts in its registers. Similar to the semijoin case, since DB is of bounded degree, say $\leq d$, the number R-facts and S-facts associated with one particular key is also bounded by d. So we can choose the number of registers in *red* to be $(k + l)d$, where k and l are the arities of R and S, respectively, to accommodate all the R-facts and S-facts. If there is at least one R-fact and one S-fact among its input, it outputs all the combination of the join among the R-facts and S-facts in the input. Otherwise, if there is no S-fact or if there is no R-fact, it outputs nothing.

Proof of (2). Note that for union, intersection, difference, selection and projection, one-round DSA presented above works for arbitrary database. Hence, it is sufficient to show that semijoin operation $R \ltimes_\theta S$ over arbitrary graph can be done in one-round DST.

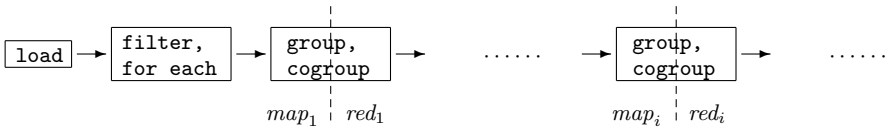
Let I and J be the projection of θ to its first and second coordinates. The mapper works as in the case of semijoin above. The reducer *red* works as follows. On key t, in the first pass it checks whether there is an S-fact among the input, which can be done trivially by a finite state automaton. If there is an S-fact, in the second pass on each R-fact $R(\bar{a})$ in the input, it outputs $T(\bar{a})$. If there is no S-fact, in the second pass it does nothing and output nothing.

Proof of (3): Again, since all the other operations can be evaluated by DSA and DST, it is sufficient to show that join operation $R \bowtie_\theta S$ over arbitrary database can be done in one-round DSTJ. The mapper works similarly as in the case of join above. Then the reducer uses joiner to pair off the R-tuples with S-tuples. ◀

Note that 6-WALK can be expressed in the semijoin algebra, and TRIANGLE can be expressed in the relational algebra. Thus, it follows that DB-DSA and DB-DST cannot evaluate all semijoin algebra and relational algebra expressions, respectively.

Comparison with Pig. To end this section, we give a brief description of the Pig system, and relate it to our models here. For more details, we refer the reader to [15, 23]. In short, Pig is a system built on the Hadoop system to evaluate queries on a large relational database written in a language called *Pig Latin*. Upon receiving an input query, Pig generates a MapReduce program that evaluates the query on a given database, where the number of rounds corresponds linearly to the number of `(CO)GROUP` and `JOIN` queries. For each `(CO)GROUP` query it generates a mapper that assigns keys to tuples based on the `BY` clauses in the query, i.e. projecting the tuples to fields in the `BY` clauses. The `JOIN` operations are handled in one of two ways: (i) rewrite into a `COGROUP` followed by a `FOR EACH` operation, which yields a parallel hash-join or sort-merge join, or (ii) use fragment-replicate join. Either way requires one round of MapReduce computation, and can be captured by the joiner.

Obviously, two independent subqueries can be evaluated simultaneously in a one round MapReduce job. Typically a MapReduce compilation of Pig Latin script looks as follows:



The `(CO)GROUP` commands form the boundary between the map and reduce phase. In the current implementation of Pig, the commands in between the boundaries are pushed into the reduce function. Obviously, the `FILTER` and `FOR EACH` command can be implemented as one of RA-reducer or RT-reducer, and `JOIN` as joiner. Hence, one round in the Pig system corresponds to one round of either DSA, DST, or DSTJ.

8 Conclusion

We introduced three simple abstractions of the key-value paradigm in terms of finite memory automata and transducers. Our results emphasise that, even though the proposed models are simple, they form a relevant subclass of MapReduce. In particular, DSTJs can evaluate the whole relational algebra, while DSTs can evaluate the semijoin algebra which forms an important subset of the relational algebra. Furthermore, on the class of bounded degree graphs (and analogously, also for bounded degree databases), DSAs can evaluate all Boolean queries formulated in relational algebra or first-order logic with modulo counting quantifiers. In fact, on this class, we believe DSAs to be equivalent to first-order logic with modulo counting quantifiers. A direction for future research is to extend the current model with arithmetic and aggregation, as SQL queries support modest forms of counting.

Acknowledgements. We thank the anonymous referees for their helpful and inspiring comments. We also thank Jan Van den Bussche for inspiring discussions. The fourth author is supported by FWO Pegasus Marie Curie Fellowship.

References

1 F. Afrati, V. Borkar, M. Carey, N. Polyzotis, and J. Ullman. Map-reduce extensions and recursive queries. In *ICDE*, 2011.

2 F. Afrati, D. Fotakis, and J. Ullman. Enumerating subgraph instances using map-reduce. In *ICDE*, 2013.

3 F. Afrati, P. Koutris, D. Suciu, and J. Ullman. Parallel skyline queries. In *ICDT*, 2012.

4 F. Afrati, A. Dash Sarma, S. Salihoglu, and J. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

5 F. Afrati and J. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, 2010.

6 F. Afrati and J. Ullman. Transitive closure and recursive datalog implemented on clusters. In *EDBT*, 2012.

7 T. Ameloot, F. Neven, and J. Van den Bussche. Relational transducers for declarative networking. *Journal of the ACM*, 60(2):15, 2013.

8 Apache Bagel. Bagel. http://spark.apache.org/docs/0.7.3/bagel-programming-guide.html.

9 P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *PODS*, 2013.

10 P. Beame, P. Koutris, and D. Suciu. Skew in parallel query processing. In *PODS*, 2014.

11 F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *WWW*, 2010.

12 E. Codd. A relational model of data for large shared data banks. *Communication of the ACM*, 13(6):377–387, 1970.

13 J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

14 J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. *Communication of the ACM*, 53(1):72–77, 2010.

15 A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. Building a highlevel dataflow system on top of mapreduce: The pig experience. *PVLDB*, 2(2):1414–1425, 2009.

16 J. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010.

17 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

18 H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *SODA*, 2010.

19 P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, 2011.

20 R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. In *SPAA*, 2013.

21 S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA*, 2011.

22 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.

23 C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD Conference*, 2008.

24 Apache Pig. Pig. `http://pig.apache.org/`.

25 Apache Spark. Spark. `http://spark.apache.org`.

26 Apache Spark. Spark programming guide. `http://spark.apache.org/docs/latest/programming-guide.html`.

27 S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, 2011.

28 Y. Tao, W. Lin, and X. Xiao. Minimal mapreduce algorithms. In *SIGMOD*, 2013.

29 A. Thusoo, J. Sen Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *ICDE*, 2010.

30 L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

31 T. White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (3. ed., revised and updated)*. O'Reilly, 2012.

32 R. Xin, J. Rosen, M. Zaharia, M. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *SIGMOD*, 2013.

33 M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

From Causes for Database Queries to Repairs and Model-Based Diagnosis and Back

Babak Salimi[1] **and Leopoldo Bertossi**[2]

1 **Carleton University, School of Computer Science, Ottawa, Canada**
bsalimi@scs.carleton.ca
2 **Carleton University, School of Computer Science, Ottawa, Canada**
bertossi@scs.carleton.ca

Abstract

In this work we establish and investigate connections between causality for query answers in databases, database repairs wrt. denial constraints, and consistency-based diagnosis. The first two are relatively new problems in databases, and the third one is an established subject in knowledge representation. We show how to obtain database repairs from causes and the other way around. Causality problems are formulated as diagnosis problems, and the diagnoses provide causes and their responsibilities. The vast body of research on database repairs can be applied to the newer problem of determining actual causes for query answers and their responsibilities. These connections, which are interesting *per se*, allow us, after a transition -inspired by consistency-based diagnosis- to computational problems on hitting sets and vertex covers in hypergraphs, to obtain several new algorithmic and complexity results for database causality.

1998 ACM Subject Classification H.2.3 [Database Management] Languages, H.2.4 [Database Management] Systems, I.2.3 [Artificial Intelligence] Deduction and Theorem Proving, I.2.4 [Artificial Intelligence] Knowledge Representation Formalisms and Methods, F.1.3 [Computation of Abstract Devices] omplexity Measures and Classes

Keywords and phrases causality, diagnosis, repairs, consistent query answering, integrity constraints

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.342

1 Introduction

When querying a database, a user may not always obtain the expected results, and the system could provide some explanations. They could be useful to further understand the data or check if the query is the intended one. Actually, the notion of explanation for a query result was introduced in [41], on the basis of the deeper concept of *actual causation*.

A tuple t is an *actual cause* for an answer $\bar{a}$ to a conjunctive query $\mathcal{Q}$ from a relational database instance D if there is a *contingent* set of tuples Γ, such that, after removing Γ from D, $\bar{a}$ is still an answer, but after further removing t from $D \setminus \Gamma$, $\bar{a}$ is not an answer anymore. Here, Γ is a set of tuples that has to accompany $\bar{a}$ for it to be a cause. Actual causes and contingent tuples are restricted to be among a pre-specified set of *endogenous tuples*, which are admissible, possible candidates for causes, as opposed to *exogenous tuples*, which may also be present in the database. In rest of this paper, whenever we simply say "cause", we mean "actual cause".

In applications involving large data sets, it is crucial to rank potential causes by their *responsibilities* [42, 41], which reflect the relative (quantitative) degrees of their causality for a query result. The responsibility measure for a cause is based on its *contingency sets*: the smallest (one of) its contingency sets, the strongest it is as a cause.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 342–362

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Actual causation, as used in [41], can be traced back to [29, 30], which provides a model-based account of causation on the basis of *counterfactual dependence*. Responsibility was introduced in [18], to capture the intuitive notion of *degree of causation*.

Apart from the explicit use of causality, research on explanations for query results has focused mainly, and rather implicitly, on provenance [12, 13, 14, 21, 36, 34, 53]. A close connection between causality and provenance has been established in [41]. However, causality is a more refined notion that identifies causes for query results on the basis of user-defined criteria, and ranks causes according to their responsibilities [42].

Consistency-based diagnosis [47], a form of model-based diagnosis [52, sec. 10.3], is an area of knowledge representation. The problem here is, given the *specification* of a system in some logical formalism and a usually unexpected *observation* about the system, to obtain *explanations* for the observation, in the form of a diagnosis for the unintended behavior.

In a different direction, a database instance, D, that is expected to satisfy certain integrity constraints may fail to do so. In this case, a *repair* of D is a database D' that does satisfy the integrity constraints and *minimally departs* from D. Different forms of minimality can be applied and investigated. A *consistent answer* to a query from D and wrt. the integrity constraints is a query answer that is obtained from all possible repairs, i.e. is invariant or certain under the class of repairs. These notions were introduced in [2] (surveys of the area can be found in [7, 9]). Although not in the context of repairs, consistency-based diagnosis has been applied to consistency restoration of a database wrt. integrity constraints [27].

These three forms of reasoning, namely inferring causes from databases, consistency-based diagnosis, and consistent query answering (and repairs) are all *non-monotonic* [49]. For example, a (most responsible) cause for a query result may not be such anymore after the database is updated. Furthermore, they all reflect some sort of *uncertainty* about the information at hand. In this work we establish natural, precise, useful, and deeper connections between these three reasoning tasks.

More precisely, we unveil a strong connection between computing causes and their responsibilities for conjunctive query answers, on one hand, and computing repairs in databases wrt. *denial constraints*, on the other. These computational problems can be reduced to each other. In order to obtain repairs wrt. *a set* of denial constraints from causes, we investigate causes for queries that are *unions of conjunctive queries*, and develop algorithms to compute causes and responsibilities.

We show that inferring and computing actual causes and their responsibilities in a database setting become diagnosis reasoning problems and tasks. Actually, a causality-based explanation for a conjunctive query answer can be viewed as a diagnosis, where in essence the first-order logical reconstruction of the relational database provides the system description [48], and the observation is the query answer. We also establish a bidirectional connection between diagnosis and repairs.

Being the causality problems the main focus of this work, we take advantage of algorithms and complexity results both for consistency-based diagnosis; and database repairs and consistent query answering [9]. In this way, we obtain new complexity results for the main problems of causality, namely computing actual causes, determining their responsibilities, and obtaining most responsible causes; and also for their decision versions. In particular, we obtain fixed-parameter tractable algorithms for some of them. More precisely, our main results are as follows:[1] (the complexity results are all in data complexity)

[1] A few of the results included here appear in [49].

1. For a boolean conjunctive query and its associated denial constraint (the former being its violation view), we establish a precise connection (characterization and computational reductions) between actual causes for the query (being true) and the subset- and cardinality-repairs of the instance wrt. the denial constraint. We obtain causes from repairs.
2. We obtain repairs from causes, for which we extend the treatment of causality to unions of conjunctive queries (to represent multiple denial constraints). We characterize an actual cause's responsibility in terms of cardinality-repairs. We provide algorithms to compute causes and their (minimal) contingency sets for unions of conjunctive queries. The causes can be computed in PTIME.
3. We establish a precise connection between consistency-based diagnosis for a boolean conjunctive query being unexpectedly true according to a system description, and causes for the query being true. In particular, we show how to compute actual causes, their contingency sets, and responsibilities using the diagnosis characterization. Hitting-set-based algorithmic approaches to diagnosis inspire our algorithmic/complexity approaches to causality.
4. We reformulate the causality problems as hitting set problems and vertex cover problems on hypergraphs, which allows us to apply results and techniques for the latter to causality.
5. (a) Checking minimal contingency sets can be done in PTIME. (b) The responsibility (decision) problem for conjunctive queries becomes *NP*-complete. (c) However, it is fixed-parameter tractable when the parameter is the inverse of the responsibility bound. (d) The functional problem of computing the causes' responsibilities is $FP^{NP(log(n))}$-complete, and deciding most responsible causes is $P^{NP(log(n))}$-complete.
6. The structure of the resulting hitting-set problem allows us to obtain efficient parameterized algorithms and good approximation algorithms for computing causes and minimal contingency sets.
7. On the basis of the causality/repair connection, and the dichotomy result for causality [41], we obtain a dichotomy result for the complexity of deciding the existence of repairs of a certain size wrt. single, self-join-free denial constraints.
8. We discuss extensions and open issues that deserve investigation.

The paper is structured as follows. Section 2 introduces technical preliminaries for relational databases, causality in databases, database repairs and consistent query answering, consistency-based diagnosis, and relevant complexity classes. Section 3 characterizes actual causes and responsibilities in terms of database repairs. Section 3 characterizes repairs and consistent query answering in terms of causes and contingency sets for queries that are unions of conjunctive queries; and presents an algorithm for computing both of the latter. Section 5 formulates causality problems as consistency-based diagnosis problems, and the latter as repair problems. Section 6 shows complexity and algorithmic results; in particular a fixed-parameter tractability result for causes' responsibilities. Finally, Section 7 discusses several relevant issues, connections and open problems around causality in databases. Proofs of results without an implicit proof in this paper can be found in [50].

2 Preliminaries

We consider relational database schemas of the form $\mathcal{S} = (U, \mathcal{P})$, where U is the possibly infinite database domain of *constants* and $\mathcal{P}$ is a finite set of *database predicates*[2] of fixed

[2] As opposed to built-in predicates (e.g. $\neq$) that we assume do not appear, unless explicitly stated otherwise.

arities. A database instance D compatible with $\mathcal{S}$ can be seen as a finite set of ground atomic formulas (in databases aka. atoms or tuples), of the form $P(c_1, ..., c_n)$, where $P \in \mathcal{P}$ has arity n, and $c_1, \ldots, c_n \in U$. A *conjunctive query* (CQ) is a formula $\mathcal{Q}(\bar{x})$ of the first-order (FO) logic language, $\mathcal{L}(\mathcal{S})$, associated to $\mathcal{S}$ of the form $\exists \bar{y}(P_1(\bar{t}_1) \wedge \cdots \wedge P_m(\bar{t}_m))$, where the $P_i(\bar{t}_i)$ are atomic formulas, i.e. $P_i \in \mathcal{P}$, and the $\bar{t}_i$ are sequences of terms, i.e. variables or constants. The $\bar{x}$ in $\mathcal{Q}(\bar{x})$ shows all the free variables in the formula, i.e. those not appearing in $\bar{y}$. If $\bar{x}$ is non-empty, the query is *open*. If $\bar{x}$ is empty, the query is *boolean* (a BCQ), i.e. the query is a sentence, in which case, it is true or false in a database, denoted by $D \models \mathcal{Q}$ and $D \not\models \mathcal{Q}$, respectively. A sequence $\bar{c}$ of constants is an answer to an open query $\mathcal{Q}(\bar{x})$ if $D \models \mathcal{Q}[\bar{c}]$, i.e. the query becomes true in D when the variables are replaced by the corresponding constants in $\bar{c}$.

An *integrity constraint* is a sentence of language $\mathcal{L}(\mathcal{S})$, and then, may be true or false in an instance for schema $\mathcal{S}$. Given a set IC of integrity constraints, a database instance D is *consistent* if $D \models IC$; otherwise it is said to be *inconsistent*. In this work we assume that sets of integrity constraints are always finite and logically consistent. A particular class of integrity constraints is formed by *denial constraints* (DCs), which are sentences κ of the form: $\forall \bar{x} \neg(A_1(\bar{x}_1) \wedge \cdots \wedge A_n(\bar{x}_n)$, where $\bar{x} = \bigcup \bar{x}_i$ and each $A_i(\bar{x}_i)$ is a database atom, i.e. predicate $A \in \mathcal{P}$. (The atoms may contain constants.) Denial constraints are exactly the negations of BCQs.

Causality and Responsibility. Assume that the database instance is split in two, i.e. $D = D^n \cup D^x$, where D^n and D^x denote the sets of *endogenous* and *exogenous* tuples, respectively. A tuple $t \in D^n$ is called a *counterfactual cause* for a BCQ $\mathcal{Q}$, if $D \models \mathcal{Q}$ and $D \smallsetminus \{t\} \not\models \mathcal{Q}$. A tuple $t \in D^n$ is an *actual cause* for $\mathcal{Q}$ if there exists $\Gamma \subseteq D^n$, called a *contingency set*, such that t is a counterfactual cause for $\mathcal{Q}$ in $D \smallsetminus \Gamma$ [41]. We will concentrate mostly on CQs. However, the definition of actual causes and contingency sets can be applied without a change to monotone queries in general [41].

The *responsibility* of an actual cause t for $\mathcal{Q}$, denoted by $\rho_{_D}(t)$, is the numerical value $\frac{1}{|\Gamma|+1}$, where $|\Gamma|$ is the size of the smallest contingency set for t. We can extend responsibility to all the other tuples in D^n by setting their value to 0. Those tuples are not actual causes for $\mathcal{Q}$.

▶ **Example 1.** Consider $D = D^n = \{R(a_4, a_3), R(a_2, a_1), R(a_3, a_3), S(a_4), S(a_2), S(a_3)\}$, and the query $\mathcal{Q} : \exists x \exists y(S(x) \wedge R(x, y) \wedge S(y))$. It holds: $D \models \mathcal{Q}$.

Tuple $S(a_3)$ is a counterfactual cause for $\mathcal{Q}$. If $S(a_3)$ is removed from D, $\mathcal{Q}$ is not true anymore. Therefore, the responsibility of $S(a_3)$ is 1. Besides, $R(a_4, a_3)$ is an actual cause for $\mathcal{Q}$ with contingency set $\{R(a_3, a_3)\}$. If $R(a_3, a_3)$ is removed from D, $\mathcal{Q}$ is still true, but further removing $R(a_4, a_3)$ makes $\mathcal{Q}$ false. The responsibility of $R(a_4, a_3)$ is $\frac{1}{2}$, because its smallest contingency sets have size 1. Likewise, $R(a_3, a_3)$ and $S(a_4)$ are actual causes for $\mathcal{Q}$ with responsibility $\frac{1}{2}$.

For the same $\mathcal{Q}$, but with $D = \{S(a_3), S(a_4), R(a_4, a_3)\}$, and the partition $D^n = \{S(a_4), S(a_3)\}$ and $D^x = \{R(a_4, a_3)\}$, it turns out that both $S(a_3)$ and $S(a_4)$ are counterfactual causes for $\mathcal{Q}$. ◀

Notation: $\mathcal{CS}(D^n, D^x, \mathcal{Q})$ denotes the set of actual causes for BCQ $\mathcal{Q}$ (being true) from instance $D = D^n \cup D^x$. When $D^n = D$ and $D^x = \emptyset$, we sometimes simply write: $\mathcal{CS}(D, \mathcal{Q})$.

Database Repairs. Given a set IC of integrity constraints, a *subset repair* (simply, S-repair) of a possibly inconsistent instance D for schema $\mathcal{S}$ is an instance D' for $\mathcal{S}$ that satisfies

IC and makes $\Delta(D, D') = (D \smallsetminus D') \cup (D' \smallsetminus D)$ minimal under set inclusion. $Srep(D, IC)$ denotes the set of S-repairs of D wrt. IC [2]. Similarly, D' is a *cardinality repair* (simply C-repair) of D if D' satisfies IC and minimizes $|\Delta(D, D')|$. $Crep(D, IC)$ denotes the class of C-repairs of D wrt. IC. C-repairs are S-repairs of minimum cardinality.

For DCs, S-repairs and C-repairs are obtained from the original instance by deleting an S-minimal, resp. C-minimal, set of tuples.[3] More generally, different *repair semantics* may be considered to restore consistency wrt. general integrity constraints. They depend on the kind of allowed updates on the database (i.e. tuple insertions/deletions, changes of attribute values), and the minimality conditions on repairs (e.g. subset-minimality, cardinality-minimality, etc.). Given D and IC, a repair semantics determines a class of intended or preferred repairs [9, sec. 2.5].

Given a repair semantics, RS, $\bar{c}$ is a *consistent answer* to an open query $\mathcal{Q}(\bar{x})$ if $D' \models \mathcal{Q}[\bar{c}]$ for every RS-repair D'. A BCQ is *consistently true* if it is true in all RS-repairs. If $\bar{c}$ is a consistent answer to $\mathcal{Q}(\bar{x})$ wrt. S-repairs, we say it is an *S-consistent answer*. Similarly for *C-consistent answers*. Consistent query answering for DCs under S-repairs was investigated in detail [17]. C-repairs and consistent query answering were investigated in detail in [39]. (Cf. [9] for more references.)

Consistency-Based Diagnosis. Consistency-based diagnosis, a form of model-based diagnosis [52, sec. 10.4], considers problems $\mathcal{M} = (SD, COMPS, OBS)$, where SD is the description in logic of the intended properties of a system under the *explicit* assumption that all the *components* in $COMPS$, are working normally. OBS is a FO sentence that represents the observations. If the system does not behave as expected (as shown by the observations), then the logical theory obtained from $SD \cup OBS$ plus the explicit assumption, say $\bigwedge_{c \in COMPS} \neg Ab(c)$, that the components are indeed behaving normally, becomes inconsistent. Ab is an *abnormality* predicate.[4]

The inconsistency is captured via the *minimal conflict sets*, i.e. those minimal subsets $COMPS'$ of $COMPS$, such that $SD \cup OBS \cup \{\bigwedge_{c \in COMPS'} \neg Ab(c)\}$ is inconsistent. As expected, different notions of minimality can be used at this point.

A *minimal diagnosis* for $\mathcal{M}$ is a minimal subset Δ of $COMPS$, such that $SD \cup OBS \cup \{\neg Ab(c) \mid c \in COMPS \smallsetminus \Delta\} \cup \{Ab(c) \mid c \in \Delta\}$ is consistent. That is, consistency is restored by flipping the normality assumption to abnormality for a minimal set of components, and those are the ones considered to be (jointly) faulty. The notion of minimality commonly used is S-minimality, i.e. a diagnosis that does not have a proper subset that is a diagnosis. We will use this kind of minimality in relation to diagnosis. Diagnosis can be obtained from conflict sets [47].

Complexity Classes. We recall some complexity classes [46] used in this paper. FP is the class of functional problems associated to decision problem in the class $PTIME$, i.e. that are solvable in polynomial time. P^{NP} (or Δ_2^P) is the class of decision problems solvable in polynomial time by a machine that makes calls to an NP oracle. For $P^{NP(log(n))}$ the number of calls is logarithmic. It is not known if $P^{NP(log(n))}$ is strictly contained in P^{NP}. $FP^{NP(log(n))}$ is similarly defined.

---

[3] We will usually say that a set is S-minimal in a class of sets $\mathcal{C}$ if it minimal under set inclusion in $\mathcal{C}$. Similarly, a set is C-minimal if it is minimal in cardinality within $\mathcal{C}$.

[4] Here, and as usual, the atom $Ab(c)$ expresses that component c is (behaving) abnormal(ly).

3 Actual Causes From Database Repairs

Let $D = D^n \cup D^x$ be an instance for schema $\mathcal{S}$, and $\mathcal{Q}$: $\exists\bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$ a BCQ. $\mathcal{Q}$ may be unexpectedly true, i.e. $D \models \mathcal{Q}$. Now, $\neg\mathcal{Q}$ is logically equivalent to the DC $\kappa(\mathcal{Q}) : \forall\bar{x}\neg(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$. The requirement that $\neg\mathcal{Q}$ holds can be captured by imposing $\kappa(\mathcal{Q})$ on D. Due to $D \models \mathcal{Q}$, it holds $D \not\models \kappa(\mathcal{Q})$. So, D is inconsistent wrt. $\kappa(\mathcal{Q})$, and could be repaired.

Repairs for (violations of) DCs are obtained by tuple deletions. Intuitively, a tuple that participates in a violation of $\kappa(\mathcal{Q})$ in D is an actual cause for $\mathcal{Q}$. S-minimal sets of tuples like this are expected to correspond to S-repairs for D and $\kappa(\mathcal{Q})$. More precisely, given an instance $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, and a tuple $t \in D^n$, we consider the class containing the sets of differences between D and those S- or C-repairs that do not contain t, and are obtained by removing a subset of D^n:

$$\mathcal{DF}^s(D, D^n, \kappa(\mathcal{Q}), t) = \{D \smallsetminus D' \mid D' \in \mathit{Srep}(D, \kappa(\mathcal{Q})),\ t \in (D \smallsetminus D') \subseteq D^n\}, \quad (1)$$

$$\mathcal{DF}^c(D, D^n, \kappa(\mathcal{Q}), t) = \{D \smallsetminus D' \mid D' \in \mathit{Crep}(D, \kappa(\mathcal{Q})),\ t \in (D \smallsetminus D') \subseteq D^n\}. \quad (2)$$

It holds $\mathcal{DF}^c(D, D^n, \kappa(\mathcal{Q}), t) \subseteq \mathcal{DF}^s(D, D^n, \kappa(\mathcal{Q}), t)$. Now, any $s \in \mathcal{DF}^s(D, D^n, \kappa(\mathcal{Q}), t)$ can be written as $s = s' \cup \{t\}$. From the S-minimality of S-repairs, $D \smallsetminus (s' \cup \{t\}) \models \kappa(\mathcal{Q})$, but $D \smallsetminus s' \models \neg\kappa(\mathcal{Q})$, i.e. $D \smallsetminus (s' \cup \{t\}) \not\models \mathcal{Q}$, but $D \smallsetminus s' \models \mathcal{Q}$. So, t is an actual cause for $\mathcal{Q}$ with contingency set s'.

▶ **Proposition 2.** *Given $D = D^n \cup D^x$, and a BCQ $\mathcal{Q}$, $t \in D^n$ is an actual cause for $\mathcal{Q}$ iff $\mathcal{DF}^s(D, D^n, \kappa(\mathcal{Q}), t) \neq \emptyset$.* ◀

▶ **Proposition 3.** *Given $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, and $t \in D^n$:*

(a) *If $\mathcal{DF}^s(D, D^n, \kappa(\mathcal{Q}), t) = \emptyset$, then $\rho(t) = 0$.*

(b) *Otherwise, $\rho(t) = \frac{1}{|s|}$, where $s \in \mathcal{DF}^s(D, D^n, \kappa(\mathcal{Q}), t)$ and there is no $s' \in \mathcal{DF}^s(D, D^n, \kappa(\mathcal{Q}), t)$ such that $|s'| < |s|$.* ◀

▶ **Corollary 4.** *Given $D = D^n \cup D^x$, and a BCQ $\mathcal{Q}$, $t \in D^n$ is a most responsible actual cause for $\mathcal{Q}$ iff $\mathcal{DF}^c(D, D^n, \kappa(\mathcal{Q}), t) \neq \emptyset$.* ◀

▶ **Example 5.** (ex. 1 cont.) Consider the same instance D and query $\mathcal{Q}$. In this case, the DC $\kappa(\mathcal{Q})$ is, in Datalog notation, a negative rule: $\leftarrow S(x), R(x, y), S(y)$.

