Rewriting Guarded Existential Rules into Small Datalog Programs

Shqiponja Ahmetaj
TU Wien, Vienna, Austria

Magdalena Ortiz
TU Wien, Vienna, Austria

Mantas Šimkus
TU Wien, Vienna, Austria

Abstract

The goal of this paper is to understand the relative expressiveness of the query language in which queries are specified by a set of guarded (disjunctive) tuple-generating dependencies (TGDs) and an output (or 'answer') predicate. Our main result is to show that every such query can be translated into a (disjunctive) Datalog program, which has polynomial size if the maximal number of variables in the (disjunctive) TGDs is bounded by a constant. To overcome the challenge that Datalog has no direct means to express the existential quantification present in TGDs, we define a two-player game that characterizes the satisfaction of the dependencies, and design a Datalog query that can decide the existence of a winning strategy for the game. For guarded disjunctive TGDs, we can obtain Datalog rules with disjunction in the heads. However, the use of disjunction is limited, and the resulting rules fall into a fragment that can be evaluated in deterministic single exponential time. We proceed quite differently for the case when the TGDs are not disjunctive and we show that we can obtain a plain Datalog query. Notably, unlike previous translations for related fragments, our translation requires only *polynomial time* under the reasonable assumption that the maximal number of variables in the (disjunctive) TGDs is bounded by a constant.

2012 ACM Subject Classification Theory of computation → Database query languages (principles), Theory of computation → Logic and databases, Theory of computation → Incomplete, inconsistent, and uncertain databases

Keywords and phrases Existential rules, Expressiveness, Query Rewriting

Digital Object Identifier 10.4230/LIPIcs.ICDT.2018.4

Acknowledgements This work was supported by the Austrian Science Fund (FWF) projects P30360, P30873, and W1255.

1 Introduction

This paper contributes to the understanding of the relative expressiveness and succinctness of *tuple-generating dependencies (TGDs)* [6] and their extension with disjunction *(DTGDs)* [11] as query languages for possibly incomplete data. TGDs and DTGDs are families of existential rules that play a crucial role as constraint languages for databases, especially in areas like data exchange and data integration. In recent years, they have also gained popularity in knowledge representation, where they are used as languages for writing *ontologies*, which can enrich possibly incomplete extensional data with intensional domain knowledge that can be leveraged at query time to obtain more complete and accurate answers.

When (D)TGDs are viewed as a query language for incomplete data, a *query* takes the form $Q = (\Sigma, q)$, where Σ is a set of DTGDs and q is a predicate. Q can be seen as a close

relative of the ontology-mediated queries (OMQs) considered in the literature on *Description Logics (DLs)* [25, 26, 1]. Given a database instance D, such a query asks to retrieve all tuples of values that are present in the relation q, over all databases that contain D and satisfy the dependencies in Σ. The corresponding query answering problem is only decidable if suitable restrictions are imposed on Σ. *Guardedness* is one of the best known such restrictions. Inspired by modal and first order logic [2], it yields a computationally robust family of TGDs whose complexity is quite well understood [9].

The problem that we study is the *rewritability* into Datalog. That is, given such a query $Q = (\Sigma, q)$, we want to build from it a Datalog query (P_Q, q_Q) that has the same certain answers over every input database instance over the signature of Σ. From the theoretical perspective, this problem is important: the existence of such a rewriting in the general case, and its size, allow us to understand the relative expressiveness and succinctness of guarded (D)TGDs as a query language. It also has practical relevance. Rewriting into standard query languages is considered one the most promising approaches to achieve scalability of expressive query languages, by reusing existing optimized database technologies. In the field of *ontology mediated querying*, for instance, rewritings have been a major research line for most of the last decade. Even 'impracticable' rewritings developed only for theoretical purposes can lay the groundwork for practical rewriting techniques, and shed light on their limits.

Our central result is a novel translation of such queries into 'small' Datalog programs. Our translation is non-trivial because guarded DTGDs allow for existential quantification in rule heads, while disjunctive Datalog (Datalog$^\vee$) does not support it. To overcome this challenge, the paper employs a game-theoretic characterization of answers to guarded DTGD queries. In particular, we tie the inclusion of a tuple in a query answer to the existence of a winning strategy in a two-player game. The Datalog$^\vee$ query that results from the translation aims to compute the answer to the input query by actually checking the existence of such winning strategies.