Here, $\mathit{Srep}(D, \kappa(\mathcal{Q})) = \{D_1, D_2, D_3\}$ and $\mathit{Crep}(D, \kappa(\mathcal{Q})) = \{D_1\}$, with $D_1 = \{R(a_4, a_3), R(a_2, a_1), R(a_3, a_3), S(a_4), S(a_2)\}$, $D_2 = \{R(a_2, a_1), S(a_4), S(a_2), S(a_3)\}$, $D_3 = \{R(a_4, a_3), R(a_2, a_1), S(a_2), S(a_3)\}$.

For tuple $R(a_4, a_3)$, $\mathcal{DF}^s(D, D, \kappa(\mathcal{Q}), R(a_4, a_3)) = \{D \smallsetminus D_2\} = \{\{R(a_4, a_3), R(a_3, a_3)\}\}$, which, by Propositions 2 and 3, confirms that $R(a_4, a_3)$ is an actual cause, with responsibility $\frac{1}{2}$. For tuple $S(a_3)$, $\mathcal{DF}^s(D, D, \kappa(\mathcal{Q}), S(a_3)) = \{D \smallsetminus D_1\} = \{S(a_3)\}$. So, $S(a_3)$ is an actual cause with responsibility 1. Similarly, $R(a_3, a_3)$ is an actual cause with responsibility $\frac{1}{2}$, because $\mathcal{DF}^s(D, D, \kappa(\mathcal{Q}), R(a_3, a_3)) = \{D \smallsetminus D_2,\ D \smallsetminus D_3\} = \{\{R(a_4, a_3), R(a_3, a_3)\}, \{R(a_3, a_3), S(a_4)\}\}$.

It holds $\mathcal{DF}^s(D, D, \kappa(\mathcal{Q}), S(a_2)) = \mathcal{DF}^s(D, D, \kappa(\mathcal{Q}), R(a_2, a_1)) = \emptyset$, because all repairs contain $S(a_2)$, $R(a_2, a_1)$. This means they do not participate in the violation of $\kappa(\mathcal{Q})$ or contribute to make $\mathcal{Q}$ true. So, they are not actual causes for $\mathcal{Q}$, confirming the result in Example 1.

$\mathcal{DF}^c(D, D, \kappa(\mathcal{Q}), S(a_3)) = \{S(a_3)\}$. From Corollary 4, $S(a_3)$ is the most responsible cause. ◀

▶ **Remark 6.** *The results in this section can be easily extended to unions of BCQs without built-ins, i.e. essentially FO monotone queries without built-ins. This can be done by associating a DC to each disjunct of the query, and considering the corresponding problems for database repairs wrt. several DCs (cf. Section 4.1).* ◀

4 Database Repairs From Actual Causes

We now characterize repairs for inconsistent databases wrt. *a set of* DCs in terms of actual causes, and reduce their computation to computation of causes. Consider an instance D for schema $\mathcal{S}$, and a set of DCs Σ on $\mathcal{S}$. For each $\kappa \in \Sigma$, of the form $\kappa\colon \leftarrow A_1(\bar{x}_1), \ldots, A_n(\bar{x}_n)$, consider its associated *violation view* defined by a BCQ, namely $V^\kappa\colon \exists \bar{x}(A_1(\bar{x}_1) \wedge \cdots \wedge A_n(\bar{x}_n))$. Next, consider the query obtained as the union of the individual violation views: $V^\Sigma := \bigvee_{\kappa \in \Sigma} V^\kappa$, a *union of* BCQs (UBCQs). Clearly, D violates (is inconsistent wrt.) Σ iff $D \models V^\Sigma$. It is easy to verify that D is consistent wrt. Σ iff $\mathcal{CS}(D, \emptyset, V^\Sigma) = \emptyset$, i.e. there are no actual causes for V^Σ to be true when all tuples are endogenous.

Now, let us collect all *S-minimal contingency sets* associated with an actual cause t for V^Σ:

$$\mathcal{CT}(D, D^n, V^\Sigma, t) \;:=\; \{s \subseteq D^n \mid D \smallsetminus s \models V^\Sigma,\ D \smallsetminus (s \cup \{t\}) \not\models V^\Sigma, \text{ and } \forall s'' \subsetneqq s,\ D \smallsetminus (s'' \cup \{t\}) \models V^\Sigma\}. \tag{3}$$

Notice that for $s \in \mathcal{CT}(D, D^n, V^\Sigma, t)$, $t \notin s$. If $t \in \mathcal{CS}(D, \emptyset, V^\Sigma)$ and $s \in \mathcal{CT}(D, D^n, V^\Sigma, t)$, from the definition of actual cause and the S-minimality of s, its holds that $s'' = s \cup \{t\}$ is an S-minimal subset of D with $D \smallsetminus s'' \not\models V^\Sigma$. So, $D \smallsetminus s''$ is an S-repair for D. Then, the following holds.

▶ **Proposition 7.** For an instance D and a set DCs Σ, $D' \subseteq D$ is an S-repair for D wrt. Σ iff, for every $t \in D \smallsetminus D'$: $t \in \mathcal{CS}(D, \emptyset, V^\Sigma)$ and $D \smallsetminus (D' \cup \{t\}) \in \mathcal{CT}(D, D, V^\Sigma, t)$. ◀

To establish a connection between most responsible actual causes and C-repairs, collect the most responsible actual causes for V^Σ:

$$\mathcal{MRC}(D, V^\Sigma) \;:=\; \{t \in D \mid t \in \mathcal{CS}(D, \emptyset, V^\Sigma),\ \nexists t' \in \mathcal{CS}(D, \emptyset, V^\Sigma) \text{ with } \rho(t') > \rho(t)\}.$$

▶ **Proposition 8.** *For instance D and set of DCs Σ, $D' \subseteq D$ is a C-repair for D wrt. Σ iff, for every $t \in D \smallsetminus D'$: $t \in \mathcal{MRC}(D, V^\Sigma)$ and $D \smallsetminus (D' \cup \{t\}) \in \mathcal{CT}(D, D, V^\Sigma, t)$.* ◀

Actual causes for V^Σ, with their contingency sets, account for the violation of some $\kappa \in \Sigma$. Removing those tuples from D should remove the inconsistency. From Propositions 7 and 8 we obtain:

▶ **Corollary 9.** *Given an instance D and a set DCs Σ, the instance obtained from D by removing an actual cause, resp. a most responsible actual cause, for V^Σ together with any of its S-minimal, resp. C-minimal, contingency sets forms an S-repair, resp. a C-repair, for D wrt. Σ.* ◀

▶ **Example 10.** Consider $D = \{P(a), P(e), Q(a, b), R(a, c)\}$ and $\Sigma = \{\kappa_1, \kappa_2\}$, with $\kappa_1\colon \leftarrow P(x), Q(x, y)$ and $\kappa_2\colon \leftarrow P(x), R(x, y)$. The violation views are $V^{\kappa_1}\colon \exists xy(P(x) \wedge Q(x, y))$ and $V^{\kappa_2}\colon \exists xy(P(x) \wedge R(x, y))$. For $V^\Sigma := V^{\kappa_1} \vee V^{\kappa_2}$, $D \models V^\Sigma$. D is inconsistent wrt. Σ.

With all tuples endogenous, $\mathcal{CS}(D, \emptyset, V^\Sigma) = \{P(a), Q(a, b), R(a, c)\}$. Its elements are associated with sets of S-minimal contingency sets: $\mathcal{CT}(D, D, V^\Sigma, Q(a, b)) = \{\{R(a, c)\}\}$,

$\mathcal{CT}(D, D, V^{\Sigma}, R(a,c)) = \{\{Q(a,b)\}\}$, $\mathcal{CT}(D, D, V^{\Sigma}, P(a)) = \{\emptyset\}$. From Corollary 9, and $\mathcal{CT}(D, D, V^{\Sigma}, R(a,c))$, $D_1 = D \smallsetminus (\{R(a,c)\} \cup \{Q(a,b)\}) = \{P(a), P(e)\}$ is an S-repair. So is $D_2 = D \smallsetminus (\{P(a)\} \cup \emptyset) = \{P(e), Q(a,b), R(a,c)\}$. These are the only S-repairs.

Furthermore, $\mathcal{MRC}(D, V^{\Sigma}) = \{P(a)\}$. From Corollary 9, D_2 is also a C-repair for D. ◀

An actual cause t with any of its S-minimal contingency sets determines a unique S-repair. The last example shows that, with different combinations of a cause and one of its contingency sets, we may obtain the same repair (e.g. for the first two $\mathcal{CT}$s). So, we may have more minimal contingency sets than minimal repairs. However, we may still have exponentially many minimal contingency sets, so as we may have exponentially many minimal repairs.

▶ **Example 11.** Consider $D = \{R(1,0), R(1,1), \ldots, R(n,0), R(n,1), S(1), S(0)\}$ and the DC $\kappa\colon \leftarrow R(x,y), R(x,z), S(y), S(z)$. D is inconsistent wrt. κ. There are exponentially many S-repairs of D: $D' = D \smallsetminus \{S(0)\}$, $D'' = D \smallsetminus \{S(1)\}$, $D_1 = D \smallsetminus \{R(1,0), \ldots, R(n,0)\}$, ..., $D_{2^n} = D \smallsetminus \{R(1,1), \ldots, R(n,1)\}$. The C-repairs are only D' and D''.

For the BCQ V^{κ} associated to κ, $D \models V^{\kappa}$, and $S(1)$ and $S(0)$ are actual causes for V^{κ} (courterfactual causes with responsibility 1). All tuples in R are actual causes, each with exponentially many S-minimal contingency sets. For example, $R(1,0)$ has the S-minimal contingency set $\{R(2,0), \ldots, R(n,0)\}$, among exponentially many others (any set built with just one element from each of the pairs $\{R(2,0), R(2,1)\}$, ..., $\{R(n,0), R(n,1)\}$ is one). ◀

The characterization results obtained so far extend those in [49] for single DCs.

4.1 Causes for unions of conjunctive queries

If we want to compute repairs wrt. sets of DCs from causes for UBCQs using, say Corollary 9, we first need an algorithm for computing the actual causes and their (minimal) contingency sets for UBCQs. These algorithms could be used as a first stage for the computation of S-repairs and C-repairs wrt. sets of DCs. However, these algorithms (cf. Section 4.2) are also interesting *per se*.

The PTIME algorithm for computing actual causes in [41] is for single conjunctive queries, but does not compute the actual causes' contingency sets. Actually, doing the latter increases the complexity, because *deciding responsibility*[5] of actual causes is *NP*-hard [41] (which would be tractable if we could efficiently compute all (minimal) contingency sets).[6] In principle, an algorithm for responsibilities can be used to compute C-minimal contingency sets, by iterating over all candidates, but Example 11 shows that there can be exponentially many of them.

We first concentrate on the problem of computing actual causes for UBCQs, without their contingency sets, which requires some notation.

▶ **Definition 12.** Given $\mathcal{Q} = C_1 \vee \cdots \vee C_k$, with each C_i a BCQ, and an instance D:

(a) $\mathfrak{S}(D)$ is the collection of all S-minimal subsets of D that satisfy a disjunct C_i of $\mathcal{Q}$;

(b) $\mathfrak{S}^n(D)$ consists of the S-minimal subsets s of D^n for which there exists a $s' \in \mathfrak{S}(D)$ with $s \subseteq s'$ and $s \smallsetminus s' \subseteq D^x$. ◀

$\mathfrak{S}^n(D)$ contains all S-minimal sets of endogenous tuples that simultaneously (and possibly accompanied by exogenous tuples) make the query true. It is easy to see that $\mathfrak{S}(D)$ and

[5] For a precise formulation, see Definition 31.

[6] Actually, [41] presents a PTIME algorithm for computing responsibilities for a restricted class of CQs.

$\mathfrak{S}^n(D)$ can be computed in polynomial time in the size of D. Now, generalizing a result for CQs in [41], actual causes for a UBCQs can be computed in PTIME in the size of D without computing contingency sets.

▶ **Proposition 13.** *Given $D = D^x \cup D^n$ and a UBCQ $\mathcal{Q}$.*

(a) *t is an actual cause for $\mathcal{Q}$ iff there is $s \in \mathfrak{S}^n(D)$ with $t \in s$.*

(b) *The decision problem (about membership of) $CPD := \{(D^x, D^n, t) \mid t \in D^n,$ and $t \in \mathcal{CS}(D^n, D^x, \mathcal{Q})\}$ belongs to PTIME.* ◀

▶ **Example 14.** (ex. 10 cont.) Consider the query $\mathcal{Q}$: $\exists xy(P(x) \wedge Q(x,y)) \vee \exists xy(P(x) \wedge R(x,y))$, and assume that for D, $D^n = \{P(a), R(a,c)\}$ and $D^x = \{P(e), Q(a,b)\}$. It holds $\mathfrak{S}(D) = \{\{P(a), Q(a,b)\}, \{P(a), R(a,c)\}\}$. Since $\{P(a)\} \subseteq \{P(a), R(a,c)\}$, $\mathfrak{S}^n(D) = \{\{P(a)\}\}$. So, $P(a)$ is the only actual cause for $\mathcal{Q}$. ◀

4.2 Contingency sets for unions of conjunctive queries

It is possible to develop a (naive) algorithm that accepts as input an instance $D = D^n \cup D^x$, and a UBCQ $\mathcal{Q}$, and returns $CS(D, D^n, \mathcal{Q})$, and also, for each $t \in CS(D, D^n, \mathcal{Q})$, its (set of) S-minimal contingency sets $\mathcal{CT}(D, D^n, \mathcal{Q}, t)$. The basis for the algorithm is a correspondence between the actual causes for $\mathcal{Q}$ with their contingency sets and a *hitting-set problem*.[7]

More precisely, for a fixed UBCQ $\mathcal{Q}$, consider the *hitting-set framework* $\mathfrak{H}^n(D) = \langle D^n, \mathfrak{S}^n(D)\rangle$, with $\mathfrak{S}^n(D)$ as in Definition 12. Different decision problems can be imposed on it. The S-minimal *hitting sets* (HSs) for $\mathfrak{H}^n(D)$ correspond to actual causes with their S-minimal contingencies for $\mathcal{Q}$. Most responsible causes for $\mathcal{Q}$ are in correspondence with minimum hitting sets for $\mathfrak{H}^n(D)$. Notice that these hitting sets are all subsets of D^n.

▶ **Proposition 15.** *For $D = D^x \cup D^n$ and a UBCQ $\mathcal{Q}$, it holds:*

(a) *t is an actual cause for $\mathcal{Q}$ with S-minimal contingency set Γ iff $\Gamma \cup \{t\}$ is an S-minimal HS for $\mathfrak{H}^n(D)$.*

(b) *t is a most responsible actual cause for $\mathcal{Q}$ with C-minimal contingency set Γ iff $\Gamma \cup \{t\}$ is a minimum HS for $\mathfrak{H}^n(D)$.* ◀

▶ **Example 16.** (ex. 10 and 14 cont.) D and $\mathcal{Q}$ are as before, but we now all tuples are endogenous. Here, $\mathfrak{S}(D) = \mathfrak{S}^n(D) = \{\{P(a), Q(a,b)\}, \{P(a), R(a,c)\}\}$. $\mathfrak{H}^n(D)$ has two S-minimal HSs: $H_1 = \{P(a)\}$ and $H_2 = \{Q(a,b), R(a,c)\}$. Each of them implicitly contains an actual cause (any of its elements) with an S-minimal contingency set (what's left after removing the actual cause). H_1 is also the C-minimal hitting set, and contains the most responsible actual cause, $P(a)$. ◀

▶ **Remark 17.** *For $\mathfrak{H}^n(D) = \langle D^n, \mathfrak{S}^n(D)\rangle$, $\mathfrak{S}^n(D)$ can be computed in PTIME, and its elements are bounded in size by $|\mathcal{Q}|$, which is the maximum number of atoms in one of $\mathcal{Q}$'s disjuncts. This is a special kind of hitting-set problems. For example, deciding if there is a hitting set of size at most k as been called the d-*hitting-set problem *[44], and d is the bound on the size of the sets in the set class. In our case, d would be $|\mathcal{Q}|$.* ◀

[7] If $\mathcal{C}$ is a collection of non-empty subsets of a set S, a subset $S' \subseteq S$ is a *hitting set* for $\mathcal{C}$ if, for every $C \in \mathcal{C}$, $C \cap S' \neq \emptyset$. S' is an S-minimal HS if no proper subset of it is also an HS. S is a minimum HS is it has minimum cardinality.

4.3 Causality, repairs and consistent answers

Corollary 9 and Proposition 15 can be used to compute repairs. If the classes of S- and C-minimal HSs for $\mathfrak{H}^n(D)$ (with $D^n = D$) are available, computing S- and C-repairs will be in PTIME in the sizes of those classes. However, it is well known that computing minimal HSs is a complex problem. Actually, as Example 11 implicitly shows, we can have exponentially many of them in $|D|$; so as exponentially many minimal repairs for a D wrt. a denial constraint.[8] So, the complexity of contingency sets computation is in line with the complexities of computing hitting sets and repairs.

The computation of causes, contingency sets, and most responsible causes via minimal/minimum HS computation can then be used to compute repairs and decide about repair questions. Since the HS problems in our case are of the d-hitting set kind, good algorithms and approximations for the latter (cf. Section 6.1) could be used in the context of repairs (all this via Corollary 9 and Proposition 15).

Consider an instance D (with all tuples endogenous) and a set Σ of DCs. For the disjunctive violation view V^Σ, the following result is obtained from Propositions 7 and 8, and Corollary 9.

▶ **Corollary 18.** *For an instance D and set DCs Σ, it holds:*
(a) *For every $t \in \mathcal{CS}(D, V^\Sigma)$, there is an S-repair that does not contain t.*
(b) *For every $t \in \mathcal{MRC}(D, V^\Sigma)$, there is a C-repair that does not contain t.*
(c) *For every $D' \in Srep(D, \Sigma)$ and $D'' \in Crep(D, \Sigma)$, $D \smallsetminus D' \subseteq \mathcal{CS}(D, V^\Sigma)$ and $D \smallsetminus D'' \subseteq \mathcal{MRC}(D, V^\Sigma)$.* ◀

For a projection-free, and a possibly non-boolean CQ $\mathcal{Q}$, we are interested in its consistent answers from D wrt. Σ. For example, for $\mathcal{Q}(x, y, z) : R(x, y) \wedge S(y, z)$, the S-consistent (C-consistent) answers would be of the form (a, b, c), where $R(a, b)$ and $S(b, c)$ belong to all S-repairs (C-repairs) of D. From Corollary 18, (a, b, c) is an S-consistent, resp. C-consistent, answer iff $R(a, b)$ and $S(b, c)$ belong to D, but they are not actual causes, resp. most responsible actual causes, for V^Σ.

▶ **Proposition 19.** *For an instance D, a set of DCs Σ, and a projection-free CQ $\mathcal{Q}(\bar{x}) : P_1(\bar{x}_1) \wedge \cdots \wedge P_k(\bar{x}_k)$:*
(a) *$\bar{c}$ is an S-consistent answer iff, for each i, $P_i(\bar{c}_i) \in (D \smallsetminus \mathcal{CS}(D, V^\Sigma))$.*
(b) *$\bar{c}$ is a C-consistent answer iff, for each i, $P_i(\bar{c}_i) \in (D \smallsetminus \mathcal{MRC}(D, V^\Sigma))$.* ◀

▶ **Example 20.** (ex. 10 cont.) Consider $\mathcal{Q}(x) : P(x)$. We had $\mathcal{CS}(D, V^\Sigma) = \{P(a), Q(a, b), R(a, c)\}$, $\mathcal{MRC}(D, V^\Sigma) = \{P(a)\}$. Then, a is both an S- and a C-consistent answer. ◀

Notice that Proposition 19 can easily be extended to conjunction of ground atomic queries. Actually, from it we obtain the following result that will be useful later on.

▶ **Corollary 21.** *Given D, a set of DCs Σ, the ground atomic query $\mathcal{Q}$: $P(c)$ is C-consistently true if $P(c) \in D$ and it is not a most responsible cause for V^Σ.* ◀

▶ **Example 22.** For $D = \{P(a, b), R(b, c), R(a, d)\}$ and the DC $\kappa : \leftarrow P(x, y), R(y, z)$: $\mathcal{CS}(D, V^\kappa) = \mathcal{MRC}(D, V^\kappa) = \{P(a, b), R(b, c)\}$. From Proposition 19, the ground atomic query $\mathcal{Q}$: $R(a, d)$ is both S- and C-consistently true in D wrt. κ, because, $D \smallsetminus \mathcal{CS}(D, V^\kappa) = D \smallsetminus \mathcal{MRC}(D, V^\kappa) = \{R(a, d)\}$. ◀

[8] An example of this kind for FDs is given in [4]. However, FDs form a special class of DCs that involve equality. Consequently, their violation views involve inequality.

The CQs considered in Proposition 19 and its Corollary 21 are not the particularly interesting, but will use those results to obtain relevant results for causality later on, e.g. Theorem 41.

5 Diagnosis: Query Answer Causality and Repairs

Let $D = D^n \cup D^x$ be an instance for schema $\mathcal{S}$, and $\mathcal{Q} : \exists\bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$, a BCQ. Assume $\mathcal{Q}$ is, possibly unexpectedly, true in D. So, for the associated DC $\kappa(\mathcal{Q}) : \forall\bar{x}\neg(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$, $D \not\models \kappa(\mathcal{Q})$. $\mathcal{Q}$ is our *observation*, for which we want to find explanations, using a consistency-based diagnosis approach.

For each predicate $P \in \mathcal{P}$, we introduce predicate Ab_P, with the same arity as P. A tuple in its extension is *abnormal* for P. The "system description", SD, includes, among other elements, the original database, expressed in logical terms, and the DC being true "under normal conditions". More precisely, we consider the following *diagnosis problem*, $\mathcal{M} = (SD, D^n, \mathcal{Q})$, associated to $\mathcal{Q}$. The FO system description, SD, contains the following elements:

(a) $Th(D)$, which is Reiter's logical reconstruction of D as a FO theory [48] (cf. Example 23).

(b) Sentence $\kappa(\mathcal{Q})^{Ab}$, which is $\kappa(\mathcal{Q})$ rewritten as follows:

$$\kappa(\mathcal{Q})^{Ab}: \ \forall\bar{x}\neg(P_1(\bar{x}_1) \wedge \neg Ab_{P_1}(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m) \wedge \neg Ab_{P_m}(\bar{x}_m)). \tag{4}$$

This formula can be refined by applying the abnormality predicate, Ab, to endogenous tuples only. For this we need to use additional auxiliary predicates End_P, with the same arity of $P \in \mathcal{S}$, which contain the endogenous tuples in P's extension (see Example 23).

(c) The inclusion dependencies: $\forall\bar{x}(Ab_P(\bar{x}) \rightarrow P(\bar{x}))$, $\forall\bar{x}(End_P(\bar{x}) \rightarrow P(\bar{x}))$, and $\forall\bar{x}(Ab_P(\bar{x}) \rightarrow End_P(\bar{x}))$, for each $P \in \mathcal{P}$.

The last entry, $\mathcal{Q}$, in $\mathcal{M}$ is the *observation*, which together with SD will produce and inconsistent theory, because we make the initial and explicit assumption that all the abnormality predicates are empty (equivalently, that all tuples are normal), i.e. we consider, for each predicate P, the sentence[9]

$$\forall\bar{x}(Ab_P(\bar{x}) \rightarrow \mathbf{false}), \tag{5}$$

where, **false** is a propositional atom that is always false. Actually, the second entry in $\mathcal{M}$ tells us how we can restore consistency, namely by (minimally) changing the abnormality condition on tuples in D^n. In other words, the rules (5) are subject to qualifications: some endogenous tuples may be abnormal. Each diagnosis shows an S-minimal set of endogenous tuples that are abnormal.

▶ **Example 23.** (ex. 1 cont.) For the instance $D = \{S(a_3), S(a_4), R(a_4, a_3)\}$, with $D^n = \{S(a_4), S(a_3)\}$, consider the diagnostic problem $\mathcal{M} = (SD, \{S(a_4), S(a_3)\}, \mathcal{Q})$, with SD containing:

(a) Predicate completion axioms: $\forall xy(R(x,y) \leftrightarrow x = a_4 \wedge y = a_3)$, $\forall x(S(x) \leftrightarrow x = a_3 \vee x = a_4)$, $\forall xy(End_R(x,y) \leftrightarrow \mathbf{false})$, $\forall x(End_S(x) \leftrightarrow x = a_3 \vee x = a_4)$.
Unique names assumption: $a_4 \neq a_3$.

(b) $\kappa(\mathcal{Q})^{Ab}$: $\forall xy\neg(S(x) \wedge End_S(x) \wedge \neg Ab_S(x) \wedge R(x,y) \wedge End_R(x,y) \wedge \neg Ab_R(x,y) \wedge S(y) \wedge \neg Ab_S(y))$.

[9] Notice that these can also be seen as DCs, since they can be written as $\forall\bar{x}\neg Ab_P(\bar{x})$.

(c) $\forall xy(Ab_R(x,y) \rightarrow R(x,y))$, $\forall x(Ab_S(x) \rightarrow S(x))$, $\forall xy(End_R(x,y) \rightarrow R(x,y))$, $\forall x(End_S(x) \rightarrow S(x))$, $\forall xy(Ab_R(x,y) \rightarrow End_R(x,y))$, $\forall x(Ab_S(x) \rightarrow End_S(x))$.
The normality assumptions for tuples are: $\forall xy(Ab_R(x,y) \rightarrow \mathbf{false})$, $\forall x(Ab_S(x) \rightarrow \mathbf{false})$. ◀

Now, the observation is $\mathcal{Q}$ (being true), obtained by evaluating query $\mathcal{Q}$ on (theory of) D. In this case, $D \not\models \kappa(\mathcal{Q})$. Since all the abnormality predicates are assumed to be empty, $\kappa(\mathcal{Q})$ is equivalent to $\kappa(\mathcal{Q})^{Ab}$, which also becomes false wrt D. As a consequence, $SD \cup \{(5)\} \cup \{\mathcal{Q}\}$ is an inconsistent FO theory. A diagnosis is a set of endogenous tuples that, by becoming abnormal, restore consistency.

▶ **Definition 24.**
(a) A *diagnosis* for $\mathcal{M}$ is a $\Delta \subseteq D^n$, such that $SD \cup \{Ab_P(\bar{c}) \mid P(\bar{c}) \in \Delta\} \cup \{\neg Ab_P(\bar{c}) \mid P(\bar{c}) \in D \smallsetminus \Delta\} \cup \{\mathcal{Q}\}$ is consistent.
(b) $\mathcal{D}(\mathcal{M}, t)$ denotes the set of S-minimal diagnoses for $\mathcal{M}$ that contain a tuple $t \in D^n$. (c) $\mathcal{MCD}(\mathcal{M}, t)$ denotes the set of C-minimal diagnoses in $\mathcal{D}(\mathcal{M}, t)$. ◀

By definition, $\mathcal{MCD}(\mathcal{M}, t) \subseteq \mathcal{D}(\mathcal{M}, t)$. Diagnoses for $\mathcal{M}$ and actual causes for $\mathcal{Q}$ are related.

▶ **Proposition 25.** *Consider $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, and the diagnosis problem $\mathcal{M}$ associated to $\mathcal{Q}$. Tuple $t \in D^n$ is an actual cause for $\mathcal{Q}$ iff $\mathcal{D}(\mathcal{M}, t) \neq \emptyset$.* ◀

The responsibility of an actual cause t is determined by the cardinality of the diagnoses in $\mathcal{MCD}(\mathcal{M}, t)$.

▶ **Proposition 26.** *For $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, the associated diagnosis problem $\mathcal{M}$, and a tuple $t \in D^n$, it holds:*
(a) *$\rho_D(t) = 0$ iff $\mathcal{MCD}(\mathcal{M}, t) = \emptyset$.*
(b) *Otherwise, $\rho_D(t) = \frac{1}{|s|}$, where $s \in \mathcal{MCD}(\mathcal{M}, t)$.* ◀

▶ **Example 27.** (ex. 23 cont.) $\mathcal{M}$ has two diagnosis: $\Delta_1 = \{S(a_3)\}$ and $\Delta_4 = \{S(a_4)\}$. Here, $\mathcal{D}(\mathcal{M}, S(a_3)) = \mathcal{MCD}(\mathcal{M}, S(a_3)) = \{\{S(a_3)\}\}$ and $\mathcal{D}(\mathcal{M}, S(a_4)) = \mathcal{MCD}(\mathcal{M}, S(a_4)) = \{\{S(a_4)\}\}$. From Propositions 25 and 26, $S(a_3)$ and $S(a_4)$ are actual cases, with responsibility 1. ◀

In consistency-based diagnosis, minimal diagnoses can be obtained as S-minimal HSs of the collection of S-minimal *conflict sets* (cf. Section 2) [47]. In our case, conflict sets are S-minimal sets of endogenous tuples that, if not abnormal (only endogenous ones can be abnormal), and together, and possibly in combination with exogenous tuples, make (4) false. It is easy to verify that the conflict sets of $\mathcal{M}$ coincide with the sets in $\mathfrak{S}(D^n)$ (cf. Definition 12 and Remark 17). As a consequence, conflict sets for $\mathcal{M}$ can be computed in PTIME, the HSs for $\mathcal{M}$ contain actual causes for $\mathcal{Q}$, and the HS problem for the diagnosis problems is of the d-hitting-set kind. The connection between consistency-based diagnosis and causality allows us, in principle, to apply techniques for the former, e.g. [25, 43], to the latter.

▶ **Example 28.** (ex. 23 cont.) The diagnosis problem $\mathcal{M} = (SD, \{S(a_4), S(a_3)\}, \mathcal{Q})$ gives rise to the HS framework $\mathfrak{H}^n(D) = \langle \{S(a_4), S(a_3)\}, \{\{(S(a_3), S(a_4)\}\}\rangle$, with $\{S(a_3), S(a_4)\}$ corresponding to the conflict set $c = \{S(a_4), S(a_3)\}$. $\mathfrak{H}^n(D)$ has two minimum HSs: $\{S(a_3)\}$ and $\{S(a_4)\}$, which are the S-minimal diagnosis for $\mathcal{M}$. Then, the two tuples are actual causes for $\mathcal{Q}$ (cf. Proposition 25). From Proposition 26, $\rho_D(S(a_3)) = \rho_D(S(a_4)) = 1$. ◀

The solutions to the diagnosis problem can be used for computing repairs.

▶ **Proposition 29.** *Consider a database instance D with only endogenous tuples, a set of DCs of the form κ: $\forall \bar{x} \neg(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m)$, and their associated "abnormality" integrity constraints*[10] *in (4) (in this case we do not need End_P atoms). Each S-minimal diagnosis Δ gives rise to an S-repair of D, namely $D_\Delta = D \smallsetminus \{P(\bar{c}) \in D \mid Ab_P(\bar{c}) \in \Delta\}$; and every S-repair can be obtained in this way. Similarly, for C-repairs using C-minimal diagnoses.* ◀

▶ **Example 30.** (ex. 27 cont.) The instance $D = \{S(a_3), S(a_4), R(a_4, a_3)\}$ has three (both S- and C-) repairs wrt. the DC κ: $\forall xy \neg(S(x) \wedge R(x, y) \wedge S(y))$, namely $D_1 = \{S(a_3)\}$, $D_2 = \{S(a_4)\}$, and $D_3 = \{R(a_4, a_3)\}$. They can be obtained as $D_{\Delta_1}, D_{\Delta_2}, D_{\Delta_3}$ from the only (S- and C-) diagnoses, $\Delta_1 = \{S(a_3)\}$, $\Delta_4 = \{S(a_4)\}$, $\Delta_3 = \{R(a_4, a_3)\}$, resp. ◀

The kind of diagnosis problem we introduced above can be formulated as a *preferred-repair problem* [9, sec. 2.5] (see [51] for a general approach to prioritized repairs). For this, it is good enough to materialize tables for the auxiliary predicates Ab_P and End_P, and consider the DCs of the form (4) (with the End_P atoms if not all tuples are endogenous), plus the DCs (5). The initial extensions for the Ab_P predicates are empty. If D is inconsistent wrt. this set of DCs, the S-repairs that are obtained by only *inserting* endogenous tuples into the extensions of the Ab_P predicates correspond to S-minimal diagnosis, and each S-minimal diagnosis can be obtained in this way.

6 Complexity Results

There are three main computational problems in database causality. For a BCQ $\mathcal{Q}$ and database D, they are:

(a) The *causality problem* (CP) that is about computing the actual causes for $\mathcal{Q}$.

(b) The *responsibility problem* (RP) that is about computing the responsibility $\rho_D(t)$ of a given actual cause t. Since a tuple that is not an actual cause has responsibility 0, the latter problem subsumes the former.

(c) Computing the *most responsible actual causes* (MRC).

These problems have corresponding *decision versions*. Both CP and its decision version, CPD, are solvable in polynomial time [41], which can be extended to UBCQs (cf. Proposition 13). We consider the decision version of the second problem.

▶ **Definition 31.** For a BCQ $\mathcal{Q}$, the *responsibility decision problem* (RPD) is (deciding about membership of) $\mathcal{RPD}(\mathcal{Q}) = \{(D^x, D^n, t, v) \mid t \in D^n, v \in \{0\} \cup \{\frac{1}{k} \mid k \in \mathbb{N}^+\}$, $D := D^x \cup D^n \models \mathcal{Q}$ and $\rho_D(t) > v\}$. ◀

The complexity analysis of RPD in [41] is restricted to conjunctive queries without self-joins, for which a dichotomy result holds: depending on the syntactic structure of a query, RPD is either in PTIME or is NP-hard. Here, we generalize the complexity analysis for RPD to general CQs.

We will also investigate the decision version, MRCD, of MRC, i.e. about deciding most responsible actual causes. This is a natural problem, because actual causes with the highest responsibility tend to provide most interesting explanations for query answers [41, 42].

▶ **Definition 32.** For a BCQ $\mathcal{Q}$, the *most responsible cause decision problem* is $\mathcal{MRCD}(\mathcal{Q})$ $= \{(D^x, D^n, t) \mid t \in D^n$ and $0 < \rho_D(t)$ is a maximum for $D := D^x \cup D^n\}$. ◀

[10] Notice that these are not denial constraints.

We start by analyzing a more basic decision problem: *S-minimal contingency checking* (MCCD).

▶ **Definition 33.** For a BCQ $\mathcal{Q}$, $\mathcal{MCCD}(\mathcal{Q}) := \{(D^x, D^n, t, \Gamma) \mid \Gamma \in \mathcal{CT}(D^n \cup D^x, D^n, \mathcal{Q}, t)\}$ (cf. (3)). ◀

Due to the results in Sections 3 and 4, it clear that there is a close connection between MCCD and the *S-repair checking* problem in consistent query answering [9, chap. 5], about deciding if instance D' is an S-repair of instance D wrt. a set of integrity constraints. Actually, the following result is obtained from the membership of the S-repair checking problem of LOGSPACE for DCs [1, prop. 5].

▶ **Proposition 34.** *For a BCQ $\mathcal{Q}$, $\mathcal{MCCD}(\mathcal{Q}) \in PTIME$.* ◀

We could also consider the decision problem defined as in Definition 33, but with C-minimal Γ. We will not use results about this problem in the following. Furthermore, its connection with the C-repair checking problem is less direct. As one can see from Section 3, C-minimal contingency sets correspond to a repair semantics somewhere between the S-minimal and C-minimal repair semantics (a subclass of *Srep*, but a superclass of *Crep*): It is about an S-minimal repair with minimum cardinality that does not contain a particular tuple.

Now we establish that RPD is *NP*-complete for CQs in general. The *NP*-hardness is shown in [41]. Membership of *NP* is obtained using Proposition 34.

▶ **Theorem 35.**
(a) *For every BCQ $\mathcal{Q}$, $\mathcal{RPD}(\mathcal{Q}) \in NP$.*
(b) *[41] There are CQs $\mathcal{Q}$ for which $\mathcal{RPD}(\mathcal{Q})$ is NP-hard.* ◀

In order to better understand the complexity of the problem, RP, of computing responsibility, we will investigate the *functional*, non-decision version of the problem.

The main source of complexity when computing responsibilities is related to the hitting-set problem associated to $\mathfrak{H}^n(D) = \langle D^n, \mathfrak{S}^n(D)\rangle$ in Remark 17. In this case, it is about computing the cardinality of a minimum hitting set that contains a given vertex (tuple) t. That this is a kind of *d-hitting-set problem* [44] will be useful in Section 6.1.

Our responsibility problem can also be seen as a *vertex cover problem* on the *hypergraph* $\mathfrak{G}^n(D) = \langle D^n, \mathfrak{E}^n(D)\rangle$ associated to $\mathfrak{H}^n(D) = \langle D^n, \mathfrak{S}^n(D)\rangle$. In it, the set of hyperedges $\mathfrak{E}^n(D)$ coincides with the collection $\mathfrak{S}^n(D)$. Determining the responsibility of a tuple t becomes the problem on hypergraphs of determining the size of a minimum vertex cover (VC)[11] that contains vertex t (among all VCs that contain the vertex). Again, in this problem the hyperedges are bounded by $|\mathcal{Q}|$.[12]

▶ **Example 36.** For $\mathcal{Q}: \exists xy(P(x) \wedge R(x,y) \wedge P(y))$, and $D = D^n = \{P(a), P(c), R(a,c), R(a,a)\}$, $\mathfrak{S}(D) = \mathfrak{S}^n(D) = \{\{P(a), R(a,a)\}, \{P(a), P(c), R(a,c)\}\}$. D is the set of vertices of hypergraph $\mathfrak{G}^n(D)$, and its hyperedges are $\{P(a), R(a,a)\}$, $\{P(a), P(c), R(a,c)\}$. The following are the minimal VCs: $vc_1 = \{P(a)\}$, $vc_2 = \{P(c), R(a,a)\}$, $vc_3 = \{R(a,a), R(a,c)\}$. Then, $P(a)$ is an actual cause with responsibility 1. The other tuples are actual causes with responsibility $\frac{1}{2}$. ◀

[11] A set of vertices is a VC for a hypergraph if it intersects every hyperedge. Obviously, when we talk of *minimum* VC, we are referring to minimal in cardinality.
[12] We recall that repairs of databases wrt. DCs can be characterized as maximal independent sets of *conflict hypergraphs* (conflict graphs in the case of FDs) whose vertices are the database tuples, and hyper-edges connect tuples that together violate a DC [4, 17].

To simplify the presentation, we will formulate and address our computational problems as problems for graphs (instead of hypergraphs). However, our results still hold for hypergraphs [39]. Actually, the following *representation lemma* holds.

▶ **Lemma 37.** *There is a fixed database schema $\mathcal{S}$ and a BCQ $\mathcal{Q} \in L(\mathcal{S})$, without built-ins, such that, for every graph $G = (V, E)$ and $v \in V$, there is an instance D for $\mathcal{S}$ and a tuple $t \in D$, such that the size of a minimum VC of G containing v equals the responsibility of t as an actual cause for $\mathcal{Q}$.* ◀

Having represented our responsibility problem as a graph-theoretic problem, we first consider the following *membership minimal VC problem* (MMVC): Given a graph $G = (V, E)$, a vertex $v \in V$, determine the size of a minimum VC of G that contains v.

▶ **Lemma 38.** *Given a graph G and a vertex v in it, there is a graph G' extending G that can be constructed in polynomial time in $|G|$, such that the size of a minimum VC for G that contains v and the size of a minimum VC for G' coincide.* ◀

From this lemma and the $FP^{NP(log(n))}$-completeness of determining the size of a maximum clique in a graph [35], we obtain:

▶ **Proposition 39.** *MMVC problem for graphs is $FP^{NP(log(n))}$-complete.* ◀

From Lemma 37 and Proposition 39 we obtain the complexity result for RP. Membership can also be obtained from Theorem 35.

▶ **Theorem 40.**

(a) *For every BCQ without built-ins, $\mathcal{Q}$, computing the responsibility of a tuple as a cause for $\mathcal{Q}$ is in $FP^{NP(log(n))}$.*

(b) *There is a database schema and a BCQ $\mathcal{Q}$, without built-ins, such that computing the responsibility of a tuple as a cause for $\mathcal{Q}$ is $FP^{NP(log(n))}$-complete.* ◀

Now we address the most responsible causes problem, MRCD. We use the connection with consistent query answering of Section 4.3, namely Corollary 21, and the $P^{NP(log(n))}$-completeness of consistent query answering under the C-repair semantics for queries that are conjunctions of ground atoms and a particular DC [39, theo. 4].

▶ **Theorem 41.**

(a) *For every BCQ without built-ins, $\mathcal{MRCD}(\mathcal{Q}) \in P^{NP(log(n))}$.*

(b) *There is a database schema and a BCQ $\mathcal{Q}$, without built-ins, for which $\mathcal{MRCD}(\mathcal{Q})$ is $P^{NP(log(n))}$-complete.* ◀

From Proposition 15 and the $FP^{NP(log(n))}$-completeness of determining the size of C-repairs for DCs [39, theo. 3], we obtain the following for the computation of the highest responsibility value.

▶ **Proposition 42.**

(a) *For every BCQ without built-ins, computing the responsibility of the most responsible causes is in $FP^{NP(log(n))}$.*

(b) *There is a database schema and a BCQ $\mathcal{Q}$, without built-ins, for which computing the responsibility of the most responsible causes is $FP^{NP(log(n))}$-complete.* ◀

6.1 FPT of responsibility

We need to cope with the intractability of computing most responsible causes. The area of *fixed parameter tractability* (FPT) [26] provides tools to attack this problem. In this regard, we recall that a decision problem with inputs of the form (I, p), where p is a distinguished parameter of the input, is fixed parameter tractable (or belongs to the class FPT), if it can be solved in time $O(f(|p|) \cdot |I|^c)$, where c and the hidden constant do not depend on $|p|$ or $|I|$, and f does not depend on $|I|$.

In our case, the *parameterized version of the decision problem* $\mathcal{RPD}(\mathcal{Q})$ (cf. Definition 31) is denoted with $\mathcal{RPD}^p(\mathcal{Q})$, and the distinguished parameter is k, such that $v = \frac{1}{k}$. That $\mathcal{RPD}^p(\mathcal{Q})$ belongs to FPT can be obtained from its formulation as a d-hitting-set problem (d being the fixed upper bound on the size of the sets in the set class); in this case about deciding if there is a HS that contains the given tuple t that has cardinality smaller that k. This problem belongs to FPT.

▶ **Theorem 43.** *For every BCQ $\mathcal{Q}$, $\mathcal{RPD}^p(\mathcal{Q})$ belongs to FPT, where the parameter is the inverse of the responsibility bound.* ◀

The proof of this result is interesting *per se*, and we sketch it here. First, there is a PTIME parameterized algorithm for the d-hitting-set problem about deciding if there is a HS of size at most k that runs in time $O(e^k + n)$, with n the size of the underlying set and $e = d - 1 + o(d^{-1})$ [44]. In our case, $n = |D|$, and $d = |\mathcal{Q}|$ (cf. also [24]).

Now, to decide if the responsibility of a given tuple t is greater than $v = \frac{1}{k}$, we consider the associated hypergraph $\mathfrak{G}^n(D)$, and we decide if it has a VC that contains t and whose size is less than k. In order to answer this, we use Lemma 38, and build the extended hypergraph $\mathfrak{G}'$. The size of a minimum VC for $\mathfrak{G}'$ gives the size of the minimum VC of $\mathfrak{G}^n(D)$ that contains t. If $\mathfrak{G}^n(D)$ has a VC that contains t of size less than k, then $\mathfrak{G}'$ has a VC of size less than k. If $\mathfrak{G}'$ has a VC of size less than k, its minimum size for a VC is less than k. Since this minimum is the same as the size of a minimum VC for $\mathfrak{G}^n(D)$ that contains t, $\mathfrak{G}^n(D)$ has a VC of size less than k that contains t. As a consequence, it is good enough to decide if $\mathfrak{G}'$ has a VC of size less than k. For this, we use the HS formulation of this hypergraph problem, and the already mentioned FPT algorithm.

This result and the corresponding algorithm show that the higher the required responsibility degree, the lower the computational effort needed to compute the actual causes with at least that level of responsibility. In other terms, parameterized algorithms are effective for computing actual causes with high responsibility or most responsible causes. In general, parameterized algorithms are very effective when the parameter is relatively small [26].

Now, in order to compute most responsible causes, we could apply, for each actual cause t, the just presented FPT algorithm on the hypergraph $\mathfrak{G}^n(D)$, starting with $k = 1$, i.e. asking if there is VC of size less than 1 that contains t. If the algorithm returns a positive result, then t is a counterfactual cause, and has responsibility 1. Otherwise, the algorithm will be launched with $k = 2, 3, \ldots, |D^n|$, until a positive result is returned. (The procedure can be improved through binary search on $k = 1, 2, 3, \ldots, m$, with m possibly much smaller than $|D|$.)

The complexity results and algorithms provided in this section can be extend to UBCQs. This is due to Remark 6 and the construction of $\mathfrak{S}^n(D)$, which the results in this section build upon.

For the d-hitting-set problem there are also efficient parameterized approximation algorithms [11]. They could be used to approximate the responsibility problem. Furthermore, approximation algorithms developed for the minimum VC problem on bounded hypergraphs

[31, 45] should be applicable to approximate most responsible causes for query answers. Via the causality/repair connection (cf. Section 4.3), it should be possible to develop approximation algorithms to compute S-repairs of particular sizes, C-repairs, and consistent query answers wrt. DCs.

6.2 The causality dichotomy's reflection on repairs

In [41] the class of *linear* CQs is introduced. For them, computing tuple responsibilities is tractable. Roughly speaking, a BCQ is linear if its atoms can be ordered in a way that every variable appears in a continuous sequence of atoms, e.g. $\mathcal{Q}_1 : \exists xvyu(A(x) \wedge S_1(x, v) \wedge S_2(v, y) \wedge R(y, u) \wedge S_3(y, z))$ is linear, but not $\mathcal{Q}_2 : \exists xyz(A(x) \wedge B(y) \wedge C(z) \wedge W(x, y, z))$, for which RPD is *NP*-hard [41]. The class of BCQs for which computing responsibility (more precisely, our $\mathcal{RPD}$ decision problem) is tractable can be extended to *weakly linear*.[13] Now, the dichotomy result in [41] says that for a BCQ $\mathcal{Q}$ without self-joins, RDP is tractable when $\mathcal{Q}$ is weakly-linear, but *NP*-hard, otherwise. Due to the causality/repair connection of Section 4, we can obtain the following results for database repairs.

▶ **Theorem 44.**

(a) *For single weakly-linear DCs, C-repair checking and deciding if the size of a C-repair is larger than a bound are both tractable.*[14]

(b) *For single, self-join free DCs κ, and the problem RepSize(κ) of deciding if there is a repair D' for a given input instance D and a tuple $t \in D$ with $|D'| \geq m$ and $t \notin D'$,*[15] *the following dichotomy holds: (b1) If κ is weakly-linear, RepSize(κ) is tractable. (b2) Otherwise, it is NP-complete.* ◀

This dichotomy result for repairs shows that interesting results in one of the areas (causality, in this case) have counterparts in some of the others. The form the reincarnation of the known result takes in the new area (repairs, in this case) is interesting *per se*.

Notice that both problems in (a) in Theorem 44 may be intractable even for single DCs [39]. More specifically, C-repair checking can be *coNP*-hard for single DCs [39, 1]. Actually, the single DC used in [39, lemma 4] is of the form $\kappa : \leftarrow V(x), V(y), E(x, y, z)$, whose associated BCQ is not weakly-linear. As a matter of fact, this BCQ is a *NP*-hard for RDP [41].

7 Discussion and Conclusions

In this research we have unveiled and formalized some first interesting relationships between causality in databases, database repairs, and consistency-based diagnosis. These connections allow us to apply results and techniques developed for each of them to the others. This is particularly beneficial for causality in databases, where still a limited number of results and techniques have been obtained or developed.

The connections we established here inspired complexity results for causality, e.g. Theorems 40 and 41, and were used to prove them. We appealed to several non-trivial results (and the proofs thereof) about repairs/CQA obtained in [39]. It is also the case that the well-established hitting-set approach to diagnosis inspired a similar approach to causal

[13] Computing sizes of minimum contingency sets is reduced to the max-flow/min-cut problem in a network.
[14] A DC κ is weakly-linear if the corresponding BCQ V^κ is weakly-linear. In this way any adjective that applies to BCQs can be applied to DCs.
[15] More precisely, D' is a subset of D that satisfies κ. Here, $0 \leq m \leq n = |D|$.

responsibility, which in its turn allowed us to obtain results about its fixed-parameter tractability. It is also the case that diagnostic reasoning, as a form of non-monotonic reasoning, can provide a solid foundation for causality in databases and query answer explanation, in general [15, 16].

Our work creates a theoretical basis for deeper and mathematically more complex investigations. In particular, it also opens interesting research directions, some of which are briefly discussed below.

Preferred causes for queries. In Section 3 we characterized causes and most responsible causes in terms of S-repairs and C-repairs, resp. This could be generalized by using the notion of *preferred repair* [51]. These are repairs whose minimization correspond to a *priority relationship*, $\preceq$, between instances. Let assume it defines a corresponding class of preferred repairs, $\preceq Rep$. Inspired by (1), we can define, for a BCQ $\mathcal{Q}$: $\mathcal{DF}^{\preceq}(D, D^n, \kappa(\mathcal{Q}), t) := \{D \smallsetminus D' \mid D' \in \preceq Rep(D, \kappa(\mathcal{Q})),\ t \in (D \smallsetminus D') \subseteq D^n\}$, and, $t \in D^n$ is a $\preceq$-cause iff $\mathcal{DF}^{\preceq}(D, D^n, \kappa(\mathcal{Q}), t) \neq \emptyset$. In this way, a whole class of preferences on causes can be introduced, which is natural problem [42].[16]

Endogenous repairs. The partition of a database into endogenous and exogenous tuples may also be of interest in the context of repairs. Considering that we should have more control on endogenous tuples than on exogenous ones, which may come from external sources, it makes sense to consider *endogenous repairs*. They are obtained by updates (of any kind) on endogenous tuples. For example, in the case of DCs, endogenous repairs would be obtained by deleting endogenous tuples only. If there are no repairs based on endogenous tuples, a preference condition could be imposed on repairs [54, 51], privileging those that change exogenous the least. (Of course, it could also be the other way around, that is we may feel more inclined to change exogenous tuples than our endogenous ones.)

As a further extension, it could be possible to assume that combinations of (only) exogenous tuples never violate the integrity constraints, which could be checked at upload time. In this sense, there would be a part of the database that is considered to be consistent, while the other is subject to possible repairs. (For slightly related research, see [28].)

Objections to causality. Causality as introduced by Halpern and Pearl in [29, 30], aka. HP-causality, is the basis for the notion of causality in [41]. HP-causality has been the object of some criticism [32], which is justified in some (more complex, non-relational) settings, specially due to the presence of different kinds of *logical variables* (or lack thereof). In our context the objections do not apply: variables just say that a certain tuple belongs to the instance (or not); and for relational databases the closed-world assumption applies. In [32], the definition of HP-causality is slightly modified. In our setting, this modified definition does not change actual causes or their properties.

ASP specification of causes. S-repairs can be specified by means of *answer set programs* (ASPs) [3, 6], and C-repairs too, with the use of weak program constraints [3]. This should allow for the introduction of ASPs in the context of causality, for specification and reasoning. There are also ASP-based specifications of diagnosis [23] that could be brought into a more complete picture.

[16] In [40] the possibility of introducing weights in the partition is considered, in this way imposing a form of preference on causes.

Causes and functional dependencies, and beyond. Functional dependencies are DCs with conjunctive violation views with inequality, and are still monotonic. There is much research on repairs and consistent query answering for functional dependencies, and more complex integrity constraints [9]. In causality, mostly CQs without built-ins have been considered. The repair connection could be exploited to obtain results for causality and CQs with inequality, and also other classes of queries.

View updates and abduction. Abduction [19, 22] is another form of model-based diagnosis, and is related to the subjects investigated in this work.The *view update problem*, about updating a database through views, is a classical problem in databases that has been treated through abduction [33, 20]. User knowledge imposed through view updates creates or reflects *uncertainty* about the base data, because alternative base instances may give an account of the intended view updates. The view update problem, specially in its particular form of of *deletion propagation*, has been recently related in [37, 38] to causality as introduced in [41]. (Notice only tuple deletions are used with violation views and repairs associated to DCs.)

Database repairs are also related to the view update problem. Actually, *answer set programs* (ASP) for database repairs [6] implicity repair the database by updating intentional, annotated predicates. Even more, in [8], in order to protect sensitive information, databases are explicitly and virtually "repaired" through secrecy views that specify the information that has to be kept secret. These are prioritized repairs that have been specified via ASPs. Abduction has been explicitly applied to database repairs [5]. The deep interrelations between causality, abductive reasoning, view updates and repairs are the objects of our ongoing research efforts [10].

Acknowledgments. Research funded by NSERC Discovery, and the NSERC Strategic Network on Business Intelligence (BIN). Conversations with Alexandra Meliou during Leo Bertossi's visit to U. of Washington in 2011 are much appreciated. He is also grateful to Dan Suciu and Wolfgang Gatterbauer for their hospitality. L. Bertossi is grateful to Benny Kimelfeld for stimulating conversations. Part of the research was developed by L. Bertossi at *LogicBlox* and *The Center for Semantic Web Research* (Chile). Their support is much appreciated.

References

1 Afrati , F. and Kolaitis, P. Repair Checking in Inconsistent Databases: Algorithms and Complexity. *Proc. ICDT* 2009 pp. 31-41.

2 Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. *Proc. ACM PODS*, 1999, pp. 68-79.

3 Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answers. *Theory and Practice of Logic Programming*, 2003, 3(4&5):393-424.

4 Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V. and Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296:405-434.

5 Arieli, O., Denecker, M., Van Nuffelen, B. and Bruynooghe, M. Coherent Integration of Databases by Abductive Logic Programming. *J. Artif. Intell. Res.*, 2004, 21:245-286.

6 Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics of Databases*, Springer LNCS 2582, 2003, pp. 1-27.

7 Bertossi, L. Consistent Query Answering in Databases. *ACM SIGMOD Record*, 2006, 35(2):68-76.

8 Bertossi, L. and Li, L. Achieving Data Privacy through Secrecy Views and Null-Based Virtual Updates. *IEEE Transaction on Knowledge and Data Engineering*, 2013, 25(5):987-1000.

9 Bertossi, L. *Database Repairing and Consistent Query Answering.* Morgan & Claypool, Synthesis Lectures on Data Management, 2011.

10 Bertossi, L. and Salimi, B. Unifying Causality, Diagnosis, Repairs and View-Updates in Databases. Presented at the First International Workshop on Big Uncertain Data (BUDA 2014). Posted at: arXiv:1405.4228 [cs.DB].

11 Brankovic, L., and H. Fernau, H. Parameterized Approximation Algorithms for Hitting Set. In *Approximation and Online Algorithms*, 2012, Springer LNCS 7164, pp. 63-76.

12 Buneman, P., Khanna, S. and Tan, W. C. Why and Where: A Characterization of Data Provenance. *Proc. ICDT*, 2001, pp. 316–330.

13 Buneman, P. and Tan, W. C. Provenance in Databases. *Proc. ACM SIGMOD*, 2007, pp. 1171–1173.

14 Cheney, J., Chiticariu, L. and Tan, W. C. Provenance in Databases: Why, How, And Where. *Foundations and Trends in Databases*, 2009, 1(4): 379-474.

15 Cheney, J., Chong, S., Foster, N., Seltzer, M. I. and Vansummeren, S. Provenance: A Future History. *OOPSLA Companion (Onward!)*, 2009, pp. 957–964.

16 Cheney, J. Is Provenance Logical? *Proc. LID*, 2011, pp. 2–6.

17 Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, 2005, 197(1-2):90-121.

18 Chockler, H. and Halpern, J. Y. Responsibility and Blame: A Structural-Model Approach. *J. Artif. Intell. Res.*, 2004, 22:93-115.

19 Console, L. and Torasso, P. A Spectrum of Logical Definitions of Model-Based Diagnosis. *Computational Intelligence*, 1991, 7:133-141.

20 Console, L., Sapino M. L. and Theseider-Dupre, D. The Role of Abduction in Database View Updating. *J. Intell. Inf. Syst.*, 1995, 4(3): 261-280.

21 Cui, Y., Widom, J. and Wiener, J. L. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Trans. Database Syst.*, 2000, 25(2):179-227.

22 Eiter, T., Gottlob, G. and Leone, N. Abduction from Logic Programs: Semantics and Complexity. *Theor. Comput. Sci.*, 1997, 189(1-2):129-177.

23 Eiter, Th., Faber, W., Leone, N. and Pfeifer, G. The Diagnosis Frontend of the DLV System. *AI Commun.*, 1999, 12(1-2):99-111.

24 Fernau, H. Parameterized Algorithmics for *d*-Hitting Set. *Int. J. Comput. Math.*, 2010, 87(14):3157-3174.

25 Feldman, A., Provan G., and Gemund A.V. Approximate model-based diagnosis using greedy stochastic search. *Journal of Artificial Intelligence Research (JAIR)*, 2010, 87(14):3157-3174.