The translation takes polynomial time in the size of Q, if we assume that the guarded DTGDs only use a bounded number of variables. In particular, since our dependencies are guarded, this applies to theories where the arities of predicates are bounded by a constant, and no arbitrary conjunctions of atoms occur in the consequent of the dependencies. We stress that this case, although apparently restrictive, is very relevant, since in practice the arities of predicates are often not high, especially in settings that are tailored to deal with incomplete information (e.g., standard DLs use at most binary predicate symbols). If the number of variables per rule are not bounded, the translation is exponential in the size of Q, which is unavoidable under common assumptions in complexity theory (see Section 6). From the translation and the combined complexity of Datalog$^\vee$ queries, we can easily infer a coNExpTime upper bound for answering guarded DTGD queries under the assumption of bounded number of variables. However, we can do better: by analyzing the shape of the Datalog$^\vee$ queries resulting from our translation, we conclude that the program can be evaluated in deterministic exponential time in its size, which yields a new worst-case optimal algorithm for answering guarded DTGD queries with bounded number of variables.

Finally, we show that when the dependencies in the query are expressed using non-disjunctive guarded TGDs, we can provide a data-independent translation into plain Datalog queries. Again, the translation takes only polynomial time, assuming bounded number of variables. We note that translations into plain (resp., disjunctive) Datalog from guarded TGDs (resp., guarded DTGDs) do exist in the literature (see Related Work), but their techniques are quite different from ours, and more importantly, they are all exponential even under bounded number of variables.

Related Work. In the database community, there is a considerable amount of works on query rewritings for variants of guarded TGDs. For instance, it is known that when q is a (union of) conjunctive queries (rather than a predicate), (Σ, q) can be rewritten into a plain Datalog program, for Σ a set of guarded TGDs or more general classes of dependencies [3, 18]. For guarded *disjunctive* TGDs and q a union of conjunctive queries, a disjunctive Datalog rewriting can be found in [7]. The rewritings in [3, 18, 7] take *exponential time*, even if the number of variables in each TGD is bounded by a constant. In [16], the authors provide a translation into non-recursive Datalog for guarded TGDs, which is polynomial if the size of the schema of the input theory is bounded, which is a significantly stronger restriction that bounding the arity only. Moreover, this rewriting adopts a so-called *combined approach* that modifies the data incorporating inferences from the TGDs. The same authors proposed in [17] a rewriting into non-recursive Datalog queries of polynomial size for linear existential rules, without bounding the schema, or the arity. We remark that, for the case of bounded arity, this result follows from [19, 14]. The polynomial rewritings into non-recursive Datalog in [19, 17, 14] assume that the data contains some fixed number of constants. Our rewritings also use a few constants, which are supported by the Datalog variants we employ as target query languages.

In the context of DLs, rewritability into traditional query languages of queries enriched with an ontology is considered one of the most central questions. For instance, the *DL-Lite* family of DLs is popular because they often support rewritability into *first-order (FO) queries* [10]. In [23], the authors introduced the *combined approach* as a means to obtain rewritings into FO queries for more expressive languages like $\mathcal{EL}$. The authors of [19] proposed a rewriting into non-recursive Datalog queries of polynomial size for the prominent *DL-Lite*$_{\mathcal{R}}$. The succinctness problem of this and related FO rewritings has been thoroughly investigated in [14]. For more expressive DLs and *(unions of) conjunctive queries (UCQs)*, which are often not FO-rewritable, there is a considerable amount of work on rewritings into Datalog queries. The authors in [21] first showed that instance queries in an expressive DL with disjunction can be rewritten into in a disjunctive Datalog a program of exponential size. For variants of expressive DLs with disjunction and (U)CQs the existence of exponential rewritings into disjunctive Datalog is known [7, 22]. The authors of [27] propose a polynomial time Datalog translation of instance queries for an expressive *Horn-DL* without disjunction. Some of the above rewritings lie at the core of implemented systems, e.g., [28, 13, 29].