26 Flum, J. and Grohe, M. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science, Springer Verlag, 2006.

27 Gertz, M. Diagnosis and Repair of Constraint Violations in Database Systems. PhD Thesis, Universität Hannover, 1996.

28 Greco, S., Pijcke, F. and Wijsen, J. Certain Query Answering in Partially Consistent Databases. *PVLDB*, 2014, 7(5):353-364.

29 Halpern, J., and Pearl, J. Causes and Explanations: A Structural-Model Approach: Part 1 *Proc. UAI*, 2001, pp. 194-202.

30 Halpern, J., and Pearl, J. Causes and Explanations: A Structural-Model Approach: Part 1. *British J. Philosophy of Science*, 2005, 56:843-887.

31 Halperin, E. Improved Approximation Algorithms for the Vertex Cover Problem in Graphs and Hyper-Graphs. *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2000, pp. 329-337.

32 Halpern, J. Appropriate Causal Models and Stability of Causation. Proc. KR'14, 2014.

33 Kakas A. C. and Mancarella, P. Database Updates through Abduction. *Proc. VLDB*, 1990, pp. 650-661.

34 Karvounarakis, G. and Green, T. J. Semiring-Annotated Data: Queries and Provenance? *SIGMOD Record*, 2012, 41(3):5-14.

35 Krentel, M. The Complexity of Optimization Problems. *J. Computer and Systems*, 1988, 36:490-509.

36 Karvounarakis, G. Ives, Z. G. and Tannen, V. Querying Data Provenance. *Proc. ACM SIGMOD*, 2010, pp. 951–962.

37 Kimelfeld, B. A Dichotomy in the Complexity of Deletion Propagation with Functional Dependencies. Proc. ACM PODS, 2012.

38 Kimelfeld, B., Vondrak, J. and Williams, R. Maximizing Conjunctive Views in Deletion Propagation. *ACM Trans. Database Syst.*, 2012, 37(4):24.

39 Lopatenko, A. and Bertossi, L. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. *Proc. ICDT*, 2007, Springer LNCS 4353, pp. 179-193. Extended version posted at: arXiv:cs/0604002 [cs.DB].

40 Meliou, A., Gatterbauer, W. and Suciu, D. Bringing Provenance to its Full Potential Using Causal Reasoning. *Proc. TaPP*, 2011.

41 Meliou, A., Gatterbauer, W. Moore, K. F. and Suciu, D. The Complexity of Causality and Responsibility for Query Answers and Non-Answers. *Proc. VLDB*, 2010, pp. 34-41.

42 Meliou, A., Gatterbauer. W., Halpern, J. Y., Koch, C., Moore K. F. and Suciu, D. Causality in Databases. *IEEE Data Eng. Bull*, 2010, 33(3):59-67.

43 Mozetic, I, Holzbaur, C. Controlling the Complexity in Model-Based Diagnosis *Annals of Mathematics and Artificial Intelligence*, 1994, 11(1-4): 297-314.

44 Niedermeier, R. and Rossmanith, P. An efficient fixed-parameter algorithm for 3-hitting set. In *J. Discrete Algorithms*, 2003 1(1):89-102.

45 Okun, M. On Approximation of the Vertex Cover Problem in Hypergraphs. In *Discrete Optimization*, 2005,2(1):101-111.

46 Papadimitriou, Ch. *Computational Complexity.* Addison-Wesley, 1994.

47 Reiter, R. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 1987, 32(1):57-95.

48 Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling*, Springer, 1984, pp. 191-233.

49 Salimi, B. and Bertossi, L. Causality in Databases: The Diagnosis and Repair Connections. Presented at *The 15th International Workshop on Non-Monotonic Reasoning (NMR 2014).* Posted at: arXiv:1404.6857 [cs.DB].

50 Salimi, B. and Bertossi, L. From Causes for Database Queries to Repairs and Model-Based Diagnosis and Back. Extended version of this paper. Posted at: arXiv:1412.4311 [cs.DB].

51 Staworko, S., Chomicki, J. and Marcinkowski, J. Prioritized Repairing and Consistent Query Answering in Relational Databases. *Ann. Math. Artif. Intell.*, 2012, 64(2-3):209-246.

52 Struss, P. Model-based Problem Solving. In *Handbook of Knowledge Representation*, chap. 10. Elsevier, 2008.

53 Tannen, V. Provenance Propagation in Complex Queries. In *Buneman Festschrift*, 2013, Springer LNCS 8000, pp. 483-493.

54 Yakout, M., Elmagarmid, A., Neville, J., Ouzzani, M. and Ilyas, I. Guided Data Repair. *PVLDB*, 2011, 4(5):279-289.

On the Relationship between Consistent Query Answering and Constraint Satisfaction Problems

Carsten Lutz[1] **and Frank Wolter**[2]

1 **Fachbereich Informatik, Universität Bremen, Germany**
clu@uni-bremen.de
2 **Department of Computer Science, University of Liverpool, UK**
wolter@liverpool.ac.uk

Abstract

Recently, Fontaine has pointed out a connection between consistent query answering (CQA) and constraint satisfaction problems (CSP) [22]. We investigate this connection more closely, identifying classes of CQA problems based on denial constraints and GAV constraints that correspond *exactly* to CSPs in the sense that a complexity classification of the CQA problems in each class is *equivalent (up to FO-reductions)* to classifying the complexity of all CSPs. We obtain these classes by admitting only monadic relations and only a single variable in denial constraints/GAVs and restricting queries to hypertree UCQs. We also observe that dropping the requirement of UCQs to be hypertrees corresponds to transitioning from CSP to its logical generalization MMSNP and identify a further relaxation that corresponds to transitioning from MMSNP to GMSNP (also know as MMSNP_2). Moreover, we use the CSP connection to carry over decidability of FO-rewritability and Datalog-rewritability to some of the identified classes of CQA problems.

1998 ACM Subject Classification H.2.4 [Systems]: Relational databases

Keywords and phrases Consistent Query Answering, Constraint Satisfaction, Data Complexity, Dichotomies, Rewritability

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.363

1 Introduction

In modern applications of database systems, it cannot always be guaranteed that the data is consistent with the relevant integrity constraints; for example, inconcistency occurs easily when the data is extracted from the web or integrated from multiple sources. A prominent approach to address this problem is consistent query answering (CQA) as introduced in [3] where one returns the certain answers over all *minimal repairs* of the inconsistent database, see also the surveys [7, 15, 41]. Since the data complexity of CQA can be coNP-complete or higher [2, 14, 16, 38], CQA is in general significantly harder than traditional query answering. This observation has resulted in a lot of research activity aiming to more precisely clarify the computational complexity of CQA, separating in particular the easy cases from the hard ones. Here, 'easy' might mean different things. The ideal result is that a CQA problem $\text{CQA}(C, q)$, defined by a set C of integrity constraints and a query q, is rewritable into a first-order logic (FO) query $\widehat{q}$ and thus answers can be computed by a classical RDBMS and in AC_0 data complexity [24, 39]. If FO-rewritability is not attainable, one might at least hope for Datalog-rewritability or PTime data complexity. Ultimate goals of this research programme would be to classify the exact complexity of *every* CQA problem and, closely related, to *decide* for a given CQA problem whether it is easy in some relevant sense, say whether it admits an FO-rewriting. Completely classifying the border between PTime and coNP

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 363–379

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

prominently involves solving dichotomy questions: for some important classes of integrity constraints and queries, it has been conjectured that CQA is in PTime or coNP-hard for every problem in the class [1, 40, 41]. In fact, with today's methods one cannot hope to completely classify the complexity of a class of CQA problems if that class does *not* have such a dichotomy.

Despite serious efforts, comprehensive results for dichotomies in CQA have so far been elusive. For several restricted cases which all require that the query to be answered is free of self-joins, dichotomies were obtained in [27, 40, 28]. An explanation of why general dichotomy results for CQA are difficult to obtain was recently given by Fontaine [22], who linked CQA with the area of constraint satisfaction problems (CSPs). CSPs constitute a subclass of NP that contains many relevant NP-complete problems such as 3SAT, 3COL, and integer programming over bounded domains, and it is widely believed that this class is computationally more well-behaved than NP itself. In particular, a long-standing conjecture due to Feder and Vardi states that CSPs enjoy a dichotomy between PTime and NP [20]. Massive research efforts in logic, complexity, and algebra have been directed towards proving this conjecture, and although steady progress has been made, the conjecture is still open. In contrast, strong results have been obtained on the FO-definability and Datalog-definability of CSPs, the counterpart of FO- and Datalog-rewritability in CQA: while both problems are undecidable for the entire class NP, they are decidable and NP-complete for CSPs [30, 5, 23].

Fontaine's main result in [22] links the Feder-Vardi conjecture to dichotomies in CQA under GAV constraints by showing that for every CSP problem $\mathrm{CSP}(A)$ defined by some template A, there is a CQA problem $\mathrm{CQA}(C, q)$ with C a set of GAV constraints and q a union of conjunctive queries (UCQ) such that $\mathrm{CQA}(C, q)$ and the complement $\mathrm{coCSP}(A)$ of $\mathrm{CSP}(A)$ are PTime-equivalent, that is, they have the same complexity up to PTime-reductions. Consequently, establishing a dichotomy between PTime and coNP for CQA with GAV constraints and UCQs implies the Feder-Vardi conjecture. The aim of this paper is to study the CSP-CQA connection in more detail, focussing on denial constraints and GAV constraints which have both received significant attention in CQA [17, 16, 8, 2, 37, 38, 22]. In particular, we aim to identify classes of CQA problems that *exactly* correspond to the class of CSPs in the sense that classifying the complexity of problems from both classes is equivalent in a strong sense.

Our first main observation is that CSPs correspond to CQA problems whose (denial or GAV) constraints involve *only monadic relation symbols* and *only a single variable* and in which the query to be answered is a *UCQ in which all CQs take the form of a hypertree* (all queries in this paper are Boolean). More specifically, let a *monadic disjointness constraint (MDiC)* be of the form $\forall x\, \neg(P_1(x) \wedge \cdots \wedge P_n(x))$ and a *monadic GAV constraint (MGAV)* be of the form $\forall x\, (P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow Q(x))$. We use $(\mathrm{MDiC}, \mathrm{tUCQ})$ to denote the class of all CQA problems whose constraints are MDiCs and whose query is a UCQ in which every CQ is a hypertree in the sense that its incidence graph is a tree (without multi-edges), and likewise for $(\mathrm{MGAV}, \mathrm{tUCQ})$. We then show that (i) for every $\mathrm{CSP}(A)$ there is a problem $\mathrm{CQA}(C, q)$ in $(\mathrm{MDiC}, \mathrm{tUCQ})$ such that $\mathrm{coCSP}(A)$ and $\mathrm{CQA}(C, q)$ are FO-equivalent (that is, they have the same complexity up to FO-reductions), (ii) for every problem in $(\mathrm{MDiC}, \mathrm{tUCQ})$ there is one from $(\mathrm{MGAV}, \mathrm{tUCQ})$ that is FO-equivalent, and (iii) for every problem $\mathrm{CQA}(C, q)$ in $(\mathrm{MGAV}, \mathrm{tUCQ})$ there is a $\mathrm{CSP}(A)$ such that $\mathrm{CQA}(C, q)$ and $\mathrm{coCSP}(A)$ are FO-equivalent. This improves upon the main result of [22] in several ways. First, the CQA problems constructed in [22] involve constraints that contain non-monadic relations and the UCQs used there are not hypertrees since they include multi-edges; we thus identify much more restricted CQA classes whose complexity classification is already as hard

as classifying CSPs; second, we also translate *from CQA to CSP* and thus show that a PTime vs. coNP dichotomy for (MDiC, tUCQ) and (MGAV, tUCQ) is *equivalent* to the Feder-Vardi conjecture (instead of only implying it); and third, we replace PTime-equivalence with FO-equivalence which (a) means that not only a PTime vs. (co)NP dichotomy carries over, but *any* complexity classification that involves complexity classes closed under FO-reductions (such as LogSpace), and (b) enables us to transfer results from CSP to the classes of CQA problems that we have identified.

Regarding (b), we show that for (MDiC, tUCQ) and (MGAV, tUCQ), rewritability into FO and into Datalog are decidable, referring to the version of Datalog which allows negation in front of body atoms that use a monadic EDB relation. Our approach yields NExpTime upper bounds for these problems and we demonstrate that the complexity is indeed high (despite the fact that we deal with rather restricted CQA problems) by establishing a PSpace lower bound for FO-rewritability, leaving the exact complexity open. We also transfer Bulatov's result that CSPs whose templates have at most three elements enjoy a dichotomy between PTime and NP [13] to CQA, identifying a (rather restricted) corresponding fragment of (MDiC, tUCQ) that has a dichotomy between PTime and coNP.

We then investigate the effect of dropping the requirement that queries are hypertrees while maintaining all restrictions on integrity constraints, which gives rise to the classes of CQA problems (MDiC, UCQ) and (MGAV, UCQ). We show that this corresponds to transitioning from CSP to its logical generalization MMSNP, which was introduced by Feder and Vardi when studying the descriptive complexity of CSP [20] and has received considerable interest, see e.g. [33, 11]. More precisely, we establish results that exactly parallel (i) to (iii) above, replacing tUCQs with UCQs and CSP with MMSNP. Again, all reductions are FO-reductions. The known results that CSP $\subseteq$ MMSNP and that for every MMSNP problem there is a CSP problem that is PTime-equivalent [20, 29] then also yields that (MDiC, UCQ) has a dichotomy between PTime and coNP if and only if this is the case for (MDiC, tUCQ), and likewise for the corresponding CQA classes based on MGAVs. This does not imply, though, that a *full* complexity classification of these classes is equivalent since the relation between CSP and MMSNP in terms of FO-reductions is open. These results also shed some light on the decidability of FO- and Datalog-rewritability in (MDiC, UCQ) and (MGAV, UCQ), which is equivalent to the decidability of FO-definability of MMSNP problems, an open problem. Finally, we generalize (MDiC, UCQ) by giving up the restriction that integrity constraints are monadic and comprise only a single variable, instead requiring that every atom in the integrity constraint comprises the same variables in the same order. We then show that this corresponds to the transition from MMSNP to GMSNP [10] (also known as MMSNP_2 [32]), which means to replace monadicity as stipulated in MMSNP for certain relations with a guardedness condition.

Some proof details are deferred to the appendix of the long version of this paper available at `http://www.informatik.uni-bremen.de/tdki/research/papers.html`.

2 Preliminaries

A *schema* is a finite collection $\mathbf{S} = (S_1, \dots, S_k)$ of relation symbols with associated non-zero arity. A *fact* over $\mathbf{S}$ is an expression of the form $S(a_1, \dots, a_n)$ where $S \in \mathbf{S}$ is an n-ary relation symbol, and $a_1, \dots, a_n$ are elements of some fixed, countably infinite set const of *constants*. An *instance* I over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* $\mathsf{adom}(I)$ of I is the set of all constants that occur in the facts of I. We will frequently use boldface notation for tuples, such as in $\mathbf{a} = a_1 \cdots a_n$.

A *conjunctive query (CQ)* takes the form $q = \exists \mathbf{x}\, \varphi(\mathbf{x})$ where φ is a conjunction of relational atoms, neither constants nor equality allowed. A *union of conjunctive queries (UCQ)* is a disjunction of CQs. Note that we consider only Boolean queries for simplicity, see the conclusion for some further remarks on this issue.

A *denial constraint (DC)* has the form $\forall \mathbf{x}\, \neg\varphi(\mathbf{x})$, where φ is a conjunction of relational atoms. A *global as view constraint (GAV)* takes the form $\forall \mathbf{x}\, \varphi(\mathbf{x}) \rightarrow S(\mathbf{x})$ where φ is a conjunction of relational atoms. Let I be an instance and C a set of constraints. An instance J is a *minimal repair of I w.r.t. C* if J satisfies all constraints in C and there is no instance J' such that J' satisfies all constraints in C and $I \,\Delta\, J' \subsetneq I \,\Delta\, J$, where '$\Delta$' denotes symmetric difference. We generally omit 'w.r.t. C' when C is clear from the context. For a query q, we write $I \models_C q$ if every minimal repair J of I satisfies $J \models q$.

A *consistent query answering (CQA) problem*, denoted $\mathrm{CQA}(C, q)$, is defined by a set of constraints C and a query q. As input, an $\mathbf{S}$-instance I is given where $\mathbf{S}$ is the set of relation symbols used in C or q. The question is whether $I \models_C q$. We use $(\mathrm{DC}, \mathrm{UCQ})$ to denote the set of problems $\mathrm{CQA}(C, q)$ where C is a set of DCs and q a UCQ, and likewise for $(\mathrm{GAV}, \mathrm{UCQ})$ and other combinations of constraint language and query language.

In this paper, all considered decision problems take instances over some fixed schema as inputs. For two such decision problems P_1, P_2, we write $P_1 \preceq_p P_2$ if P_1 reduces to P_2 by a polynomial time reduction. We write $P_1 \preceq_{\mathsf{FO}} P_2$ if P_1 reduces to P_2 by an *FO-reduction*, defined as in [26]. However, most of our FO-reductions are of the following simple form. For problems P_1 and P_2 over schemas $\mathbf{S}_1$ and $\mathbf{S}_2$, a map T that assigns to each $\mathbf{S}_1$-instance I an $\mathbf{S}_2$-instance $T(I)$ is *FO-definable* if for every k-ary relation symbol R in $\mathbf{S}_2$, there is an FO-formula φ_R over $\mathbf{S}_1$ (equality and constants allowed) with k free variabes such that $R(\mathbf{a}) \in T(I)$ iff $I \models \varphi_R[\mathbf{a}]$, for all $\mathbf{a}$ in $\mathsf{adom}(I)$. Such a map gives rise to an FO-reduction of P_1 to P_2 if for all $\mathbf{S}_1$-instances I, we have $I \in P_1$ iff $T(I) \in P_2$. FO-reductions of this simple form differ from the general case in that (i) no arithmetic operations are admitted in the formulas φ_R and (ii) the domain of the $T(I)$ cannot be larger than the domain of I. With *FO-equivalence* and *PTime-equivalence* of two problems P_1 and P_2, denoted $P_1 \approx_{\mathsf{FO}} P_2$ and $P_1 \approx_p P_2$, we mean that there are reductions between P_1 and P_2 in both directions. For two classes of decision probems $\mathcal{C}_1$ and $\mathcal{C}_2$, we write $\mathcal{C}_1 \preceq_{\mathsf{FO}} \mathcal{C}_2$ if for every problem $p_1 \in \mathcal{C}_1$, there is a problem $p_2 \in \mathcal{C}_2$ such that $p_1 \preceq_{\mathsf{FO}} p_2$ and $p_2 \preceq_{\mathsf{FO}} p_1$. We write $\mathcal{C}_1 \approx_{\mathsf{FO}} \mathcal{C}_2$ and say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are *FO-equivalent* if $\mathcal{C}_1 \preceq_{\mathsf{FO}} \mathcal{C}_2$ and $\mathcal{C}_2 \preceq_{\mathsf{FO}} \mathcal{C}_1$. The definition of $\mathcal{C}_1 \preceq_p \mathcal{C}_2$ and $\mathcal{C}_1 \approx_p \mathcal{C}_2$ is analogous, but based on polynomial time reductions.

Let A be an instance over schema $\mathbf{S}$. The *constraint satisfaction problem* $\mathrm{CSP}(A)$ is to decide, given an instance I over $\mathbf{S}$, whether there is a homomorphism from I to A, which we denote with $I \rightarrow A$. In this context, A is called the *template* of $\mathrm{CSP}(A)$. We will generally and w.l.o.g. assume that the template is a core, that is, every automorphism is an isomorphism. It is often useful to further assume that the template A *admits precoloring*,[1] that is, for each $a \in \mathsf{adom}(A)$, there is a unary relation symbol $P_a \in \mathbf{S}$ such that $P_a(b) \in A$ iff $b = a$ [18]. It is known that for every template A (which is a core), there is a template A' that admits precoloring such that $\mathrm{CSP}(A) \approx_{\mathsf{FO}} \mathrm{CSP}(A')$ [31]. We use $\mathrm{coCSP}(A)$ to denote the complement problem of $\mathrm{CSP}(A)$ and coCSP to denote the set of all problems $\mathrm{coCSP}(A)$ whose template A admits precoloring.

The logic MMSNP was introduced by Feder and Vardi as a descriptive complexity counterpart of CSPs [20]. Since we will mostly be concerned with the *complement* of MMSNP, we refrain from giving the original definition and directly introduce its complement,

[1] This property is also known as 'full idenpotence' and 'pointedness'.

which can conveniently be defined as (negation-free) monadic disjunctive Datalog [10]. A *monadic disjunctive Datalog (MDDLog) rule* ρ has the form $R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n) \to S_1(x_1) \vee \cdots \vee S_m(x_m)$ with $m \geq 0$, $n > 0$, and $S_1, \ldots, S_m$ monadic. Every variable that occurs in the *head* $S_1(x_1) \vee \cdots \vee S_m(x_m)$ of ρ is also required to occur in ρ's *body* $R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n)$. Empty rule heads are denoted $\bot$. An *MDDLog program* Π is a finite set of MDDLog rules with a selected nullary *goal relation* goal that does not occur in rule bodies and only in *goal rules* of the form $R_1(\mathbf{x}_1) \wedge \cdots \wedge R_n(\mathbf{x}_n) \to \mathsf{goal}()$. Relation symbols that occur in the head of at least one rule of Π are *intensional (IDB)*, and all remaining relation symbols in Π are *extensional (EDB)*. Every MDDLog program Π defines a decision problem: given an $\mathbf{S}$-instance I, where $\mathbf{S}$ is the set of EDB relations in Π, decide whether $I \models \Pi$, that is, whether in every extension of I that satisfies all non-goal rules in Π, the body of at least one goal rule applies. We use coMMSNP to denote the class of all these decision problems and MMSNP to denote the class of their complements.

3 Relating CQA and CSP

We identify fragments of (DC, UCQ) and (MGAV, UCQ) that are FO-equivalent to coCSP. They involve restrictions on the admitted constraints as well as on the admitted queries. A denial constaint is called a *monadic disjointness constraint (MDiC)* if all relation symbols that occur in it are monadic and it contains only a single variable; a GAV is called *monadic* or an *MGAV* if it satisfies the same conditions. MDiCs and MGAVs are clearly rather restricted classes of constraints. Note, however, that they are useful for speaking about *type systems*. For example, the MDiC $\forall x\, \neg(\mathsf{person}(x) \wedge \mathsf{process}(x))$ asserts disjointness of the types person and $\mathsf{process}$ while the MGAV $\forall x\, (\mathsf{professor}(x) \to \mathsf{person}(x))$ ensures a subsumption between the types $\mathsf{professor}$ and person.

While we restrict constraints to MDiCs and MGAVs, UCQs are required to contain only CQs that have the shape of a hypertree. The *incidence graph* of a CQ q is the bipartite undirected graph whose nodes are the variables and the atoms in q and whose edges connect a variable x with each atom $R(\mathbf{x})$ such that $x \in \mathbf{x}$. A CQ is a *hypertree conjunctive query (tCQ)* if its incidence graph is a tree (without multi-edges). A *hypertree UCQ (tUCQ)* is a UCQ in which every CQ is a tCQ. Our notion of hypertree CQ is rather restrictive, see e.g. [6, 25] for more general forms of hypertrees; our form of hypertrees, though, is known to be intimately connected to the expressive power of CSPs [34].

Before proceeding, we observe the following connection between MDiCs and MGAVs.

▶ **Lemma 1.** *$(MDiC, \mathcal{Q}) \preceq_{\mathsf{FO}} (MGAV, \mathcal{Q})$ for $\mathcal{Q} \in \{UCQ, tUCQ\}$. Moreover, given a problem $CQA(C, q)$ from $(MDiC, \mathcal{Q})$ one can construct a problem $CQA(C', q')$ in $(MGAV, \mathcal{Q})$ such that $CQA(C, q) \approx_{\mathsf{FO}} CQA(C', q')$ in polynomial time.*

Proof. Let $\mathrm{CQA}(C, q)$ be a problem from (MDiC, $\mathcal{Q}$) over schema $\mathbf{S}$. Define $C' = \{\forall x\, \varphi(x) \to M(x) \mid \forall x\, \neg\varphi(x) \in C\}$ where M is a fresh monadic relation symbol and $q' = q \vee \exists x\, M(x)$. We show that $\mathrm{CQA}(C, q) \preceq_{\mathsf{FO}} \mathrm{CQA}(C', q')$ and vice versa. For the former, we observe that for all $\mathbf{S}$-instances I, we have $I \models_C q$ iff $I \models_{C'} q'$. For the latter, it is easy to show that, for all $\mathbf{S} \cup \{M\}$-instances I, we have $I \models_{C'} q'$ iff $I \models \exists x\, M(x)$ or $I \models_C q$. Clearly, these reductions can be implemented as FO-reductions. ◀

Our aim is to show that $(\mathrm{MDiC}, \mathrm{tUCQ}) \approx_{\mathsf{FO}} (\mathrm{MGAV}, \mathrm{tUCQ}) \approx_{\mathsf{FO}} \mathrm{coCSP}$. By Lemma 1, it suffices to show that $\mathrm{coCSP} \preceq_{\mathsf{FO}} (\mathrm{MDiC}, \mathrm{tUCQ})$ and $(\mathrm{MGAV}, \mathrm{tUCQ}) \preceq_{\mathsf{FO}} \mathrm{coCSP}$. We start with the former, improving upon a reduction by Fontaine [22] which shows that $\mathrm{coCSP} \preceq_p (\mathrm{GAV}, \mathrm{UCQ})$. Consider $\mathrm{CSP}(A)$ over schema $\mathbf{S}$ where A admits precoloring. We

construct a problem $\mathrm{CQA}(C_A, q_A)$ from $(\mathrm{MDiC}, \mathrm{tUCQ})$ over schema $\mathbf{S}'$ which extends $\mathbf{S}$ with unary relation symbols Q_a, $a \in \mathsf{adom}(A)$ (these symbols should be distinguished from the monadic relation symbols P_a in $\mathbf{S}$, $a \in \mathsf{adom}(A)$, which exist since A admits precoloring). C_A contains one monadic disjointness constraint:

$$\forall x \neg \bigwedge_{a \in \mathsf{adom}(A)} Q_a(x).$$

For each $a \in \mathsf{adom}(A)$, we use $\mathsf{con}_a(x)$ to denote the conjunction $\bigwedge_{e \in \mathsf{adom}(A) \setminus \{a\}} Q_e(x)$. The tUCQ q_A contains the following tCQ for each $R \in \mathbf{S}$ of arity n and each $\mathbf{a} = a_1 \cdots a_n \in \mathsf{adom}(A)^n$ such that $R(\mathbf{a}) \notin A$:

$$\exists x_1 \cdots \exists x_n \, (\mathsf{con}_{a_1}(x_1) \wedge \cdots \wedge \mathsf{con}_{a_n}(x_n) \wedge R(x_1, \ldots, x_n)).$$

To understand the construction of $\mathrm{CQA}(C_A, q_A)$, consider the reduction from $\mathrm{CSP}(A)$ to the complement of $\mathrm{CQA}(C_A, q_A)$. Given an $\mathbf{S}$-instance I that is an input to $\mathrm{CSP}(A)$, we construct an $\mathbf{S}'$-instance $T^{\uparrow}(I)$ by adding $Q_a(b)$ for all $b \in \mathsf{adom}(I)$ and all $a \in \mathsf{adom}(A)$. Then each minimal repair J of $T^{\uparrow}(I)$ must satisfy, for each $b \in \mathsf{adom}(I)$, $J \models \mathsf{con}_a[b]$ for a unique $a \in \mathsf{adom}(A)$. In this way, J represents a function h that assigns to each $b \in \mathsf{adom}(I)$ the unique $a \in \mathsf{adom}(A)$ such that $Q_a(b) \notin J$. If $J \not\models q_A$, then h must clearly be a homomorphism. Indeed, we show in the appendix that $I \to A$ iff $T^{\uparrow}(I) \not\models_{C_A} q_A$.

▶ **Lemma 2.** *$coCSP(A) \preceq_{\mathsf{FO}} CQA(C_A, q_A)$ and $CQA(C_A, q_A) \preceq_{\mathsf{FO}} coCSP(A)$.*

Proof. The first reduction was already described above. Clearly, $T^{\uparrow}$ is FO-definable and thus the reduction is an FO-reduction. For the converse reduction, let I be an $\mathbf{S}'$-instance. Denote by X the set of $b \in \mathsf{adom}(I)$ such that there are at least two distinct $a_1, a_2 \in \mathsf{adom}(A)$ with neither $Q_{a_1}(b) \in I$ nor $Q_{a_2}(b) \in I$. Then $T^{\downarrow}(I)$ is obtained from I by dropping all facts that involve a constant from X or a relation symbol in $\mathbf{S}' \setminus \mathbf{S}$, and adding all facts $P_a(b)$ such that $Q_e(b) \in I$ iff $e \neq a$ for all $e \in \mathsf{adom}(A)$. Note that it is crucial here that A admits precoloring as we use the relation symbols P_a. Clearly $T^{\downarrow}(I)$ is FO-definable. We show in the appendix that $I \not\models_{C_A} q_A$ iff $T^{\downarrow}(I) \to A$. ◀

It is easy to see that, given A, we can construct $\mathrm{CQA}(C_A, q_A)$ in polynomial time. We have thus established the following.

▶ **Theorem 3.** *For every CSP template A that admits precoloring, there is a problem $CQA(C, q)$ from (MDiC,tUCQ) that satisfies $CQA(C, q) \approx_{\mathsf{FO}} coCSP(A)$ and can be constructed in polynomial time.*

We now show that $(\mathrm{MGAV}, \mathrm{tUCQ}) \preceq_{\mathsf{FO}} \mathrm{coCSP}$. Let $\mathrm{CQA}(C, q_0)$ be over schema $\mathbf{S}$ with C a set of MGAVs and q_0 a hypertree UCQ. We use $\mathbf{S}^{(1)}$ to denote the restriction of $\mathbf{S}$ to monadic relation symbols and $\mathbf{S}^{(>1)}$ for the restriction of $\mathbf{S}$ to non-monadic symbols. In the following, we define a CSP template A_{C,q_0} over schema $\mathbf{S}' = \mathbf{S}^{(>1)} \cup \{P_\Gamma \mid \Gamma \subseteq \mathbf{S}^{(1)}\}$. Note that there is an obvious natural translation of $\mathbf{S}$-instances into corresponding $\mathbf{S}'$-instances and vice versa; for example, an $\mathbf{S}'$-instance I is translated to an $\mathbf{S}$-instance J by replacing every fact $P_\Gamma(a)$ with $P(a)$ for all $P \in \Gamma$. The change of schema is used to deal with the complication that the 'yes'-instances of each CSP are closed under homomorphic pre-images while the 'no'-instances of CQA problems from $(\mathrm{MGAV}, \mathrm{tUCQ})$ are not (unless the schema is modified in the described way). For any set $\Gamma \subseteq \mathbf{S}^{(1)}$, we use $\mathsf{rep}(\Gamma)$ to denote the set of all sets $\Gamma' \subseteq \mathbf{S}^{(1)}$ such that $\{P(a) \mid P \in \Gamma'\}$ is a minimal repair of the instance $\{P(a) \mid P \in \Gamma\}$.

Let Q be the set of all connected subqueries of CQs in q_0 (which are again hypertrees) and of all queries of the form $P(x)$, $P \in \mathbf{S}^{(1)}$. A *place* is a pair (q, x) with $q \in Q$ and $x \in \mathsf{var}(q)$. A *type* t is a set of places. The type $\mathsf{tp}_I(a)$ *realized by constant* a *in instance* I is the set of all places (q, x) such that there is a homomorphism h from q to I that takes x to a. We say that a type t is *realizable* if there is an instance I such that I satisfies all constraints in C and t is realized by some constant in I; we say that t *avoids* q_0 if $(q, x) \notin t$ for any disjunct q of q_0 and $x \in \mathsf{var}(q)$. For $R \in \mathbf{S}$ of arity n, we say that a tuple $(t_1, \dots, t_n)$ of types is *R-coherent* if for any instance I and tuples of constants $(a_1, \dots, a_n)$ such that $\mathsf{tp}_I(a_i) = t_i$ for $1 \leq i \leq n$, after adding to I the fact $R(a_1, \dots, a_n)$, we still have $\mathsf{tp}_I(a_i) = t_i$ for $1 \leq i \leq n$. Now, the template A_{C,q_0} is defined as follows:

- the constants in A_{C,q_0} are the pairs $\langle t, \Gamma \rangle$ with t a realizable type that avoids q_0 and $\Gamma \subseteq \mathbf{S}^{(1)}$ such that $t|_{\mathbf{S}^{(1)}} \in \mathsf{rep}(\Gamma)$ where $t|_{\mathbf{S}^{(1)}}$ is the restriction of t to schema $\mathbf{S}^{(1)}$;
- A_{C,q_0} contains all facts of the form $P_\Gamma(\langle t, \Gamma \rangle)$;
- A_{C,q_0} contains all facts $R(\langle t_1, \Gamma_1 \rangle, \dots, \langle t_n, \Gamma_n \rangle)$, R of arity n, if $(t_1, \dots, t_n)$ is R-coherent.

To understand the construction, consider the reduction from the complement of $\mathrm{CQA}(C, q_0)$ to $\mathrm{CSP}(A_{C,q_0})$ and let I be an $\mathbf{S}$-instance that is an input to the former. We replace I with the corresponding $\mathbf{S}'$-instance $T^{\uparrow}(I)$ obtained from I by dropping all facts that involve a monadic relation and adding $P_{\Gamma_a}(a)$ for every element $a \in I$, where $\Gamma_a = \{P \mid P(a) \in I\}$. Then, a homomorphism h from $T^{\uparrow}(I)$ to A_{C,q_0} defines the repair of I that is obtained by repairing the monadic relations at each $a \in \mathsf{adom}(I)$ as suggested by $h(a) = \langle t, \Gamma \rangle$, namely to remove all $P(a)$ with $(P(x), x) \notin t$. Indeed, we show in the appendix that $I \not\models_C q_0$ iff $T^{\uparrow}(I) \to A_{C,q_0}$. It is again easy to see that $T^{\uparrow}(I)$ is FO-definable. For later use, we note explicitly that an FO-formula φ_{P_Γ} for defining P_Γ in $T^{\uparrow}(I)$ is given by

$$\varphi_{P_\Gamma}(x) = \bigwedge_{P \in \Gamma} P(x) \wedge \bigwedge_{P \in \mathbf{S}^{(1)} \setminus \Gamma} \neg P(x).$$

It is also interesting to note that the $\mathbf{S}$-instance I_A obtained from the template A_{C,q_0} by dropping all facts $P_\Gamma(\langle t, \Gamma \rangle)$ and adding $P(\langle t, \Gamma \rangle)$ for all $(P(x), x) \in t$ is a *universal minimal repair* in the following sense: (i) it is a minimal repair of the $\mathbf{S}$-instance that corresponds to A_{C,q_0} and (ii) any minimal repair of any $\mathbf{S}$-instance homomorphically maps to I_A.

▶ **Lemma 4.** *$CQA(C, q_0) \preceq_{\mathsf{FO}} coCSP(A_{C,q_0})$ and $coCSP(A_{C,q_0}) \preceq_{\mathsf{FO}} CQA(C, q_0)$.*

Proof. The first reduction was described above. Correctness is proved in full detail in the appendix. For the second reduction, let an $\mathbf{S}'$-instance I be given. If there exists $a \in \mathsf{adom}(I)$ with $P_\Gamma(a), P_\Delta(a) \in I$ for some $\Gamma \neq \Delta$, then there is no homomorphism from I to A_{C,q_0} and "false" is returned. Otherwise define an $\mathbf{S}$-instance $T^{\downarrow}(I)$ by dropping all facts of the form $P_\Gamma(a)$ and adding $P(a)$ whenever $P_\Gamma(a) \in I$ with $P \in \Gamma$. We show in the appendix that $T^{\downarrow}(I) \to A_{C,q_0}$ iff $I \not\models_C q_0$. Clearly $T^{\downarrow}(I)$ is FO-definable. ◀

▶ **Theorem 5.** *For every problem $CQA(C, q)$ from (MGAV,tUCQ), there is a CSP template A that satisfies $coCSP(A) \approx_{\mathsf{FO}} CQA(C, q)$ and can be constructed in single exponential time.*

Summarizing Theorems 3 and 5, we thus obtain the following FO-equivalences.

▶ **Corollary 6.** *$(MDiC, tUCQ) \approx_{\mathsf{FO}} (MGAV, tUCQ) \approx_{\mathsf{FO}} coCSP$.*

It follows that there is a dichotomy between PTime and coNP for (MDiC, tUCQ) and (MGAV, tUCQ) if and only if the Feder-Vardi conjecture is true, and more generally that classifying the complexity of these classes of CQA problems is equivalent to classifying the complexity of CSPs (up to FO reductions).

4 FO- and Datalog-Rewritability

We exploit the reductions from the previous section and recent results concerning the FO- and Datalog-definability of CSPs to show that FO-rewritability and Datalog-rewritability are decidable in (MDiC, tUCQ) and (MGAV, tUCQ). A problem CQA(C, q) over schema $\mathbf{S}$ is *FO-rewritable* if there is a Boolean FO-query $\widehat{q}_{C,q}$ such that for all $\mathbf{S}$-instances I, we have $I \models_C q$ iff $I \models \widehat{q}_{C,q}$. Thus, FO-rewritability ensures that CQA(C, q) can be implemented using a conventional RDBMS. Datalog-rewritability is defined accordingly, where we refer to the version of Datalog that admits *negated monadic EDB atoms* in rule bodies. Without such atoms, Datalog-rewritability would be an extremely elusive property, as illustrated by the trivial problem CQA(C, q) where $C = \{\forall x\, \neg(A_1(x) \wedge A_2(x))\}$ and $q = \exists x\, A_1(x)$, which can be rewritten into $A_1(x) \wedge \neg A_2(x) \to \mathsf{goal}()$, but not into any Datalog program without negated monadic EDB atoms. Note that in constrast to consistent query answering problems, a class coCSP(A) is Datalog-definable with negated EDB atoms in rule bodies if and only if it is definable by a negation-free Datalog program [21]. We rely on the following results from [30, 5, 23].

▶ **Theorem 7.** *Given a CSP template A, it is* NP*-complete to decide whether coCSP(A) is FO-definable. The same is true for Datalog-definability.*

Since the reductions between CQA(C, q) and coCSP$(A_{C,q})$ used in the proof of Theorem 5 are FO-reductions, it follows immediately that CQA(C, q) is FO-rewritable if and only if coCSP$(A_{C,q})$ is FO-definable [26]. Given that $A_{C,q}$ can be constructed in single exponential time, we obtain a NExpTime upper bound for deciding FO-rewritability of CQA problems in (MDiC, tUCQ) and (MGAV, tUCQ).

▶ **Theorem 8.** *Given (C, q) such that CQA(C, q) is from (MDiC, tUCQ) or (MGAV, tUCQ), it is decidable in* NExpTime *whether CQA(C, q) is FO-rewritable. Both problems are PSpace-hard.*

Proof (Sketch). The PSpace lower bounds are proved in the appendix by a non-trivial reduction of the word problem of polynomially space-bounded Turing machines. Similar reductions have been used to establish PSpace-hardness of boundedness in linear monadic datalog [19] and of certain FO-rewritability problems in ontology-based data access [9]. The general idea is to start with a DTM M that solves a PSpace-complete problem and to first modify it so that M terminates when started in *any* configuration (which must not even be reachable from an initial configuration). Then, one crafts a problem CQA(C, q) such that if M accepts an input x, then any FO-rewriting of CQA(C, q) would have to query for the existence of unboundedly long paths of facts that represent an accepting computation of M on x, repeated over and over again. Clearly, this contradicts the locality of FO-queries. For technical reasons, we actually cannot ensure that the first computation on the mentioned paths starts with the initial configuration for x. However, M terminates from any configuration, and the second computation on the paths (as well as all subsequent ones) are guaranteed to start with the initial configuration for x. Details are given in the appendix. We note that C consists only of a single constraint, which is of the form $\forall x\, \neg(B(x) \wedge B'(x))$. ◀

The exact complexity remains open, but we speculate that the problems in Theorem 8 are at least ExpTime-hard. To obtain complexity bounds for Datalog-rewritability, we inspect the FO-formulas required to define $T^{\uparrow}(I)$ and $T^{\downarrow}(I)$ in the proof of Lemma 4.

▶ **Lemma 9.** *Let CQA(C, q_0) and A_{C,q_0} be as in Lemma 4. Then CQA(C, q_0) is Datalog-rewritable iff coCSP(A_{C,q_0}) is Datalog-definable.*

Proof. Let Π be a Datalog program that defines coCSP(A_{C,q_0}). Replace in Π all body atoms $P_\Gamma(x)$ with $\bigwedge_{P\in\Gamma} P(x) \wedge \bigwedge_{P\in\mathbf{S}^{(1)}\setminus\Gamma} \neg P(x)$ and denote the resulting program by Π'. It can be verified using the proof of Lemma 4 that $I \models_C q_0$ iff $I \models \Pi'$ for all $\mathbf{S}$-instances I. Thus Π' is a Datalog-rewriting of CQA(C, q_0).

Conversely, assume that Π is a Datalog-rewriting of CQA(C, q_0). Replace every rule ρ in Π by a set of rules as follows. For every variable x in ρ, replace the set S of conjuncts in the body of ρ that are of the form $P(x)$ and $\neg P(x)$, $P \in \mathbf{S}^{(1)}$, by any atom $P_\Gamma(x)$ such that $S \subseteq \{P(x) \mid P \in \Gamma\} \cup \{\neg P(x) \mid P \in \mathbf{S}^{(1)} \setminus \Gamma\}$. Moreover, add $P_\Gamma(x) \wedge P_\Lambda(x) \to \mathsf{goal}()$ as a new rule for all $\Gamma, \Lambda \subseteq \mathbf{S}^{(1)}$ with $\Gamma \neq \Lambda$. Denote the resulting program by Π'. It can be verified using the proof of Lemma 4 that Π' defines coCSP(A_{C,q_0}). ◀

The following is a consequence of Lemma 4, Lemma 9, and Theorem 7.

▶ **Theorem 10.** *Given (C, q) such that CQA(C, q) is from $(\mathit{MDiC}, \mathit{tUCQ})$ or $(\mathit{MGAV}, \mathit{tUCQ})$, it is decidable in* NExpTime *whether CQA(C, q) is Datalog-rewritable.*

We do not have any non-trivial lower bound. Finally, we observe that, thanks to allowing negation in Datalog-rewritings as described above, FO-rewritability implies Datalog-rewritability.

▶ **Theorem 11.** *In $(\mathit{MDiC}, \mathit{tUCQ})$ and $(\mathit{MGAV}, \mathit{tUCQ})$, FO-rewritability implies (non-recursive) Datalog-rewritability.*

Proof. It was observed by Atserias (and follows from Rossman's homomorphism preservation theorem) that for any CSP template A, coCSP(A) being FO-definable implies that coCSP(A) is UCQ-definable [4, 35]. Thus FO-rewritability of CQA(C, q) implies FO-definability of coCSP($A_{C,q}$). The latter implies UCQ-definability of coCSP($A_{C,q}$) which implies (non-recursive) Datalog-definability of coCSP($A_{C,q}$). The latter implies (non-recursive) Datalog-rewritability of CQA(C, q). ◀

5 A Dichotomy Result

For restricted classes of CSPs, a dichotomy between PTime and NP has been established. The most notable results include Schaefer's famous dichotomy theorem for templates with at most two elements [36] and its generalization to three element templates obtained much later by Bulatov [13]. It is natural to ask whether these dichotomies transfer to restricted classes of CQA problems. In this section, we identify a subclass of (MDiC, tUCQ) for which this is the case, thus obtaining a dichotomy between PTime and coNP.

We consider the class of problems CQA(C, q_0) such that C consists of a single MDiC of the form $\forall x\, \neg(A_1(x) \wedge A_2(x))$ and q_0 is a tUCQ such that, in every tCQ in q_0, there is at most one atom with a relation symbol distinct from A_1, A_2. Let us call a problem of this form a *restricted binary CQA problem* where 'binary' refers to the number of atoms allowed in the MDiC. Note that the resulting class r2CQA of CQA problems is not trivial since a straightforward analysis of the proof of Theorem 3 shows that 2coCSP $\preceq_{\mathsf{FO}}$ r2CQA, where $i$coCSP is the class of complements of all problems that can be defined by a CSP template with at most i elements. Consequently, establishing a PTime / coNP dichotomy for r2CQA implies Schaefer's theorem; we actually find it remarkable that a class of CQA problems as simple as 2rCQA turns out to be that complex. In the following, we show that r2CQA $\preceq_{\mathsf{FO}}$ 3coCSP and thus obtain a dichotomy between PTime and coNP for r2CQA by Bulatov's result.

Let $\mathrm{CQA}(C, q_0)$ be a restricted binary CQA problem over schema $\mathbf{S}$. We define a CSP template A_{C,q_0} over schema $\mathbf{S}' = (\mathbf{S} \setminus \{A_1, A_2\}) \cup \{P_\Gamma \mid \Gamma \subseteq \{A_1, A_2\}\}$ as follows:

1. the constants in A_{C,q_0} are $0, 1, 2$;
2. A_{C,q_0} contains the facts $P_\emptyset(0)$, $P_{\{A_1\}}(1)$, $P_{\{A_2\}}(2)$, $P_{\{A_1,A_2\}}(1)$, and $P_{\{A_1,A_2\}}(2)$;
3. A_{C,q_0} contains the fact $R(i_1, \ldots, i_k)$, $R \in \mathbf{S} \setminus \{A_1, A_2\}$ and $i_j \in \{1, 2\}$, when q_0 does not evaluate to true on the $\mathbf{S}$-instance $\{R(i_1, \ldots, i_k)\} \cup \{A_{i_j}(i_j) \mid 1 \leq j \leq k\}$.

The general idea is the same as in the proof of Theorem 5, that is, a homomorphism h from an $\mathbf{S}'$-instance J to A_{C,q_0} defines a repair of the corresponding $\mathbf{S}$-instance I. In fact, for any situation $A_1(a), A_2(a) \in I$ we must have $h(a) \in \{1, 2\}$ and in the repair of I described by h we then keep $A_{h(a)}$ and remove $A_{3-h(a)}$. The element 0 in A_{C,q_0} is needed as a homomorphism target for constants $a \in \mathsf{adom}(I)$ such that neither $A_1(a)$ nor $A_2(a)$ are in I.

▶ **Lemma 12.** *$CQA(C, q_0) \preceq_{\mathsf{FO}} coCSP(A_{C,q_0})$ and $coCSP(A_{C,q_0}) \preceq_{\mathsf{FO}} CQA(C, q_0)$.*

The FO-reductions used for proving Lemma 12 are identical to those in the proof of Lemma 4, but of course the correctness proofs differ. Details are given in the appendix.

▶ **Theorem 13.** *Every binary restricted CQA problem is in PTime or* coNP*-complete.*

6 Relating CQA and MMSNP

The classes of CQA problems identified so far all require queries to be hypertree UCQs. We now show that the transition from hypertree UCQs to unrestricted UCQs corresponds to the transition from CSP to MMSNP: while classifying the complexity of $(\mathrm{MDiC}, \mathrm{tUCQ})$ and $(\mathrm{MGAV}, \mathrm{tUCQ})$ is equivalent to classifying the complexity of coCSP (up to FO-reductions), classifying $(\mathrm{MDiC}, \mathrm{UCQ})$ and $(\mathrm{MGAV}, \mathrm{UCQ})$ is equivalent in the same sense to classifying coMMSNP. Since it is known that MMSNP $\approx_p$ CSP, the results in this section also imply that there is a dichotomy between PTime and coNP for $(\mathrm{MDiC}, \mathrm{tUCQ})$ if and only if there is such a dichotomy for $(\mathrm{MDiC}, \mathrm{UCQ})$ if and only if the Feder-Vardi conjecture holds (and likewise for the corresponding CQA languages based on MGAVs). They also yield some insight into the problem of deciding FO-rewritability in $(\mathrm{MDiC}, \mathrm{UCQ})$ and $(\mathrm{MGAV}, \mathrm{UCQ})$: these problems are decidable if and only if FO-definability in MMSNP is decidable, which is an open problem.

Recall that coMMSNP is the class of problems definable by an MDDLog program. Thus, let Π be an MDDLog program over schema $\mathbf{S}$ and assume that $\mathbf{Q}$ is the set of IDB relations in Π. A $\mathbf{Q}$-*type* is a subset $t \subseteq \mathbf{Q}$. We say that Π *admits precoloring* if the EDB schema $\mathbf{S}$ includes a monadic relation symbol S_t for each $\mathbf{Q}$-type t and Π includes rules (i) $S_t(x) \to Q(x)$ for all $\mathbf{Q}$-types t and all $Q \in t$ and (ii) $S_t(x) \wedge Q(x) \to \bot$ for all $\mathbf{Q}$-types t and all $Q \in \mathbf{Q} \setminus t$; the S_t relations are not allowed to be used in any other rule. We use $\mathrm{coMMSNP}^{\mathsf{pre}}$ to denote the class of all problems defined by a MDDLog program that admits precoloring and $\mathrm{MMSNP}^{\mathsf{pre}}$ to denote the class of their complements. Recall that we can w.l.o.g. assume CSPs to admit precoloring. The following result says that the same is true for (co)MMSNP.

▶ **Theorem 14** (Bodirsky and Madelaine [12])**.** *$MMSNP \preceq_{\mathsf{FO}} MMSNP^{\mathsf{pre}}$.*

We start with establishing a counterpart of Theorem 3. Let Π be an MDDLog program over schema $\mathbf{S}$ that admits precoloring with IDB relations $\mathbf{Q}$, as above. We use tp to denote the set of all $\mathbf{Q}$-types. Construct a problem $\mathrm{CQA}(C_\Pi, q_\Pi)$ in $(\mathrm{MDiC}, \mathrm{UCQ})$ over schema $\mathbf{S}'$ which extends $\mathbf{S}$ with unary relation symbols Q_t, $t \in \mathsf{tp}$ (to be distinguished from the EDB

relations S_t required because Π admits precoloring). C_Π contains one monadic disjointness constraint:

$$\forall x \neg \bigwedge_{t \in \mathsf{tp}} Q_t(x).$$

For each $\mathbf{Q}$-type t, we use $\mathsf{con}_t(x)$ to denote the conjunction $\bigwedge_{t' \in \mathsf{tp} \setminus \{t\}} Q_{t'}(x)$. We now construct the UCQ q_Π, which contains the following two kinds of CQs.

1. Consider each non-goal rule

$$\rho = \bigwedge_{1 \leq i \leq n} R_i(\mathbf{x}_i) \wedge \bigwedge_{1 \leq i \leq m} S_i(y_i) \to \bigvee_{1 \leq i \leq \ell} S'_i(z_i)$$

where all R_i are from $\mathbf{S}$ and all S_i and S'_i are from $\mathbf{Q}$. Let $x_1, \ldots, x_k$ be the variables in ρ. Then include in q_Π the following CQ, for all sequences of $\mathbf{Q}$-types $t_1, \ldots, t_k$ such that (i) if $S(x_i)$ occurs the body of ρ with $S \in \mathbf{Q}$, then $S \in t_i$ and (ii) if $S(x_i)$ occurs in the head of ρ, then $S \notin t_i$:

$$\bigwedge_{1 \leq i \leq n} R_i(\mathbf{x}_i) \wedge \bigwedge_{1 \leq i \leq k} \mathsf{con}_{t_i}(x_i) \wedge \bigwedge_{1 \leq i \leq \ell} \mathsf{con}_{t'_i}(z_i);$$

2. Consider each goal rule

$$\rho = \bigwedge_{1 \leq i \leq n} R_i(\mathbf{x}_i) \wedge \bigwedge_{1 \leq i \leq m} S_i(y_i) \to \mathsf{goal}()$$

where all R_i are from $\mathbf{S}$ and all S_i are from $\mathbf{Q}$. Let $x_1, \ldots, x_k$ be the variables in ρ. Then include in q_Π the following CQ, for all sequences of $\mathbf{Q}$-types $t_1, \ldots, t_k$ such that if $S(x_i)$ occurs in the body of ρ with $S \in \mathbf{Q}$, then $S \in t_i$:

$$\bigwedge_{1 \leq i \leq n} R_i(\mathbf{x}_i) \wedge \bigwedge_{1 \leq i \leq k} \mathsf{con}_{t_i}(x_i).$$

The above construction parallels the one used in the proof of Theorem 3, with $\mathbf{Q}$-types playing the role of elements of the CSP template. In the reduction from Π to $\mathrm{CQA}(C_\Pi, q_\Pi)$, we are given an $\mathbf{S}$-instance I and construct an $\mathbf{S}'$-instance $T^\uparrow(I)$ by adding $Q_t(a)$ for all $a \in \mathsf{adom}(I)$ and all $\mathbf{Q}$-types t. Then each minimal repair J of $T^\uparrow(I)$ must satisfy, for each $a \in \mathsf{adom}(I)$, $J \models \mathsf{con}_t[a]$ for a unique $\mathbf{Q}$-type t and thus assigns to each $a \in \mathsf{adom}(I)$ a $\mathbf{Q}$-type t_a. In this way, it describes a unique $\mathbf{S} \cup \mathbf{Q}$-instance I' that is obtained from I by adding $P(a)$ whenever $P \in t_a$. If $J \not\models q_\Pi$, then I' satisfies all non-goal rules of Π, but no goal rule applies. Indeed, we show in the appendix that $I \models \Pi$ iff $T^\uparrow(I) \models_{C_\Pi} q_\Pi$. We also give an FO-reduction from $\mathrm{CQA}(C_\Pi, q_\Pi)$ to Π, which is again similar to what is done in the proof of Theorem 3.

▶ **Theorem 15.** *For every MDDLog program Π that admits precoloring, there is a problem $CQA(C, q)$ from (MDiC,UCQ) that satisfies $CQA(C, q) \approx_{\mathsf{FO}} \Pi$ and can be constructed in polynomial time.*

We now show that $(\mathrm{MGAV}, \mathrm{UCQ}) \preceq_{\mathsf{FO}} \mathrm{coMMSNP}$. Let $\mathrm{CQA}(C, q)$ be from (MGAV,UCQ) with schema $\mathbf{S}$. We define an MDDLog program $\Pi_{C,q}$ over EDB schema $\mathbf{S}' = \mathbf{S}^{(>1)} \cup \{P_\Gamma \mid \Gamma \subseteq \mathbf{S}^{(1)}\}$ where the P_Γ are fresh monadic relation symbols. Recall from Section 3 that for each $\Gamma \subseteq \mathbf{S}^{(1)}$, $\mathsf{rep}(\Gamma)$ denotes the corresponding set of repairs. The IDB relations of $\Pi_{C,q}$

are $\mathbf{Q} = \mathbf{S}^{(1)} \cup \{Q_\Gamma \mid \Gamma \subseteq \mathbf{S}^{(1)}\}$ where the Q_Γ are monadic. The rules of $\Pi_{C,q}$ are as follows:

$$\begin{array}{rcll} P_\Gamma(x) & \to & \bigvee\limits_{\Lambda \in \mathsf{rep}(\Gamma)} Q_\Lambda(x) & \text{for each } \Gamma \subseteq \mathbf{S}^{(1)} \\ Q_\Gamma(x) & \to & P(x) & \text{for each } \Gamma \subseteq \mathbf{S}^{(1)},\ P \in \Gamma \\ P_\Gamma(x) \wedge P_\Lambda(x) & \to & \mathsf{goal}() & \text{for all distinct } \Gamma, \Lambda \subseteq \mathbf{S}^{(1)} \\ q' & \to & \mathsf{goal}() & \text{for each disjunct } q' \text{ of } q. \end{array}$$

▶ **Lemma 16.** *$CQA(C,q) \preceq_{\mathsf{FO}} \Pi_{C,q}$ and $\Pi_{C,q} \preceq_{\mathsf{FO}} CQA(C,q)$.*

The reductions used for establishing this lemma are *exactly* the FO-reductions from the proof of Lemma 4. In the appendix, we prove that these reductions are correct also in this case.

▶ **Theorem 17.** *For every CQA problem in (MGAV,UCQ), there is an FO-equivalent MDDLog program Π that can be constructed in single exponential time.*

We thus obtain the following equivalences.

▶ **Corollary 18.**
1. *$(MDiC, UCQ) \approx_{\mathsf{FO}} (MGAV, UCQ) \approx_{\mathsf{FO}} coMMSNP$;*
2. *$(MDiC, UCQ) \approx_p (MDiC, tUCQ)$ and $(MGAV, UCQ) \approx_p (MGAV, tUCQ)$.*

Since the FO-reductions in the proof of Lemma 16 are identical to those in the proof of Lemma 4, all arguments used in Section 4 for relating FO- and Datalog-rewritability of CQA problems with hypertree UCQs to the FO- and Datalog-definability of CSPs can also be used to relate, in exactly the same way, CQA problems with unrestricted UCQs to MMSNP. Consequently, we obtain that FO-rewritability in (MDiC, UCQ) and (MGAV, UCQ) is decidable iff FO-definability in MMSNP is decidable and the former is at most exponentially harder than the latter.

7 Relating CQA and GMSNP

Monadic disjointness constraints are surely a rather restricted class of denial constraints. In this section, we analyze a slightly more powerful class obtained by dropping the monadicity of MDiCs while still retaining some 'uniformity' accross the atoms in the constraint. More precisely, a *disjointness constraint (DiC)* has the form $\forall \mathbf{x}\, \neg(R_1(\mathbf{x}) \wedge \cdots \wedge R_n(\mathbf{x}))$ where all relations R_i have the same arity and all atoms $R_i(\mathbf{x})$ use the same variables from $\mathbf{x}$ in the same order (multiple occurrences are allowed). We show that the connection between (MDiC, UCQ) and coMMSNP established in Section 6 can be lifted to (DiC, UCQ) and a generalization of coMMSNP called coGMSNP that corresponds to replacing MDDLog with frontier-guarded disjunctive Datalog [10]. It is straightforward to define a corresponding generalization of GAV constraints and to establish analogous results for them, but for simplicity we refrain from doing so.

Recall that we defined coMMSNP in terms of MDDLog. A *frontier-guarded disjunctive Datalog (GDDLog) rule* takes the form $R_1(\mathbf{y}_1) \wedge \cdots \wedge R_k(\mathbf{y}_k) \to S_1(\mathbf{z}_1) \vee \cdots \vee S_m(\mathbf{z}_m)$ where the IDB predicates need not be monadic and for each head atom $S_i(\mathbf{z}_i)$, there must be a body atom $R_j(\mathbf{y}_j)$ with $\mathbf{z}_i \subseteq \mathbf{y}_j$. A GDDLog program is then defined in the obvious way (with nullary goal predicate) and we use coGMSNP to denote the class of decision problems that are defined by a GDDLog program. GMSNP, which is also known as MMSNP_2 [32],

is considered an interesting candidate for a generalization of MMSNP that is still PTime-equivalent to CSP. It is, however, an open problem whether this is really the case. We nevertheless believe that relating (DiC, UCQ) and coGMSNP provides interesting additional insight into the computational complexity of CQA. Note that coGMSNP is also closely related to ontology-based data access with the guarded fragment of FO [10].

What we actually show in this section is coGMSNP$^{\mathsf{pre}} \preceq_{\mathsf{FO}}$ (DiC, UCQ) $\preceq_{\mathsf{FO}}$ coGMSNP where GMSNP$^{\mathsf{pre}}$ is a version of GMSNP that admits precoloring defined as follows in analogy with MMSNP$^{\mathsf{pre}}$. Let Π be a GDDLog program over schema $\mathbf{S}$, assume that $\mathbf{Q}$ is the set of IDB relations in Π, and let m be the maximal arity of relations in $\mathbf{Q}$. Fix variables $x_1, \dots, x_m$. For $i \leq m$, an *i-type* is a set t of relational atoms using relation symbols from $\mathbf{Q}$ and variables from $\mathbf{x}_i := x_1 \cdots x_i$. We say that Π *admits precoloring* if the EDB schema $\mathbf{S}$ includes a relation symbol S_t of arity i for each i-type t, $i \leq m$, and Π includes rules (i) $S_t(\mathbf{x}_i) \to R(x_{i_1}, \dots, x_{i_\ell})$ for all i-types t and all $R(x_{i_1}, \dots, x_{i_\ell}) \in t$ and (ii) $S_t(\mathbf{x}_i) \wedge R(x_{i_1}, \dots, x_{i_\ell}) \to \bot$ for all i-types t and all $R(x_{i_1}, \dots, x_{i_\ell}) \notin t$ with $R \in \mathbf{Q}$ and $x_{i_1}, \dots, x_{i_\ell} \in \mathbf{x}_i$; the S_t relations are not allowed to be used in any other rule. We leave it open whether coGMSNP $\preceq_p$ coGMSNP$^{\mathsf{pre}}$, but consider it likely that this is the case given the corresponding result for MMSNP mentioned in Section 6.

We start with proving coGMSNP$^{\mathsf{pre}} \preceq_{\mathsf{FO}}$ (DiC, UCQ), first observing that it suffices to consider GDDLog programs of a certain form, which we introduce next. Let $\mathbf{Q}$ be a schema that consists of relations which all have the same arity m. A *$\mathbf{Q}$-type* is a set of relational atoms $R(\mathbf{x})$ with $R \in \mathbf{Q}$ and where $\mathbf{x}$ is a permutation of $x_1 \cdots x_m$. We say that a GDDlog program Π with IDB relations $\mathbf{Q}$ is *normalized* if it satisfies the following conditions:

1. all EDB and IDB relations (except the goal relation) have the same arity m, each variable may occur at most once in each head and body atom, and Π includes all rules of the form $R(\dots, x, \dots, x, \dots) \to \bot$ for each EDB and IDB relation R;
2. the EDB schema $\mathbf{S}$ includes a relation symbol S_t of arity m for each $\mathbf{Q}$-type t and Π includes rules (i) $S_t(x_1, \dots, x_m) \to R(x_{i_1}, \dots, x_{i_m})$ for all $\mathbf{Q}$-types t and all $R(x_{i_1}, \dots, x_{i_m}) \in t$ and (ii) $S_t(x_1, \dots, x_m) \wedge R(x_{i_1}, \dots, x_{i_m}) \to \bot$ for all $\mathbf{Q}$-types t and atoms $R(x_{i_1}, \dots, x_{i_m}) \notin t$, where $R \in \mathbf{Q}$ and $x_{i_1} \dots x_{i_m}$ is a permutation of $x_1 \dots x_m$; the S_t relations are not allowed to be used in any other rule.[2]

Working with normalized programs will simplify the subsequent constructions.

▶ **Lemma 19.** *For every GDDLog program Π that admits precoloring, there is a normalized GDDLog-Program Π' with $\Pi \approx_{\mathsf{FO}} \Pi'$.*

Let Π be a normalized GDDLog program over schema $\mathbf{S}$, let $\mathbf{Q}$ be the set of IDB relations in Π and m the unique arity of relations in Π. We define a CQA setup $\mathrm{CQA}(C_\Pi, q_\Pi)$ from (DiC, UCQ) over schema $\mathbf{S}'$, that is, $\mathbf{S}$ extended with one relation Q_t of arity m for each $\mathbf{Q}$-type t. For an $\mathbf{S}'$-instance J, and a tuple $\mathbf{a} \in \mathsf{adom}(J)^m$, we say that $\mathbf{a}$ is *assigned* $\mathbf{Q}$*-type t in J* if there is a permutation $\mathbf{b}$ of $\mathbf{a}$ and a $\mathbf{Q}$-type $\widehat{t}$ such that (i) $Q_{t'}(\mathbf{b}) \in J$ if $t' \neq \widehat{t}$ for all $\mathbf{Q}$-types t' and (ii) $\widehat{t}$ is obtained from t by permuting the variables $x_1 \cdots x_m$ in the same way in which the constants in $\mathbf{a}$ are permuted in $\mathbf{b}$, that is, if $\mathbf{a} = a_1 \cdots a_m$ and $\mathbf{b} = a_{i_1} \cdots a_{i_m}$, then $\widehat{t} = t[x_{i_1} \cdots x_{i_m}/x_1 \cdots x_m]$. We say that J is *proper* if every $\mathbf{S}$-guarded tuple[3] $\mathbf{a} \in \mathsf{adom}(J)^m$ is assigned a unique $\mathbf{Q}$-type. A proper $\mathbf{S}'$-instance J represents an

[2] Note that this is different from requiring Π to admit precoloring because admitting precoloring is about i-types whereas for normalized programs we use $\mathbf{Q}$-types. In fact, a program in normal form cannot admit precoloring in the original sense because the conditions on the rules required for normality and admitting precoloring are incompatible.

[3] A tuple $\mathbf{a} \in \mathsf{adom}(I)^m$ is $\mathbf{S}$-*guarded* if I contains some fact $R(\mathbf{a})$ with $R \in \mathbf{S}$.

$\mathbf{S}$-instance and an $\mathbf{S} \cup \mathbf{Q}$-instance: the former is the reduct of J to schema $\mathbf{S}$ and the latter is obtained by starting with that reduct and then adding the fact $R(a_{i_1}, \dots, a_{i_m})$ whenever $\mathbf{a} = a_1 \cdots a_m \in \mathsf{adom}(J)^m$ is assigned $\mathbf{Q}$-type t and $R(x_{i_1}, \dots, x_{i_m}) \in t$. Intuitively, the latter instance is supposed to represent an extension of the former instance obtained by applying the rules in Π.

We now define $\mathrm{CQA}(C_\Pi, q_\Pi)$. The set C_Π contains one disjointness constraint, namely

$$\forall x_1 \cdots \forall x_m \neg\big(\bigwedge_{t \text{ a } \mathbf{Q}\text{-type}} Q_t(x_1, \dots, x_m) \big).$$

For a $\mathbf{Q}$-type t, we use $C_t(\mathbf{x})$ to denote the conjunction $\bigwedge_{t' \text{ a } \mathbf{Q}\text{-type}, t' \neq t} Q_{t'}(\mathbf{x})$. A CQ q over schema $\mathbf{S}'$ is *forbidden* if for each proper $\mathbf{S}'$-instance J such that the $\mathbf{S} \cup \mathbf{Q}$-instance I of J satisfies all non-goal rules in Π and no goal-rule of Π applies in I, we have $J \not\models q$. The UCQ q_Π consists of the following CQs:

1. all forbidden CQs q over schema $\mathbf{S}'$ such that
   a. the number of variables in q is bounded by the maximum number of variables in a rule body in Π;
   b. for every atom $R(\mathbf{x})$ there is a $\mathbf{Q}$-type t and a permutation $\mathbf{y}$ of $\mathbf{x}$ such that $C_t(\mathbf{y})$ is a subconjunction of q;
2. all CQs $C_t(\mathbf{x}) \wedge C_{t'}(\mathbf{y})$ such that $\mathbf{y}$ is a permutation of $\mathbf{x}$ and $t' \neq t[\mathbf{y}/\mathbf{x}]$.

▶ **Lemma 20.** *$\Pi \preceq_{\mathsf{FO}} CQA(C_\Pi, q_\Pi)$ and $CQA(C_\Pi, q_\Pi) \preceq_{\mathsf{FO}} \Pi$.*

Proof. For the first reduction, assume that an $\mathbf{S}$-instance I is given. Define an $\mathbf{S}'$-instance $T^{\uparrow}(I)$ as the extension of I with all facts $Q_t(\mathbf{a})$ such that t is a $\mathbf{Q}$-type and $\mathbf{a} \in \mathsf{adom}(I)^m$. We show in the appendix that $I \models \Pi$ iff $T^{\uparrow}(I) \models_{C_\Pi} q_\Pi$. Clearly, $T^{\uparrow}$ is FO-definable.

For the second reduction, let I be an $\mathbf{S}'$-instance. Let $T^{\downarrow}(I)$ be the $\mathbf{S}$-instance obtained from the reduct of I to schema $\mathbf{S}$ by the following sequence of operations:

1. drop each fact $R(\mathbf{a})$ such that for every permutation $\mathbf{b}$ of $\mathbf{a}$, there are distinct $\mathbf{Q}$-types t and t' such that neither $Q_t(\mathbf{b})$ nor $Q_{t'}(\mathbf{b})$ are in I;
2. add $S_t(\mathbf{a})$ for each $\mathbf{S}$-guarded tuple $\mathbf{a} \in \mathsf{adom}(I)^m$ that is assigned $\mathbf{Q}$-type t in I (the assignment need not be unique).

We show in the appendix that $I \models_{C_\Pi} q_\Pi$ iff $T^{\downarrow}(I) \models \Pi$. Again $T^{\downarrow}$ is clearly FO-definable. ◀

▶ **Theorem 21.** *For every GDDLog program Π that admits precoloring, there is an FO-equivalent problem $CQA(C, q)$ from (DiC,UCQ).*

We now turn towards showing that $(\mathrm{DiC}, \mathrm{UCQ}) \preceq_{\mathsf{FO}} \mathrm{coGMSNP}$. Let $\mathrm{CQA}(C, q)$ over schema $\mathbf{S}$ be given with C a set of disjointness constraints. Let m be the maximum arity of a relation in $\mathbf{S}$. Fix variables $x_1, \dots, x_m$. For $i \leq m$, an *i-type* is a set of atoms $R(x_1, \dots, x_i)$ with $R \in \mathbf{S}$ (all variables occur in fixed order and are distinct). We use tp_i to denote the set of all i-types. For an $\mathbf{S}$-instance I and $\mathbf{a} \in \mathsf{adom}(I)^i$, we use $\mathsf{tp}_I(\mathbf{a})$ denote the i-type realized at $\mathbf{a}$ in I, that is, the set of all atoms $R(x_1, \dots, x_i)$ such that $R(\mathbf{a}) \in I$.

We define a GDDLog program $\Pi_{C,q}$ over schema $\mathbf{S}' = \{R_t \mid t \in \mathsf{tp}_i, i \leq m\}$. The quantified relations in $\Pi_{C,q}$ are $\mathbf{S} \cup \{Q_t \mid t \in \mathsf{tp}_i, i \leq m\}$ where the arities are defined in the obvious way. The disjointness constraints in C assign to each i-type t and each sequence of variables $\mathbf{x} \in \{x_1, \dots, x_m\}^i$ (repetitions allowed) a set of possible minimal repairs (which are also i-types), denoted with $\mathsf{rep}_{\mathbf{x}}(t)$. Formally, $\mathsf{rep}_{\mathbf{x}}(t)$ are the minimal repairs of $\{R(\mathbf{x}) \mid R(x_1, \dots, x_i) \in t\}$ viewed as an instance. Note that different sequences $\mathbf{x}$ may give rise to different repair sets for the same i-type t since the constraints in C might use variables multiple times in the same atom. The rules of $\Pi_{C,q}$ are as follows:

1. for each i-type t, $i \leq m$, and each $\mathbf{x} \in \{x_1, \ldots, x_m\}^i$: $R_t(\mathbf{x}) \to \bigvee_{t' \in \mathsf{rep}_{\mathbf{x}}(t)} Q_{t'}(\mathbf{x})$
2. for each i-type t, $i \leq m$, and each $R(x_1, \ldots, x_i) \in t$: $Q_t(x_1, \ldots, x_i) \to R(x_1, \ldots, x_i)$
3. for all distinct i-types t, t', $i \leq m$: $R_t(x_1, \ldots, x_i) \wedge R_{t'}(x_1, \ldots, x_i) \to \mathsf{goal}()$
4. for each CQ q' in q: $q' \to \mathsf{goal}()$

▶ **Lemma 22.** *$CQA(C,q) \preceq_{\mathsf{FO}} \Pi_{C,q}$ and $\Pi_{C,q} \preceq_{\mathsf{FO}} CQA(C,q)$.*

Proof. For the first reduction, let an $\mathbf{S}$-instance I be given. Define an $\mathbf{S}'$-instance $T^{\uparrow}(I)$ that consists of all facts $R_{\mathsf{tp}_I(\mathbf{a})}(\mathbf{a})$ with $\mathbf{a} \in \mathsf{adom}(I)^i$, $i \leq m$. Clearly, $T^{\uparrow}$ is FO-definable. We show in the appendix that $I \models_C q$ iff $T^{\uparrow}(I) \models \Pi_{C,q}$.

For the second reduction, let an $\mathbf{S}'$-instance I be given. If $R_t(\mathbf{a}), R_{t'}(\mathbf{a}) \in I$ for any $\mathbf{a} \in \mathsf{dom}(I)^i$, $i \leq m$, and $t \neq t'$, then answer 'inconsistent'. Otherwise, replace each fact $R_t(\mathbf{a})$ with the set of facts $\{R(\mathbf{a}) \mid R(\mathbf{x}) \in t\}$, and call the result $T^{\downarrow}(I)$. Clearly, $T^{\downarrow}$ is FO-definable. We show in the appendix that $I \models \Pi_{C,q}$ iff $T^{\downarrow}(I) \models_C q$. ◀

▶ **Theorem 23.** *For every CQA problem in (DiC,UCQ), there is an FO-equivalent GDDLog program Π.*

8 Conclusion

We find it intriguing that very simple integrity constraints such as MDiCs and MGAVs combined with structurally simple queries such as tUCQs result in classes of CQA problems that are as difficult to analyze as CSPs, and that slight generalizations (such as DiCs) result in entering essentially unknown terrain (in the form of GMSNP). We believe that the CSP-CQA connection is not yet explored to the end. For example, the known non-dichotomy between PTime and NP for MMSNP extended with inequality [20] gives rise to the speculation that $(\text{MDiC}, \text{UCQ}^{\neq})$ might have no dichotomy between PTime and coNP either, where $\text{UCQ}^{\neq}$ denotes the class of UCQs that also admit inequality atoms. The proof of such a result will not be entirely simple, though, as it seems to require a version of Ladner's theorem that relates to Ladner's original theorem in a similar way in which the mortality problem of Turing machines relates to the halting problem. We leave this as future work.

Another interesting question is whether larger and more natural classes of CQA problems, such as those based on unrestricted denial constraints or on some form of functional dependency also have natural counterparts in the world of CSP and MMSNP. In fact, it is not even clear whether the correspondence between (MDiC, tUCQ) and coCSP can be extended from monadic disjointness constraints to monadic *denial* constraints in which relations are still required to be monadic, but where more than one variable can be used. For example, the constraint $\forall x \forall y \neg(A(x) \wedge B(y))$ is a monadic denial constraint, but not an MDiC. The main challenge is to deal with the problem that the 'yes'-instances of each CSP are closed under homomorphic pre-images while the 'no'-instances of CQA problems are not. Simply changing the monadic relations in the schema as in the proof of Theorem 5 is no longer sufficient because, in contrast to MDiCs, monadic denial constraints are not local to a single constant. We believe, however, that even if it should turn out that such differences prevent the CQA-CSP connection from gracefully extending beyond the classes of CQA problems considered here, it might still be possible to carry over techniques and intuitions from the CSP/MMSNP world, which has seen frantic development in the last decade.

Acknowledgements. We want to thank Manuel Bodirsky, Hubie Chen, and André Hernich for interesting and helpful discussions.

References

1 Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *Proc. of ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 31–41. ACM, 2009.

2 Marcelo Arenas and Leopoldo E. Bertossi. On the decidability of consistent query answering. In *Proc. of AMW*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

3 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS*, pages 68–79. ACM Press, 1999.

4 Albert Atserias. On digraph coloring problems and treewidth duality. In *Proc. of LICS*, pages 106–115, 2005.

5 Libor Barto and Marcin Kozik. Constraint satisfaction problems of bounded width. In *Proc. of FOCS*, pages 595–603, 2009.

6 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

7 Leopoldo E. Bertossi. Consistent query answering in databases. *SIGMOD Record*, 35(2):68–76, 2006.

8 Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008.

9 Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-order rewritability of atomic queries in Horn description logics. In *Proc. of IJCAI*. IJCAI/AAAI, 2013.

10 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: a study through disjunctive datalog, CSP, and MMSNP. In *Proc. of PODS*, pages 213–224. ACM, 2013.

11 Manuel Bodirsky, Hubie Chen, and Tomás Feder. On the complexity of MMSNP. *SIAM J. Discrete Math.*, 26(1):404–414, 2012.

12 Manuel Bodirsky and Florent Madeleine. Feder and Vardi's logic revisited. In preparation.

13 Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM*, 53(1):66–120, 2006.

14 Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS*, pages 260–271. ACM, 2003.

15 Jan Chomicki. Consistent query answering: Five easy pieces. In *Proc. of ICDT*, volume 4353 of *LNCS*, pages 1–17. Springer, 2007.

16 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

17 Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Computing consistent query answers using conflict hypergraphs. In *Proc. of CIKM*, pages 417–426. ACM, 2004.

18 David Cohen and Peter Jeavons. *The complexity of constraint languages*, chapter 8. Elsevier, 2006.

19 Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *Proc. of STOC*, pages 477–490. ACM, 1988.

20 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

21 Tomás Feder and Moshe Y. Vardi. Homomorphism closed vs. existential positive. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*, pages 311–320, 2003.

22 Gaëlle Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? In *Proc. of LICS*, pages 550–559. IEEE Computer Society, 2013.

23 Ralph Freese, Marcin Kozik, Andrei Krokhin, Miklós Maróti, Ralph KcKenzie, and Ross Willard. On Maltsev conditions associated with omitting certain types of local structures. In preparation. Manuscript available from `http://www.math.hawaii.edu/~ralph/Classes/619/OmittingTypesMaltsev.pdf`.

24 Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In *Proc. of ICDT*, volume 3363 of *LNCS*, pages 337–351. Springer, 2005.

25 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

26 Neil Immerman. *Descriptive complexity*. Springer, 1999.

27 Phokion G. Kolaitis and Enela Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.

28 Paraschos Koutris and Dan Suciu. A dichotomy on the complexity of consistent query answering for atoms with simple keys. In *Proc. of ICDT*, pages 165–176. OpenProceedings.org, 2014.

29 Gábor Kun. Constraints, MMSNP and expander relational structures. *Combinatorica*, 33(3):335–347, 2013.

30 Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007.

31 Benoit Larose and Pascal Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theor. Comput. Sci.*, 410(18):1629–1647, 2009.

32 Florent R. Madelaine. Universal structures and the logic of forbidden patterns. *Logical Methods in Computer Science*, 5(2), 2009.

33 Florent R. Madelaine and Iain A. Stewart. Constraint satisfaction, logic and forbidden patterns. *SIAM J. Comput.*, 37(1):132–163, 2007.

34 Jaroslav Nesetril. Many facets of dualities. In *Bonn Workshop of Combinatorial Optimization*, pages 285–302, 2008.

35 Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3), 2008.

36 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proc. of STOC*, pages 216–226. ACM, 1978.

37 Slawomir Staworko and Jan Chomicki. Consistent query answers in the presence of universal constraints. *Inf. Syst.*, 35(1):1–22, 2010.

38 Balder ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. On the data complexity of consistent query answering. In *Proc. if ICDT*, pages 22–33. ACM, 2012.

39 Jef Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *Proc. of PODS*, pages 179–190. ACM, 2010.

40 Jef Wijsen. Charting the tractability frontier of certain conjunctive query answering. In *Proc. of PODS*, pages 189–200. ACM, 2013.

41 Jef Wijsen. A survey of the data complexity of consistent query answering under key constraints. In *Proc. of FoIKS*, volume 8367 of *LNCS*, pages 62–78. Springer, 2014.

On the Data Complexity of Consistent Query Answering over Graph Databases

Pablo Barceló and Gaëlle Fontaine

Department of Computer Science
University of Chile
pbarcelo@dcc.uchile.cl, gaelle@dcc.uchile.cl

Abstract

Areas in which graph databases are applied – such as the semantic web, social networks and scientific databases – are prone to inconsistency, mainly due to interoperability issues. This raises the need for understanding query answering over inconsistent graph databases in a framework that is simple yet general enough to accommodate many of its applications. We follow the well-known approach of consistent query answering (CQA), and study the data complexity of CQA over graph databases for regular path queries (RPQs) and regular path constraints (RPCs), which are frequently used. We concentrate on subset, superset and symmetric difference repairs. Without further restrictions, CQA is undecidable for the semantics based on superset and symmetric difference repairs, and Π_2^{P}-complete for subset repairs. However, we provide several tractable restrictions on both RPCs and the structure of graph databases that lead to decidability, and even tractability of CQA. We also compare our results with those obtained for CQA in the context of relational databases.

1998 ACM Subject Classification H.2.3 Database Management – Query Languages

Keywords and phrases graph databases, regular path queries, consistent query answering, description logics, rewrite systems

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.380

1 Introduction

Query languages for graph databases are typically navigational, in the sense that they allow for recursively traversing the labeled edges while checking for the existence of a path whose label satisfies a particular regular condition (see, e.g., [39, 5]). The basic building block for navigational languages over graph databases is the class of *regular path queries*, or RPQs [14]. Each RPQ is a regular expression L, and its evaluation $L(G)$ over a graph database G corresponds to a binary relation that contains all pairs of nodes in G that are linked by some path whose label matches L. The evaluation problem for RPQs can be solved in NLogspace in *data complexity*; that is, when the RPQ is fixed (cf., [5]).

Although graph databases are schema-less, it is possible to enforce data consistency over them using *path constraints* [1, 9]. These constraints have been used in several scenarios that are based on the graph database paradigm, e.g., to express local knowledge about semi-structured data [1]; to enforce restrictions over object-oriented databases, XML and RDF [36, 12, 22, 3, 30]; and to capture ontological hierarchies in the context of description logics (DLs) [16, 17]. Here we concentrate on a simple class of path constraints based on RPQs that was introduced by Abiteboul and Vianu; namely, the *regular path constraints*, or RPCs [1]. An RPC is an expression of the form $L_1 \sqsubseteq L_2$, where L_1 and L_2 are RPQs. In the graph database and DL contexts, a graph database G satisfies $L_1 \sqsubseteq L_2$ if and only if $L_1(G) \subseteq L_2(G)$ [26, 16, 17] (but we also consider a more restrictive semantics for RPCs,

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 380–397

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

motivated by their application over semi-structured data, following the original proposal of Abiteboul and Vianu). RPCs are a constituent part of the semantics of graph data and can also be used in the optimization process for RPQ evaluation [1].

An important problem when dealing with dependencies is that databases might be *inconsistent*, i.e., the databases might fail to satisfy the integrity constraints. In the case of graph database applications, high levels of inconsistency appear due to interoperability and distribution (e.g., in RDF and social/scientific networks [27, 40]). As an example, inconsistency might arise while integrating several sources into a single RDF graph, or while performing statistical inference on a scientific or social network. This raises the need for developing an *inconsistency-tolerant* semantics for graph databases in a simple yet general framework that abstracts away from its many implementations. In order to tackle this problem, we use the widespread approach of *consistent query answering* (as first introduced in the seminal work of Arenas, Bertossi and Chomicki [2]), which we now describe.

The approach is based on the notion of *repair*, which represents a possible minimal way in which consistency over the data could be restored. More formally, a repair is a database that satisfies the constraints and "minimally differs" from the original database. In general, a database does not admit a unique repair. This leads to an inconsistency-intolerant semantics based on the *consistent answers* of a query, i.e., the answers that hold in every possible repair. The problem of computing consistent answers to a query is known as *consistent query answering* (CQA).

Here we study the data complexity of CQA over graph databases in the scenario in which queries are RPQs and constraints are RPCs. That is, we study the complexity of evaluating consistent answers over an inconsistent graph database G for a *fixed* RPQ L under a *fixed* set Γ of RPCs. We explain next the context and main contributions of our work.

The context. The complexity of CQA has received considerable attention over different data models, notions of repairs and classes of constraints. Under the relational model, for instance, this problem has been studied for set-based [2, 18, 24], cardinality-based [29], and attribute-based repairs [37]; for constraints expressed as traditional functional dependencies, inclusion dependencies and denial constraints (see, e.g., [11, 19, 25, 38, 28, 24]); and for constraints expressed as tuple-generating dependencies (tgds) and equality-generating dependencies (egds) that arise in the context of data integration and data exchange [18].

CQA has also been studied in depth in the DL context, starting from the work of Lembo and Ruzzi [31]. The DL semantics is *open-world* in nature, meaning that the non-presence of a fact is not sufficient to ensure that the negation of the fact holds. This implies that in the DL context the only meaningful set-based repairs are the *subset* repairs; i.e., those that allow to restore consistency with respect to the DL constraints (i.e., the ones in the *TBox*) only by *deleting* facts from the database (i.e., the *ABox*). It is worth mentioning that the notion of DL repair is slightly different to its relational counterpart: A DL repair is not a subinstance that satisfies the constraints in the TBox, but one that does not lead to a contradiction in conjunction with those constraints [31, 32].

Some applications of graph databases, such as RDF, are open-world in nature. However, graph databases are not tied to this interpretation and may also accommodate closed-world applications. Therefore, there should be no a priori restriction on the class of set-based repairs one allows in this context. We thus study CQA for graph databases under the three usual notions of set-based repairs: *subset*, *superset*, and *symmetric difference* repairs [18].

Our contributions. We first look at the data complexity of CQA over graph databases under subset repairs. The problem is in Π_2^{P}, and we prove that it is complete for this class in

restricted cases. Moreover, it remains intractable even under an approximation semantics (based on the intersection of all subset repairs) that is motivated by the DL context [32].

In order to deal with this high complexity, we provide tractable cases by restricting the class of RPCs or the class of graph databases allowed. In the first case, we prove that the problem is tractable if RPCs are in *LAV* form, i.e., if they are of the form $a \sqsubseteq L$, where a is a single letter. This result is, in a sense, tight, since allowing a single RPC of the form $ab \sqsubseteq c$, for symbols a, b and c, leads to intractability. In the case of restrictions on graph databases, by applying a deep result of Courcelle [20] we obtain that the data complexity of CQA under subset repairs is tractable over graph databases of *bounded treewidth*.

We then move to study CQA under superset and symmetric difference repairs, and prove that in both cases the problem is undecidable. However, we obtain decidability by either restricting the class of RPCs allowed or their semantics. Let us consider first the restrictions on RPCs. We prove that if RPCs are in LAV form then the data complexity of CQA is tractable when dealing with symmetric difference repairs. We leave open whether CQA is also decidable under the semantics of superset repairs, but prove that at least tractability in data complexity is not preserved. We then prove that if RPCs are in *GAV* form, i.e., if they are of the form $L \sqsubseteq a$, where a is a single letter, our CQA problem is tractable under the semantics of superset repairs, but intractable under symmetric difference repairs. With respect to restrictions on the semantics of RPCs, we prove that by forcing RPCs to be read from a particular node in the graph database, called the *origin* (which corresponds to the original semantics Abiteboul and Vianu defined for these constraints), the problem can be solved in coNP under superset repairs.

Comparison with previous results. The main difference between the query and constraint languages we study here (RPQs and RPCs) and the ones studied in the relational context, is that our languages allow recursion while CQA has been studied in the relational world mostly in the absence of recursive features. Interestingly, our results show that recursion does not add to the complexity of the problems studied. In fact, almost all of our lower bounds hold in the restricted setting in which RPQs do not mention the Kleene-star and RPCs are *word constraints* of the form $w_1 \sqsubseteq w_2$, for words w_1 and w_2 [1].

If we interpret graph databases as relational databases, word constraints can be represented as tgds. The tgds corresponding to word constraints have a special restricted structure and are called *chain* tgds. On the other hand, CQA under tgds has been intensively studied [18], and it is tempting to think that lower bounds obtained in such setting could be adapted to work in our scenario. This is not the case, however, as those proofs do not apply to the class of chain tgds. Actually, the proofs of our lower bounds are considerably more involved than the ones for arbitrary tgds. As a side-effect, we obtain that several of our lower bounds extend to CQA in the relational case for queries defined as unions of CQs under chain tgds.

DL databases are (essentially) graph databases. In such scenario, Π_2^{P}-hardness results have been obtained by Rosati for the data complexity of CQA under subset repairs, in particular, for the case when queries are unions of CQs and constraints are expressed in the logic $\mathcal{ALC}$ [35]. However, constraints in this logic cannot be directly expressed as RPCs due to the presence of negation. Furthermore, the notion of repair in [35] is different to ours.

Organisation of the paper. Preliminaries are in Section 2. Results about the subset repair semantics are in Section 3, and those about the semantics of superset and symmetric difference repairs are in Section 4. Comparison with previous work is in Section 5 and conclusions in Section 6. Due to space constraints, complete proofs are in the appendix.

2 Preliminaries

Graph databases and regular path queries. As it is customary in the graph database literature [14, 15, 39, 5], we consider graph databases to be finite, edge-labeled and directed graphs. Formally, let Σ be a finite alphabet. A *graph database* $G = (V, E)$ over Σ consists of a finite set V of nodes and a set of labeled edges $E \subseteq V \times \Sigma \times V$. We interpret each tuple $(u, a, v) \in E$, for $u, v \in V$ and $a \in \Sigma$, as an edge from node u to node v whose label is a. If $G = (V, E)$ and $G' = (V', E')$ are graph databases over Σ, we write $G \subseteq G'$ to denote that $V \subseteq V'$ and $E \subseteq E'$. If in addition it is not the case that $G' = G$, we write $G \subsetneq G'$.

Navigational query languages for graph databases, such as the one of RPQs, express properties of *paths*. Formally, a path π in $G = (V, E)$ of length m (where $m > 0$) from node v_0 to node v_m is a sequence of the form $(v_0, a_1, v_1)(v_1, a_2, v_2) \dots (v_{m-1}, a_m, v_m)$, where (v_{i-1}, a_i, v_i) is an edge in E, for each $1 \leq i \leq m$. The *label* of π, denoted $\lambda(\pi)$, is the string $a_1 a_2 \dots a_m \in \Sigma^*$. A path π of length 0 is simply a node v and its label $\lambda(\pi)$ is the empty string ε.

Here we concentrate on the simplest navigational language for graph databases, namely, *regular path queries*, or RPQs [14]. An RPQ is a regular expression L over Σ. The evaluation $L(G)$ of L over a graph database $G = (V, E)$ is the set of pairs (u, v) of nodes in V for which there is a path π in G from u to v such that $\lambda(\pi)$ satisfies L. If L does not mention the Kleene-star (i.e., if L defines a finite language) then we say that L is *non-recursive*. It is well-known (cf., [5]) that computing $L(G)$, for an RPQ L and a graph database G, can be solved in polynomial time (and in NLogspace if L is fixed, that is, in data complexity).

Regular path constraints. Graph database constraints based on the class of RPQs are known as *regular path constraints* [1, 26], or RPCs. Formally, an RPC over Σ is an expression of the form $L_1 \sqsubseteq L_2$, where L_1 and L_2 are RPQs over Σ. A *word constraint* is an RPC in which both L_1 and L_2 are words.

An RPC $L_1 \sqsubseteq L_2$ expresses that the evaluation of the RPQ L_1 is contained in the evaluation of the RPQ L_2 [26]. Formally, a graph database G *satisfies* $L_1 \sqsubseteq L_2$, denoted $G \models L_1 \sqsubseteq L_2$, if and only if $L_1(G) \subseteq L_2(G)$. If Γ is a finite set of constraints, we write $G \models \Gamma$ to denote that for each RPC $L_1 \sqsubseteq L_2$ in Γ it is the case that $G \models L_1 \sqsubseteq L_2$. It follows from previous remarks that the problem of checking whether $G \models \Gamma$, for a fixed set Γ of RPCs, is in NLogspace.

▶ **Example 1.** Let Γ be the following set of RPCs:

1. $\texttt{child\_of} \sqsubseteq \texttt{son\_of} \cup \texttt{daughter\_of}$.
2. $\texttt{brother\_of} \cdot (\texttt{brother\_of} \cup \texttt{sister\_of}) \sqsubseteq \texttt{brother\_of}$.
3. $\texttt{sister\_of} \cdot (\texttt{brother\_of} \cup \texttt{sister\_of}) \sqsubseteq \texttt{sister\_of}$.
4. $\texttt{child\_of} \cdot (\texttt{brother\_of} \cup \texttt{sister\_of}) \sqsubseteq \texttt{is\_nephew} \cup \texttt{is\_niece}$.

Intuitively, the first RPC expresses that if u is a child of v, then u is a son or a daughter of v. The second and third RPCs express that if u is a brother (resp., sister) of v and v is a sibling of w, then v is a brother (resp., sister) of w. The forth RPC states that each child of a person v is the niece or nephew of every sibling of v. ◀

Repairs. A *repair* of a database D under a set of constraints Γ is a database D' that satisfies Γ but "minimally" differs from D [2]. We formalise this idea for graph databases and RPCs below, following closely its formalisation in the relational context [2].

The *symmetric difference* between two relational databases D and D' is defined as a database $D \oplus D'$ that contains those "facts" that belong to D but not to D' or to D' but

not to D. We can analogously define the symmetric difference between two graph databases $G = (V, E)$ and $G' = (V', E')$ over the same alphabet Σ, as the graph database

$$G \oplus G' \;:=\; \big(V_0,\, (E \oplus E')\big),$$

where $E \oplus E' := (E \setminus E') \cup (E' \setminus E)$ and V_0 is the set of nodes occurring in $E \oplus E'$. That is, the nodes (resp., edges) of $G \oplus G'$ are those nodes (resp., edges) that appear in either G or G' but not in both. Notice that $G \oplus G'$ is also a graph database over Σ.

Three notions of set-based repairs have been studied in the literature [18]: the *subset*, *superset* and *symmetric difference* repairs ($\subseteq$–, $\supseteq$–, and $\oplus$–repairs, respectively). We introduce them below in the context of graph databases. Let $G = (V, E)$ and $G' = (V', E')$ be two graph databases over Σ, and assume that Γ is a finite set of RPCs over Σ. Then:

1. G' is a $\oplus$-repair (i.e., symmetric difference repair) of G under Γ, if (1) $G' \models \Gamma$, and (2) there is no graph database G'' over Σ such that $G'' \models \Gamma$ and $G \oplus G'' \subsetneq G \oplus G'$.
2. G' is a $\subseteq$-repair (i.e., subset repair) of G under Γ, if $G' \subseteq G$ and G' is a $\oplus$-repair of G. Equivalently, if (1) $G' \subseteq G$, (2) $G' \models \Gamma$, and (3) there is no graph database G'' over Σ such that $G'' \models \Gamma$ and $G' \subsetneq G'' \subseteq G$.
3. G' is a $\supseteq$-repair (i.e., superset repair) of G under Γ, if $G \subseteq G'$ and G' is a $\oplus$-repair of G. Equivalently, if (1) $G \subseteq G'$, (2) $G' \models \Gamma$, and (3) there is no graph database G'' over Σ such that $G'' \models \Gamma$ and $G \subseteq G'' \subsetneq G'$.

▶ **Example 2** (Example 1 cont.)**.** Consider a graph database G whose set of edges is

$$\{(a, \texttt{child\_of}, b), (b, \texttt{sister\_of}, c), (c, \texttt{brother\_of}, d)\}.$$

Then G has two $\subseteq$-repairs under Γ: $\{(b, \texttt{sister\_of}, c)\}$ and $\{(c, \texttt{brother\_of}, d)\}$. On the other hand, G has eight $\supseteq$-repairs under Γ. One of them is the one that extends G with edges:

$$\{(a, \texttt{son\_of}, b), (b, \texttt{sister\_of}, d), (a, \texttt{is\_nephew}, c), (a, \texttt{is\_nephew}, d)\}.$$

Finally, G has seven $\oplus$-repairs under Γ that are neither $\subseteq$-repairs nor $\supseteq$-repairs. One of them is $\{(b, \texttt{sister\_of}, c), (c, \texttt{brother\_of}, d), (b, \texttt{sister\_of}, d)\}$. ◀

Repairs might not exist in some situations. Consider an RPC of the form $L \sqsubseteq \varepsilon$, where ε is the empty word, and a graph database G that consists of nodes u and v linked by a path labeled in L. Assume that G has a $\supseteq$-repair H. Then it must be the case that $u = v$ in H, which is impossible. As the the next lemma shows, repairs exist in all other cases.

▶ **Lemma 3.** *Let $G = (V, E)$ be a graph database and Γ a finite set of RPCs over Σ. Then:*

1. *There is a $\subseteq$-repair and a $\oplus$-repair of G under Γ.*
2. *If Γ contains no RPC of the form $L \sqsubseteq \varepsilon$, then there is a $\supseteq$-repair of G under Γ.*

Proof. Let $G = (V, E)$ be a graph database and Γ a finite set of RPCs. The empty database $G_\emptyset = (\emptyset, \emptyset)$ satisfies Γ. Hence, there is an $\subseteq$-repair H of G under Γ such that $G_\emptyset \subseteq H \subseteq G$. By definition, H is also a $\oplus$-repair of G under Γ. Consider now the graph database $G_c = (V, V \times \Sigma \times V)$. It is easy to see that G_c satisfies Γ since Γ contains no RPC of the form $L \sqsubseteq \varepsilon$. Since $G \subseteq G_c$, there is a $\supseteq$-repair H of G under Γ such that $G \subseteq H \subseteq G_c$. ◀

Consistent query answering. We are now ready to define our most important notion, that of a *consistent answer* to an RPQ. The consistent answers are the pairs of nodes that belong to the evaluation of the RPQ over every single repair of the original graph database.

▶ **Definition 4** (Consistent answers). Assume that $\star \in \{\oplus, \subseteq, \supseteq\}$. Let $G = (V, E)$ be a graph database, Γ a set of RPCs and L an RPQ, all of them over Σ. We define the set $\star$-Cons(G, L, Γ) of *$\star$-consistent answers of L over G under Γ* as:

$$\star\text{-Cons}(G, L, \Gamma) \;=\; \bigcap \{L(G') \mid G' \text{ is a } \star\text{-repair of } G \text{ under } \Gamma\}. \quad \blacktriangleleft$$

▶ **Example 5.** (Example 1 cont.) Consider the RPQ $L = \texttt{child\_of} \cdot \texttt{sister\_of}$. The pair (a, d) belongs to $\supseteq$-Cons(G,L,Γ). This pair also belongs to $\supseteq$-Cons(G,L',Γ), for $L' = \texttt{is\_nephew} \cup \texttt{is\_niece}$. On the other hand, the only way in which a pair (u, v) can belong to $\subseteq$-Cons(G,L'',Γ), for an RPQ L'', is when $u = v = c$ and $L'' = \varepsilon$. $\blacktriangleleft$

Here we study the data complexity of the problem of computing certain answers. We formalise this decision problem as follows. Assume that L is an RPQ and Γ is a finite set of RPCs over Σ. We denote by $\star$-CQA(L,Γ) the problem of, given a graph database $G = (V, E)$ over Σ and a pair (u, v) of nodes in V, checking whether $(u, v) \in \star$-Cons(G, L, Γ).

3 CQA under Subset Repairs

We start by proving that under the subset repair semantics our CQA problem is Π_2^{P}-complete. This holds even in the case in which all RPCs are word constraints and the RPQ is non-recursive.

▶ **Theorem 6.**

1. *For each RPQ L and finite set Γ of RPCs over the same alphabet Σ, it is the case that $\subseteq$-CQA(L,Γ) is in Π_2^{P}.*
2. *There exist a finite alphabet Σ, a non-recursive RPQ L and a finite set Γ of word constraints over Σ, such that $\subseteq$-CQA(L,Γ) is Π_2^{P}-complete.*

Proof. We sketch the hardness proof, i.e., that there is a non-recursive RPQ L and a set Γ of word constraints such that $\subseteq$-CQA(L,Γ) is Π_2^{P}-hard. We do so by providing a reduction from the quantified boolean satisfaction problem for Π_2^{P} to $\subseteq$-CQA(L,Γ).

Let ϕ be a quantified boolean formula of the form $\forall X \exists Y \psi$, where X, Y are disjoint sets of variables and ψ is of the form $(z_{11} \vee z_{12} \vee z_{13}) \wedge \cdots \wedge (z_{n1} \vee z_{n2} \vee z_{n3})$, for $z_{ij} \in \{x, \neg x, y, \neg y \mid x \in X, y \in Y\}$ $(1 \leq i \leq n,\ 1 \leq j \leq 3)$. For all i, j, we define u_{ij} as the variable z_{ij} if $z_{ij} \in X \cup Y$, or we define u_{ij} as u if z_{ij} is of the form $\neg u$. Without loss of generality, we may assume that in each clause $C_i := z_{i1} \vee z_{i2} \vee z_{i3}$, there is at least one variable in[1] Y.

We will define the RPQ L, the constraints Γ and associate a graph database G_ϕ with ϕ in such a way that

$$\phi \text{ is satisfiable} \quad \text{iff} \quad (n_s, n_s) \in \subseteq -\text{Cons}(G_\phi, L, \Gamma).$$

where n_s is a node of G_ϕ.

[1] Suppose for example that the clause C_1 of ψ contains only variables in X. We construct a new formula ψ' defined by $(z_{11} \vee z_{12} \vee y) \vee (\neg y \vee z_{13} \vee z_{13}) \vee C_2 \wedge \cdots \wedge C_n$, where y is a new fresh variable. We define Y' as $Y \cup \{y\}$. Then $\forall X \exists Y \psi$ is satisfiable iff $\forall X \exists Y' \psi'$ is satisfiable.

It is convenient to start defining a graph database G'_ϕ over alphabet

$$\Sigma \;=\; \{t, f, t_0, f_0, a, y, z, w, d, s, e, r_{-,-}, r_{-,+}, r_{+,-}, r_{+,+}\}.$$

The nodes of G'_ϕ are the following set:

$$V \;:=\; \{n_{ij} \mid 1 \leq i \leq n,\, 1 \leq j \leq 3\} \;\cup\; \{n_t, n_f, n_s\}.$$

With each variable u_{ij}, we associate a node id n_{ij} in V. Observe that even if $u_{ij} = u_{kl}$ (with $(i,j) \neq (k,l)$), the associated nodes are distinct. We also have three special nodes n_s, n_t and n_f. We let $<$ be an arbitrary irreflexive partial order over $\{1,\ldots,n\} \times \{1,2,3\}$ such that (a) for each $1 \leq i,k \leq n$ and $1 \leq j,l \leq 3$, if $u_{ij} = u_{kl}$, then $(i,j) < (k,l)$ or $(k,l) < (i,j)$, and (b) for each (i,j), where $1 \leq i \leq n$ and $1 \leq j \leq 3$, there is at most one pair (k,l) such that (k,l) is a "successor" of (i,j) with respect to $<$ (where we say that (k,l) is a *successor* of (i,j) if $(i,j) < (k,l)$ and there is no (k',l') such that $(i,j) < (k',l') < (k,l)$).

The set of edges of G'_ϕ is defined as the union of all the sets below:

$$E_t = \{(n_{ij}, t, n_{ij}) \mid 1 \leq i \leq n,\, 1 \leq j \leq 3\} \qquad E_f = \{(n_{ij}, f, n_{ij}) \mid 1 \leq i \leq n,\, 1 \leq j \leq 3\}$$
$$E_{t_0} = \{(n_t, t_0, n_t)\} \qquad E_{f_0} = \{(n_f, f_0, n_f)\}$$
$$E_a = \{(n_{ij}, a, n_s) \mid u_{ij} \in Y\} \qquad E_y = \{(n_s, y, n_s)\}$$
$$E_z = \{(n_{ij}, z, n_t), (n_{ij}, z, n_f) \mid u_{ij} \in Y\} \qquad E_w = \{\{(n_t, w, n_s), (n_f, w, n_s)\}$$
$$E_d = \{(n_{ij}, d, n_{kl}) \mid (i,j) < (k,l)\} \qquad E_s = \{(u, s, v) \mid u, v \in V\}$$
$$E_{r_{\sim\sharp}} = \{(n_{ij}, r_{\sim\sharp}, n_{i(j+1)}) \mid z_{ij} = {\sim} u_{ij},\, z_{i(j+1)} = \sharp u_{i(j+1)},\, 1 \leq i \leq n,\, 1 \leq j \leq 2\}$$
$$E_e = \emptyset$$

where $\sim, \sharp$ belong to $\{+,-\}$ and $+u := u$, $-u := \neg u$.

The intuition is as follows. There is no edge with label e and there is an edge with label s between any two nodes. For the labels t and f, each node n_{ij} associated with a variable admits loops with labels t and f. The $\subseteq$-repairs will be such that each such node admit at most one loop, either with label t or f. This allows us to define a partial map associating a truth value with each node n_{ij} ($\top$ if the loop has label t and $\bot$ if the loop has label f). The special node n_t has a loop with label t_0 and the special node n_f has a loop with label f_0.

The labels r_{++}, r_{-+}, r_{+-} and r_{--} express which variables occur in the same clause and whether each variable occurs positively or negatively. The label d specifies which nodes corresponds to the same variable. The constraints will be such that in a $\subseteq$-repair, two nodes linked by d (corresponding to the same variables) admit loops with the same label (t or f). This implies that the partial mapping associating a truth value to each node ($\top$ if the loop has label t and $\bot$ if the loop has label f), corresponds to a partial valuation over the variables in $X \cup Y$.

The special node n_s admits a loop with label y. That loop will act as a "witness". If that loop appears in a repair, we will say that the node n_s gets *activated*. When n_s is activated in a repair, the idea is that each node n_{ij} associated with a variable in Y admits a loop with label t or f. That is, each such a node receives a truth value ($\top$ if the loop has label t and $\bot$ otherwise). Conversely, if in a repair one node associated with a variable in Y admits a loop with label t or f, then the presence of that loop "activates" the node n_s (i.e. n_s admits a loop with label y). As a consequence, all the nodes associated with variables in Y will admit a loop. Basically, the node n_s guarantees that either all the nodes associated with variables in Y admit a loop, or none of those nodes admits a loop.

In order to encode in the graph database which variables belong to Y, we add an edge with label a between the source n_s and each node of the form n_{ij} with $u_{ij} \in Y$. We also add an edge with label z from n_{ij} to the special node n_t and from n_{ij} to n_f.

We use the nodes n_t and n_f in the following way. Recall that in a given $\subseteq$-repair, the (possible) truth value of a node is given by the label (either t or f) of the loop of the node. Now, for the nodes associated with variables in Y, we have an extra way of encoding the truth value. If the truth value of a node n_{ij} (with $u_{ij} \in Y$) is true, this will also be witnessed by an arrow with label z from n_{ij} to the special node n_t (and the other arrow with label z from n_{ij} to n_f is deleted). If the truth value is false, this will be witnessed by an arrow with label z from n_{ij} to n_f.

We now define the set Γ of constraints. We have six different sets of constraints (C1), (C2), (C3), (C4), (C5) and (C6) in Γ. The set (C2) contains the constraint

$$tf \sqsubseteq e.$$

Since e is empty, it says that a node cannot admit both a loop with label t and f. The set (C3) contains the two constraints

$$td \sqsubseteq dt$$
$$fd \sqsubseteq df.$$

It expresses that if two nodes are linked by d (hence, they are associated with the same variable), then they must admit a loop with the same label (in case they admit a loop). This guarantees that the partial map associating a truth value to each node (depending on the label of the loop), can be transformed into a partial valuation over the variables. The set of constraints (C4) is given by

$$F(j)r_{jk}F(k)r_{kl}F(l) \sqsubseteq e,$$

where $j, k, l \in \{+, -\}$, $F(+) = f$ and $F(-) = t$. These constraints express that the formula ψ is not false under the partial valuation associated with the repair.

The constraints (C5) are given by

$$zt_0 \sqsubseteq tz$$
$$zf_0 \sqsubseteq fz.$$

They express that if a node associated with a variable in Y is linked to the node n_t, then it admits a loop with label t. Similarly, if such a node is linked to the node n_f, then it admits a node with label f.

Indeed, let us look for example at the constraint $zt_0 \subseteq tz$. Suppose that there is a path with label zt_0 from a node u to a node v. By definition of t_0 and z, this can only happen if u is a node of the form n_{ij} (with $u_{ij} \in Y$) and v is the node n_t. Now since $zt_0 \subseteq tz$, this implies that n_{ij} admits a loop with label t.

The set (C6a) contains the two constraints

$$ta \sqsubseteq ay$$
$$fa \sqsubseteq ay$$

while the constraint (C6b) is given by $ay \subseteq zw$. The constraints (C6a) express that if one node associated with a variable in Y admits a loop with label t or f, then the node n_s is activated (i.e. admits a loop with label y). Indeed, if there is a path with label ta from a node u to a node v, then, u must be a node of the form n_{ij} admitting a loop with label t and v is the node n_s. Moreover, since n_{ij} admits an outgoing edge with label a, n_{ij} must be associated with a variable in Y. Since $ta \subseteq ay$, the presence of the loop with label t and the fact that u_{ij} belongs to Y imply that there is an edge with label y. That is, n_s is activated.

The constraint (C6b) expresses that if the node n_s is activated (admits a loop with label y), then each node n_{ij} associated with a variable in Y is linked either to the node n_t or to the node n_f. Together with the contraints (C5), this means that if n_s is activated, then each node n_{ij} associated with a variable in Y admits a loop with label t or f.

We define an RPQ L' as sys. This RPQ evaluates to $V \times V$ in the $\subseteq$-repairs admitting a y-labeled edge, i.e., the repairs in which n_s is activated. Recall that this means that the partial valuation associated with the repair assigns a truth value to all the variables in Y.

We would like to prove that ϕ is satisfiable iff (n_s, n_s) belongs to $\subseteq$-Cons(G'_ϕ, L', Γ). Now the implication from left to right is not true. The problem is that since we are allowed to delete edges in order to obtain a repair, we may loose some "relevant information". For example, in a repair, we may loose an edge with label a encoding the fact that a node is associated with a variable in Y. In such repairs, it might not be the case that (n_s, n_s) belongs to the answer of L'.

The solution is to extend the graph database G'_ϕ with a set of extra edges E, define a new graph G_ϕ, and add a set of constraints (C1). If in a repair H of G_ϕ we lost some "relevant information", the constraints (C1) will be such that at least one extra edge in E occurs in H. We then modify the RPQ L' into an RPQ L, in such a way that if a graph database contains a new edge in E, then the evaluation of L consists of all the pairs of nodes. Hence, in all the repairs in which we lose some relevant information, the answer of the RPQ contains any pair of nodes.

As an example, we show how to modify the graph G'_ϕ into a graph G_a, add new constraints (C1d) and modify the RPQ L into a RPQ L_a in such a way that if in a repair H of G_a, we "lost" an edge with label a, then H trivially satisfies[2] L_d. We define G_a by adding the following edges

$$E_{a'} = \{(n_{ij}, a', n_{ij}) \mid (n_{ij}, a, n_s) \in E_a\}.$$

That is, we add a loop with label a' to all the nodes who must admit an outgoing edge with label a. We define (C1d) as the following constraint

$$a'a \subseteq e.$$

Using the maximality property of the repairs, we can show that this constraint ensure that in each repair H, exactly one of the following properties holds: (a) we did not loose any edge with label a, and there is no edge with label a' or e, (b) there is an edge with label a' or e. Case (a) means that we did not lost any "relevant information" with respect to the label a. In case (b), the fact that we lost that information is witnessed by the fact that in the repair, there is an edge with label a' or e.

Finally we let L_a be the RPQ $s(y + a' + e)s$. That is, L_a is equivalent to $sys + s(a' + e)s$. There are two possibilities for a repair to be such that (n_s, n_s) belongs to L_a: either (n_s, n_s) belongs to L' or there is an edge with label a' or e. Those two possibilities corresponds to cases (a) and (b) of the previous paragraph. ◀

Semantics based on the intersection of subset repairs. In order to obtain good complexity bounds for CQA in the DL context, Lembo et al. introduced a sound approximation of the CQA semantics based on the idea of evaluating queries over the intersection of all

[2] For a complete definition of G, (C1) and L, we should make such modifications for the labels d, r_{++}, r_{-+}, r_{+-}, r_{--}, t_0, f_0 and w.

subset repairs [32]. This approximation leads to tractability for inconsistency-tolerant query answering over some DLs of interest (e.g., for *DL-Lite*$_{\mathcal{A}}$). Although the repairs studied in the DL context are different to ours, it makes sense to study the pertinence of this semantics as a tool for establishing tractability results also in the graph database context.

Formally, let L be an RPQ and Γ a finite set of RPCs over Σ. We define $\cap$-CQA(L,Γ) as the problem of, given a graph database $G = (V, E)$ over Σ and a pair (u, v) of nodes in V, checking whether (u, v) belongs to the evaluation of L over the intersection of all $\subseteq$-repairs of G under Γ. (The intersection of graph databases $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ over Σ is the graph database $(V_1 \cap V_2, E_1 \cap E_2)$).

We prove below that the approximation semantics is not easier to evaluate than the original one. Therefore, in order to obtain tractability for our CQA problem it is necessary to look for restrictions that are proper to the scenario of graph databases.

▶ **Proposition 7.** *There exist a finite alphabet Σ, a non-recursive RPQ L and a finite set Γ of word constraints over Σ, such that $\cap$-CQA(L,Γ) is Π_2^{P}-hard.*

3.1 Tractable restrictions

Due to the inherent high complexity of our CQA problem under the subset repair semantics, it is important to look for meaningful restrictions leading to tractability. We provide two such restrictions in this section. The first one is based on the class of LAV RPCs, and the second one on the class of graph databases of bounded treewidth.

Restricting RPCs. In the relational case, the data complexity of CQA for unions of CQs under the class of *LAV tgds* (i.e., tgds with a single atom in the left-hand side [33]) is tractable. This actually holds for any of the three repair semantics [18]. The direct analogue of LAV tgds in our setting is the class of *LAV* RPCs, which are RPCs with a single symbol in the left-hand side. Formally, a LAV RPC over Σ is an RPC of the form $a \sqsubseteq L$, where $a \in \Sigma$ and L is an RPQ over Σ. We can leverage the techniques used to study CQA under LAV tgds to prove tractability in data complexity for our CQA problem under LAV RPCs.

▶ **Theorem 8.** *For each RPQ L and finite set Γ of LAV RPCs over the same alphabet Σ, it is the case that $\subseteq$-CQA(L,Γ) is in* NLogspace.

It is interesting to also consider RPCs based on the class of *GAV* tgds [33], that only allow for one symbol on the right-hand side. That is, a GAV RPC over Σ is of the form $L \sqsubseteq a$, for L an RPQ over Σ and $a \in \Sigma$. While this restriction improves the complexity of the CQA problem, it does not lead to tractability.

▶ **Proposition 9.**

1. *For each RPQ L and finite set Γ of GAV RPCs over the same alphabet Σ, it is the case that $\subseteq$-CQA(L,Γ) is in* coNP.
2. *There exist a finite alphabet Σ, a non-recursive RPQ L over Σ and a single GAV RPC of the form $ab \sqsubseteq c$, where $a, b, c \in \Sigma$, such that $\subseteq$-CQA(L,Γ) is* coNP*-complete.*

Note that in the setting of relational databases, the data complexity of consistent query answering for unions of conjunctive queries with respect to GAV constraints was also shown to be complete for the class coNP [18]. It is worth mentioning that we obtained the upper bound of Proposition 9 using techniques from the relational case.

The second part of the previous proposition shows that, in a sense, the tractability result for LAV RPCs in Theorem 8 is optimal: Allowing two-letter words on the left-hand side of RPCs leads to intractability, even if the right-hand side consists of a single letter.

Restricting graph databases. Our CQA problem under the subset repair semantics can be reformulated as a *monadic second-order logic* (MSO) evaluation problem over a relational representation of graph databases. This allows us to apply results establishing the tractability of MSO over structures of bounded treewidth [20]. From those results, we obtain tractability for the CQA problem over graph databases of bounded treewidth.

Formally, let $G = (V, E)$ be a graph database. A *tree decomposition* of G is a pair (T, λ), where T is a tree and $\lambda : T \to 2^V$ maps each node t in T to a nonempty set $\lambda(t)$ of nodes in V, that satisfies the following conditions:

- The set $\{t \in T \mid v \in \lambda(t)\}$ is a connected subset of T.
- For each edge $(u, a, v) \in E$, it is the case that $\{u, v\} \subseteq \lambda(t)$, for some $t \in T$.

The *width* of the tree decomposition (T, λ) is $\max\{|\lambda(t)| - 1 \mid t \in T\}$. The *treewidth* of G is the minimum width of a tree decomposition of G. For instance, the treewidth of G is one if and only if the underlying undirected graph of G is a tree.

We then obtain the following:

▶ **Theorem 10.** *Let L be an RPQ and Γ a set of RPCs. Then $\subseteq$-CQA(L,Γ) can be solved in linear time over graph databases of treewidth $\leq k$, for each $k \geq 1$.*

4 CQA under Superset and Symmetric Difference Repairs

We prove in this section that our CQA problem is undecidable under the semantics of $\supseteq$- and $\oplus$-repairs. This holds even if RPQs are non-recursive and RPCs are word constraints:

▶ **Theorem 11.** *Assume $\star \in \{\supseteq, \oplus\}$. There exist a finite alphabet Σ, a non-recursive RPQ L and a set Γ of word constraints over Σ, such that $\star$-CQA(L,Γ) is undecidable.*

In order to prove this theorem we establish a connection with the *implication problem* for RPCs [1, 26]. Recall that this is the problem of, given a finite set Γ of RPCs and an RPC $L_1 \sqsubseteq L_2$, checking whether $\Gamma \models L_1 \sqsubseteq L_2$, i.e., if $G \models \Gamma$ implies $G \models L_1 \sqsubseteq L_2$, for every graph database G. Grahne and Thomo proved this problem to be undecidable, even for word constraints, using a reduction from the *word rewriting* problem [26]. We develop nontrivial adaptations of such reduction to prove Theorem 11. The reason why we have to develop such adaptations is that there exist differences in nature between CQA and the implication problem for constraints. First, in the CQA problem we do not reason about all graph databases that satisfy the constraints (as in the case of the implication problem), but only about those that minimally differ from the original graph database. Note that in the special case of the superset semantics, this first problem does not apply. Indeed, given a graph database G, the intersection of the answers of a *monotone* query L over all the $\supseteq$-repairs is equal to the intersection of the answers of L over all the databases containing G and satisfying the constraints.

Second, we study the data complexity of the CQA problem, and, therefore, our goal is to prove undecidability of CQA for a *fixed* set of RPCs and a *fixed* RPQ. This is different to the case of the implication problem in which RPCs and RPQs define the input, and, therefore, cannot be fixed.

Proof of Theorem 11. We only prove the case when $\star$ is $\supseteq$. We start by recalling the basic notions of rewrite systems. Let Δ be a finite alphabet. A *semi-Thue rewrite system* $\mathcal{R}$ over Δ is a finite subset of $\Delta^* \times \Delta^*$. A rewrite system $\mathcal{R}$ induces a single-step reduction relation $\to_{\mathcal{R}}$ over Δ^* defined as:

$$\to_{\mathcal{R}} = \{(v, w) \mid v = xty, w = xuy, \text{ for some } (t, u) \in \mathcal{R} \text{ and } x, y \in \Delta^*\}.$$

We let $\rightarrow_{\mathcal{R}}^*$ be the reflexive transitive closure of $\rightarrow_{\mathcal{R}}$. The problem of testing whether a pair (u, v) belongs to $\rightarrow_{\mathcal{R}}^*$ is called the *rewrite problem* for $\mathcal{R}$. It is well known that there is a semi-Thue rewrite system $\mathcal{R}_0$ over Δ such that the rewrite problem for $\mathcal{R}_0$ is undecidable (see e.g., [8]).

We prove that $\supseteq$-CQA(L,Γ) is undecidable using a reduction from the rewrite problem for $\mathcal{R}_0$. Let $\hat{\Delta} = \Delta \cup \{\hat{a} \mid a \in \Delta\} \cup \{\$\}$, where the $\hat{a}$'s and \$ are fresh symbols. We define a set Γ of RPCs and an RPQ L over $\hat{\Delta}$, such that there is an algorithm that takes as input two words w_1 and w_2 over Δ and constructs a graph database G over $\hat{\Delta}$ with a node n_0 such that $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$ iff $(n_0, n_0) \in \supseteq$-Cons(G,L,Γ). The set Γ consists of all RPCs of the form:

$$\begin{aligned} u &\sqsubseteq v, \\ a\$\hat{a} &\sqsubseteq \$, \end{aligned}$$

for $(u, v) \in \mathcal{R}_0$ and $a \in \Delta$. We define the RPQ L as the letter \$.

Let w_1 and w_2 be two words in Δ^*. We assume that

$$\begin{aligned} w_1 &= w_{11} w_{12} \dots w_{1k}, \\ w_2 &= w_{21} w_{22} \dots w_{2l}, \end{aligned}$$

where $w_{1i}, w_{2j} \in \Delta$ $(1 \leq i \leq k,\ 1 \leq j \leq l)$. We define the graph database $G = (V, E)$ over $\hat{\Delta}$ as follows. The set V of nodes of G is $\{n_i : 0 \leq i \leq k\} \cup \{m_i : 0 < i < l\}$. Let us define $m_0 = n_0$ and $m_l = n_k$. Then the set E of edges of G is defined as $E_1 \cup E_2 \cup E_3$, where E_1, E_2 and E_3 are as follows:

$$\begin{aligned} E_1 &= \{(n_{i-1}, a, n_i) \mid w_{1i} = a,\ 1 \leq i \leq k\}, \\ E_2 &= \{(m_i, \hat{a}, m_{(i-1)}) \mid w_{2i} = a,\ 1 \leq i \leq l\}, \\ E_3 &= \{(n_k, \$, n_k)\}. \end{aligned}$$

Notice that G consists of a path with label w_1 from n_0 to n_k and a path with label $\hat{w}_{2l}\hat{w}_{2(l-1)} \dots \hat{w}_{21}$ from n_k to n_0. The node n_k admits a loop with label \$.

The intuition is as follows. We start with the path with label w_1 in G_0. The idea is that if $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$, then applying the constraints of the form $u \sqsubseteq v$ in Γ, we will construct a path with label w_2. The query is \$, and thus we have to check whether n_0 admits a loop with label \$ in every repair.

Intuitively, the presence of a path with label w_2 from n_0 to n_k is witnessed by a loop with label \$ at n_0. This is due to the presence of the constraints of the form $a\$\hat{a} \sqsubseteq \$$ in Γ. Indeed, suppose that $s_0 w_{21} s_1 w_{22} \dots s_l$ is a path with label w_2 from n_0 to n_k (where each s_i is a node, and thus $s_0 = n_0$ and $s_l = n_k$). Then, by induction on $0 \leq i \leq l$, using the constraints $a\$\hat{a} \sqsubseteq \$$ and the fact that n_k admits a loop with label \$, we can prove that there is an edge with label \$ from s_{l-i} to m_{l-i}. In particular, there is an edge with label \$ from s_0 to m_0. Since $s_0 = m_0 = n_0$, this implies that n_0 admits a loop with label \$.

We have to prove that $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$ iff $(n_0, n_0) \in \supseteq$-Cons(G,L,Γ). The proof of the implication from left to right follows basically from the intuition that we gave in the previous two paragraphs. For the implication from right to left, suppose that $(n_0, n_0) \in \supseteq$-Cons(G,L,Γ). We have to prove that $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$. The strategy is as follows.

(a) We construct a graph database H such that $G \subseteq H$ and $H \models \Gamma$. Moreover, if there is a path with label w_2 from n_0 to n_k in H, then $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$.

(b) Since $H_0 \models \Gamma$, there is a repair H' of G under Γ such that $G \subseteq H' \subseteq H$. As the consistent answer of $L = \$$ contains the pair (n_0, n_0), this implies that n_0 admits a loop with label \$ in H'. In particular, n_0 admits a loop with label \$ in H.

(c) We prove that if n_0 admits a loop with label \$ in H, then there is a path with label w_2 from n_0 to n_k in H. Together with (a), this finishes the proof that $w_1 \to^* w_2$.

The construction of the graph database H uses ideas from the construction in the undecidability proof of [26]. In fact, H is an extension of the graph database used in such construction. Let k_0 be the maximum of $\{k, l\}$. We define the set of nodes V' of H as:

$$\{[u] \mid u \in \Delta^*, |u| \leq k_0\} \cup \{m_i : 0 < i < l\}.$$

We identify n_0 with $[\epsilon]$. For all $1 \leq i \leq k$, we identify n_i with $[w_{11} \dots w_{1i}]$. In particular, n_k is $[w_1]$. Note that this implies that the set V of nodes of G is a subset of V'.

Let E_1' be the relation:

$$E_1' = \{([u], a, [v]) \mid v \to^*_{\mathcal{R}_0} ua\}.$$

The graph $H_0 := (V', E_1')$ is precisely the graph constructed in the undecidability proof of [26]. We now define H as the graph $(V', E_1' \cup E_2' \cup E_3')$, where E_2' and E_3' are as follows:

$$\begin{aligned} E_2' &= \{(m_i, \hat{a}, m_{(i-1)}) \mid w_{2i} = a, 1 \leq i \leq l\}, \\ E_3' &= \{([u], \$, m_{(l-i)}) \mid \text{ there is a path with label } w_{2(l-i+1)} \dots w_{2l} \\ &\qquad \text{from } [u] \text{ to } n_k \text{ in } H_0\}. \end{aligned}$$

If $i = l$, by convention, we define $w_{2(l-i+1)} \dots w_{2l}$ as the empty word ϵ. Notice that $G \subseteq H$.

We skip the details of the proof that (a), (b) and (c) hold. An important part of this proof of is to show that if there is a path with label w_2 from n_0 to n_k in H, then $w_1 \to^*_{\mathcal{R}_0} w_2$. This can be obtained as an immediate consequence of Lemma 2 in the proof of Theorem 2 in [26]. Notice that although it might be impossible to construct the graph database H (since E_1' is defined in terms of the rewrite problem for $\mathcal{R}_0$), our proof only requires the existence of such graph database. ◀

4.1 Decidable restrictions

Since the CQA problem in this context is undecidable, it is crucial to look for decidable (and, ideally, tractable) restrictions of it. We provide three such restrictions in this section: The first one is based on the class of LAV RPCs, while the second one is based on the class of GAV RPCs. The third one is obtained by modifying RPC interpretation to be *from the origin*. It is worth noticing that the restriction to classes of graph databases of bounded treewidth, which leads to tractability under the semantics of subset repairs, is not useful in this context: The undecidability result in Theorem 11 holds even over graph databases of treewidth two.

Restriction to the class of LAV RPCs. As mentioned before, in the relational scenario the CQA problem for unions of CQs under LAV tgds is tractable, no matter which repair semantics is used. We already stated a similar result for CQA over graph databases under LAV RPCs and the subset repair semantics (Theorem 8). We can further extend those techniques to obtain tractability for our CQA problem under the semantics of $\oplus$-repairs.

▶ **Theorem 12.** *For each RPQ L and finite set Γ of LAV RPCs over the same alphabet Σ, it is the case that $\oplus$-CQA(L,Γ) is in* NLogspace.

The case of the $\supseteq$-repair semantics is different: We do not know whether LAV RPCs yield decidability in this context, but we prove next that at least they do not yield tractability in data complexity. This establishes a first difference in complexity between CQA under LAV tgds in the relational context and under LAV RPCs over graph databases.

▶ **Proposition 13.** *There exist a finite alphabet Σ, a non-recursive RPQ L, and a finite set Γ of LAV RPCs (without Kleene-star) over Σ, such that $\supseteq$-CQA(L,Γ) is* coNP*-hard.*

Notice that, unlike all previous lower bounds, the one in Proposition 13 is not stated in terms of the class of word constraints. In fact, the techniques developed for studying CQA under tgds in the relational case [18] can be adapted to show that under a set Γ of LAV word constraints the problem $\supseteq$-CQA(L,Γ) is tractable.

Restriction to the class of GAV RPCs. In the case of the symmetric difference semantics, it is easy to adapt the proof of Proposition 9 in order to show that when restricting to GAV RPCs, our CQA problem is coNP-hard. Note that in the relational case, a similar result holds.

▶ **Proposition 14.**

1. *For each RPQ L and finite set Γ of GAV RPCs over the same alphabet Σ, it is the case that $\oplus$-CQA(L,Γ) is in* coNP*.*
2. *There exist a finite alphabet Σ, a non-recursive RPQ L over Σ and a single GAV RPC of the form $ab \sqsubseteq c$, where $a, b, c \in \Sigma$, such that $\oplus$-CQA(L,Γ) is* coNP*-complete.*

In the case of the superset semantics, the restriction to GAV RPCs leads to tractability. Given a graph database G and a set of GAV RPCs Γ, using a classical chase argument, we can easily compute in Logspace the *unique* superset repair of G with respect to Γ. This is identical to what happens in the setting of relational databases.

▶ **Proposition 15.** For each RPQ L and finite set Γ of GAV RPCs over the same alphabet Σ, it is the case that $\supseteq$-CQA*(L,Γ)* is in NLogspace.

Modifying the interpretation of RPCs. In the original proposal of Abiteboul and Vianu, RPQs, and therefore RPCs, are only evaluated from a particular node called the *origin*. This is motivated by their application over semi-structured data. Formally, let o be a fixed node id that we identify as the origin. A graph database $G = (V, E)$, satisfies $L_1 \sqsubseteq L_2$ *from the origin*, denoted $G \models_o L_1 \sqsubseteq L_2$, if and only if the origin o belongs to V and $\{v \in V \mid (o, v) \in L_1(G)\} \subseteq \{v \in V \mid (o, v) \in L_2(G)\}$. If Γ is a set of RPCs, we write $G \models_o \Gamma$ if $G \models_o L_1 \sqsubseteq L_2$, for each RPC $L_1 \sqsubseteq L_2$ in Γ.

We can now modify the definition of repairs and consistent answers with respect to the restricted $\models_o$ interpretation of RPCs. Assume $\star \in \{\subseteq, \supseteq, \oplus\}$. An $\{o, \star\}$*-repair* of a graph database G under a set of RPCs Γ is defined exactly as an $\star$-repair of G under Γ, except that now the satisfaction of the RPCs in Γ is defined with respect to the relation $\models_o$. (For safety reasons, we assume that G contains the origin o in this case). For instance, G' is a $\{o, \supseteq\}$-repair of G under Γ, if (1) $G \subseteq G'$, (2) $G' \models_o \Gamma$, and (3) there is no graph database G'' such that $G \subseteq G'' \subsetneq G'$ and $G'' \models_o \Gamma$.

Furthermore, if L is an RPQ and Γ is a finite set of RPCs, we define $\{o, \star\}$-CQA(L,Γ) as the problem of, given a graph database $G = (V, E)$ and a pair (u, v) of nodes in V, checking whether (u, v) is an $\{o, \star\}$*-consistent answer of L over G under* Γ, i.e., if $(u, v) \in L(G')$, for each $\{o, \star\}$-repair G' of G under Γ.

By interpreting RPCs under the relation $\models_o$, we obtain decidability for our CQA problem under the semantics of $\supseteq$-repairs. We do not know whether this can be extended to the semantics of $\oplus$-repairs. Notice the difference with the restriction to LAV RPCs we studied before: For the latter we could only obtain decidability under the $\oplus$-repairs semantics.

▶ **Theorem 16.** *For each RPQ L and finite set Γ of RPCs over the same alphabet Σ, it is the case that $\{o, \supseteq\}$-CQA(L,Γ) is in* coNP.

The difference here is that the implication problem for RPCs becomes decidable if RPCs are interpreted under the relation $\models_o$ [1]. We adapt the techniques used to prove this fact in order to obtain Theorem 16.

We do not know whether the bound in Theorem 16 is tight. Interestingly, we can show that a slight extension on the query language leads to intractability. We reduce from the problem of 3-colorability. Assume we are given an undirected graph H. From H we construct a graph database $G = (V, E)$, such that (1) V corresponds to the set of nodes of H plus the origin o, and (2) E contains edges (o, a, v), for each node v in H, and (u, e, v) and (v, e, u), for each edge $\{u, v\}$ in H. Assume that Γ consists of the single constraint $a \sqsubseteq c_1 \cup c_2 \cup c_3$. Intuitively, this tells us that a $\supseteq$-repair of G contains, for each node $v \neq o$, one, and only one, edge of the form (o, c_i, v), for $1 \leq i \leq 3$. This edge represents the color assigned to node v by an assignment of three colors to the nodes of H.

It is not hard to prove that H is 3-colorable if and only if there exists a $\supseteq$-repair G' of G such that no two nodes linked by an edge labeled e in G' are assigned the same color. This is equivalent to checking that it is not the case that there are paths labeled $c_i e$ and c_i in G', for $1 \leq i \leq 3$, that start in the origin o and reach the same node v. While this cannot be expressed as an RPQ, it can be easily expressed as an RPQ with *inverses* [14]. The query is $Q := \bigcup_{1 \leq i \leq 3} c_i e c_i^-$, where c_i^- represents a backward traversal of an edge labeled c_i. We then have that H is 3-colorable if and only if (o, o) is not an $\{o, \supseteq\}$-consistent answer of Q over G under Γ. This query can also be expressed as a *conjunction* of two RPQs [13].

▶ Remark. The restriction presented in this section does not help reducing the complexity under the semantics of $\subseteq$-repairs. In fact, it can be proved that all lower bounds obtained in Section 3 continue to hold for the semantics of $\{o, \subseteq\}$-repairs. This is done in the appendix.

5 Comparison with CQA in the relational context

We compare our results with previous results on CQA obtained in the relational context. We assume familiarity with relational schemas and CQs. Tuple-generating dependencies, or tgds, define one of the most important classes of relational database constraints. They subsume several other classes of interest, such as inclusion dependencies. In addition, they have important applications in data integration, data exchange and ontological query answering [33, 23, 10]. Formally, a tgd over a relational schema σ is a formula of the form $\forall \bar{x}(\phi(\bar{x}) \rightarrow \psi(\bar{x}))$, where both $\phi(\bar{x})$ and $\psi(\bar{x})$ are CQs over σ and each variable in $\bar{x}$ is mentioned in $\phi(\bar{x})$. A relational database D over σ satisfies this tgd if $D \models \phi(\bar{a})$ implies $D \models \psi(\bar{a})$, for each tuple $\bar{a}$ of elements in D of the same length than $\bar{x}$.

As mentioned in the introduction, each word constraint can be naturally seen as a tgd over the standard relational representation of graph databases. However, lower bounds for CQA under tgds in the relational setting, such as the ones obtained by ten Cate et al. [18], cannot be used to obtain lower bounds for CQA under word constraints (or even RPCs) in the graph database context. This is because word constraints correspond to a restricted class of tgds defined by *chain* CQs (which we call *chain tgds*). However, none of the lower bounds developed for the data complexity of CQA under tgds applies to this class.

We formalise the class of chain tgds as follows. Let σ be a relational schema that contains only binary relation symbols. A chain CQ over σ is a CQ of the form

$$\phi(x, y) \;:=\; \exists u_1 u_2 \dots u_{m-1} \big(R_1(x, u_1) \wedge R_2(u_1, u_2) \wedge \dots \wedge R_{m-1}(u_{m-1}, y)\big),$$

where each R_i is a relation symbol in σ [21]. That is, the underlying *directed* graph of a chain CQ is a path. A chain tgd is one of the form $\forall x \forall y(\phi(x,y) \rightarrow \psi(x,y))$, where both $\phi(x,y)$ and $\psi(x,y)$ are chain CQs. It is easy to see that each word constraint can be represented as a chain tgd over the standard relational representation of graph databases (in which, for each $a \in \Sigma$, there is a binary relation symbol E_a that contains all pairs of nodes that are linked by an a-labeled edge in the graph database). Conversely, each chain tgd is the representation of a word constraint.

This allows us to use our proof techniques to obtain lower bounds for CQA under the restricted class of chain tgds in the relational context:

▶ **Proposition 17.**

1. *Consider a semantics based on subset repairs of relational databases. There is a relational schema σ that contains only binary relation symbols, a finite set $\mathcal{T}$ of chain tgds and a union Q of CQs over σ, such that the problem of evaluating certain answers for Q under $\mathcal{T}$ is Π_2^{P}-hard. The same holds for the semantics based on the intersection of all subset repairs.*
2. *Consider a semantics based on superset repairs of relational databases. There is a relational schema σ that contains only binary relation symbols, a finite set $\mathcal{T}$ of chain tgds and a union Q of CQs over σ, such that evaluating certain answers for Q under $\mathcal{T}$ is undecidable. The same holds for the semantics of symmetric difference repairs.*

6 Conclusions and Future Work

In this work we initiated the study of CQA over graph databases. The data complexity of the problem is in general undecidable or highly intractable, which motivated our search for decidable, and even tractable restrictions. In the case of subset repair semantics we obtain tractability by either restricting to the class of LAV RPCs or to the class of graph databases of bounded treewidth. The class of LAV RPCs also yields tractability for our problem under the semantics of $\oplus$-repairs. On the other hand, for the semantics of superset repairs we obtain decidability if RPCs are interpreted from the origin or if we restrict to the class of GAV RPCs.

Several questions regarding CQA under the semantics of $\supseteq$- and $\oplus$-repairs remain open. For instance, we do not know if CQA under $\supseteq$-repairs is decidable when RPCs are in LAV form. Neither do we know whether CQA under $\oplus$-repairs is decidable when RPCs are interpreted from the origin. We plan to study this in the future.

It would also be interesting to look for different kinds restrictions that yield decidability for our CQA problem. For instance, in the relational scenario it is possible to obtain tractability in data complexity for CQA under $\supseteq$- and $\oplus$-repair semantics if the set Γ of tgds is *weakly acyclic* [18]. The reason is that in this case there is a polynomial that bounds the size of each repair of a database D under Γ. We would like to develop a meaningful adaptation of this notion to the scenario of RPCs in search for similar positive results. This is more difficult than in the relational case, however, since the notion of acyclicity will have to consider how regular expressions interact with each other. Another way in which positive results for our CQA problem under $\supseteq$- and $\oplus$-repair semantics could be obtained, is by restricting to classes of RPCs for which the implication problem is decidable. This includes, for instance, classes of word constraints for which the associated rewrite problem is decidable [26].

While our work concentrates on queries defined as RPQs, all upper bounds presented in the paper continue to hold in the extended scenario in which queries are defined as *conjunctive* RPQs (CRPQs) [13]. In turn, CRPQs might lead to an expressive class of *conjunctive* RPCs,

which are expressions of the form $\phi(\bar{x}) \sqsubseteq \psi(\bar{x})$, for CRPQs ϕ and ψ. It is interesting to study how the positive results presented in this paper can be extended to be applied over this class of constraints.

Acknowledgements. We are grateful to Aidan Hogan and Leonid Libkin for their helpful comments in earlier versions of the paper. Carsten Lutz and Meghyn Bienvenu also provided us with important insights on the nature of DL repairs. Barceló and Fontaine are funded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004. Fontaine is also funded by Fondecyt postdoctoral grant 3130491.

References

1 S. Abiteboul, V. Vianu. Regular path queries with constraints. *JCSS*, 58(3), pages 428–452, 1999.
2 M. Arenas, L. E. Bertossi, Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS* 1999, pages 68–79.
3 M. Arenas, W. Fan, L. Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38(3), pages 841–880, 2008.
4 F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.). The description logic handbook: Theory, implementation, and applications. *Cambridge University Press*, 2003.
5 P. Barceló. Querying graph databases. In *PODS* 2013, pages 175–188.
6 M. Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In *AAAI* 2012.
7 M. Bienvenu, R. Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *IJCAI* 2013.
8 R. Book, F. Otto String. *Rewriting Systems.* Springer Verlag, 1993.
9 P. Buneman, W. Fan, S. Weinstein. Path constraints in semi-structured databases. *J. Comput. Syst. Sci.* 61(2), pages 146–193, 2000.
10 A. Calì, G. Gottlob, M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.* (JAIR) 48, pages 115–174, 2013.
11 Andrea Calì, D. Lembo, R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS* 2003, pages 260–271.
12 D. Calvanese, G. de Giacomo, M. Lenzerini. Structured objects: Modeling and reasoning. In *DOOD* 1995, pages 229–246.
13 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR* 2000, pages 176–185.
14 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3), pages 443–465, 2002.
15 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record* 32(4), pages 83–92, 2003.
16 D. Calvanese, G. de Giacomo, M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.* 9(3), 2008.
17 D. Calvanese, M. Ortiz, M. Simkus. Containment of regular path queries under description logic constraints. In *IJCAI* 2011, pages 805–812.
18 B. ten Cate, G. Fontaine, Ph. G. Kolaitis. On the data complexity of consistent query answering. In *ICDT* 2012, pages 22–33.
19 J. Chomicki, J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.* 197(1–2), pages 90–121, 2005.

20 B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite hraphs. *Inf. Comput.* 85(1), pages 12–75, 1990.

21 G. Dong. On datalog linearization of chain queries. *Theoretical Studies in Computer Science* 1992, pages 181–206.

22 W. Fan, J. Siméon: Integrity constraints for XML. *J. Comput. Syst. Sci.*, 66(1), pages 254–291, 2003.

23 R. Fagin, Ph. G. Kolaitis, R. J. Miller, L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.* 336(1), pages 89–124, 2005.

24 G. Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? In *LICS* 2013, pages 550–559.

25 A. Fuxman, R. J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.* 73(4), pages 610–635, 2007.

26 G. Grahne, A. Thomo. Query containment and rewriting using views for regular path queries under constraints. In *PODS* 2003, pages 111–122.

27 A. Hogan, A. Harth, A. Passant, S. Decker, A. Polleres. Weaving the pedantic web. In *LDOW* 2010.

28 Ph. G. Kolaitis, Enela Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.* 112(3), pages 77–85, 2012.

29 A. Lopatenko, L. E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT* 2007, pages 179–193.

30 G. Lausen, M. Meier, M. Schmidt. SPARQLing constraints for RDF. In *EDBT* 2008, pages 499–509.

31 D. Lembo, M. Ruzzi. Consistent query answering over description logic ontologies. In *DL* 2007.

32 D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D. Fabio Savo. Inconsistency-tolerant semantics for description logics. In *RR* 2010, pages 103–117.

33 M. Lenzerini. Data integration: A theoretical perspective. In *PODS* 2002, pages 233–246.

34 Th. Lukasiewicz, M. V. Martinez, G. I. Simari. Complexity of inconsistency-tolerant query answering in Datalog+/-. In *OTM Conferences* 2013, pages 488–500.

35 R. Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In *IJCAI* 2011, pages 1057–1062.

36 K.-D. Schewe, B. Thalheim, J. W. Schmidt, I. Wetzel. Integrity enforcement in object-oriented databases. In *FMLDO* 1992, pages 174–195.

37 J. Wijsen. Condensed representation of database repairs for consistent query answering. In *ICDT* 2003, pages 375–390.

38 J. Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.* 37(2), 9, 2012.

39 P. T. Wood. Query languages for graph databases. *SIGMOD Record* 41(1), pages 50–60, 2012.

40 Y. Yuan, G. Wang, L. Chen, H. Wang. Efficient subgraph similarity search on large probabilistic graph databases. In *PVLDB* 5(9), pages 800–811, 2012.