This paper is inspired by the technique in [1], where it was shown that instance queries mediated by ontologies in the expressive Description Logic $\mathcal{ALCHIO}$ can be rewritten in polynomial time into a Datalog$^{\vee}$ program. Guarded DTGDs can be seen as an ontology language that is orthogonal to $\mathcal{ALCHIO}$, which only supports unary and binary predicates and has a rather restricted syntax, yet it features *nominals* (essentially, constants). The guarded DTGDs considered here allow for predicates of arbitrary arities, but we disallow constants. In general, the higher arities and the rather relaxed syntax of guarded DTGDs makes the adaptation of the results in [1] highly non-trivial.

2 Preliminaries

General Technical Definitions. Let Δ_c, Δ_n and Δ_v be infinite mutually disjoint sets of *constants*, *labeled nulls*, and *variables*, respectively. Elements in $\Delta_c \cup \Delta_n \cup \Delta_v$ are *terms*. If no confusion arises, we may abuse notation and write a tuple of terms in the place of the set of its elements. An *atom* α is an expression of the form $R(t_1, \ldots, t_n)$, where R is a

predicate name with *arity* n, and $t_1, \ldots, t_n$ are terms. We let $\mathsf{terms}(\alpha) = \{t_1, \ldots, t_n\}$ and $\mathsf{vars}(\alpha) = \mathsf{terms}(\alpha) \cap \Delta_v$. If $\mathsf{terms}(\alpha) \subseteq \Delta_c$, then α is *ground*. For a set Γ of atoms, we let $\mathsf{terms}(\Gamma) = \bigcup_{\alpha \in \Gamma} \mathsf{terms}(\alpha)$ and $\mathsf{vars}(\Gamma) = \mathsf{terms}(\Gamma) \cap \Delta_v$. An instance I is a (possibly infinite) set of atoms with terms from $\Delta_n \cup \Delta_c$. A *database* D is a finite instance with only ground atoms. We denote with $\mathsf{dom}(I)$ the set of terms that occur in I. A *substitution* is a partial function h from a set of symbols S to a set of symbols S'. We let $h(\vec{t}) = (h(t_1), \ldots, h(t_n))$ for a tuple $\vec{t}$, $h(R(\vec{t})) = R(h(\vec{t}))$ for an atom $R(\vec{t})$, and $h(\Gamma) = \{h(R_1(\vec{t}), \ldots, h(R_n(\vec{t}))\}$ for a set of atoms $\Gamma = \{R_1(\vec{t}), \ldots, R_n(\vec{t})\}$.

Disjunctive Tuple-Generating Dependencies. A *disjunctive tuple-generating dependency (DTGD)* (a.k.a. *disjunctive existential rule*) σ is an expression of the form

$$\forall \vec{x}(\varphi(\vec{x}) \rightarrow \bigvee_{i=1}^{n} \exists \vec{y}_i.\psi_i(\vec{x}, \vec{y}_i)), \tag{1}$$

where $n \geq 1$, $\vec{x}, \vec{y}_1, \ldots, \vec{y}_n \subseteq \Delta_v$, and $\varphi, \psi_1 \ldots \psi_n$, are conjunctions of atoms with terms from Δ_v only. If $n = 1$, then σ is simply called a *tuple-generating dependency (TGD)* (a.k.a. *existential rule*). For simplicity, we use a comma for conjoining atoms and we omit the universal quantifiers in front of DTGDs. An instance I *satisfies* σ, denoted $I \models \sigma$, if for every substitution h from the variables in $\vec{x}$ to $\mathsf{dom}(I)$ such that $h(\varphi) \subseteq I$ there exists an $i \in [n]$ and a substitution h' from the variables in $\mathsf{vars}(\psi_i)$ to $\mathsf{dom}(I)$ such that $h'(\psi_i) \in I$ and $h'(x) = h(x)$ for all $x \in \vec{x}$. We say σ is *guarded* if there exists an atom α in φ, called a *guard*, such that $\vec{x} \subseteq \mathsf{vars}(\alpha)$. We refer the reader to [9] and [8] for more details on guarded (D)TGDs.

A set Σ of (D)TGDs is called a *theory*. The *schema* of Σ, denoted by $\mathsf{sch}(\Sigma)$, is the set of predicate names that appear in Σ, and for a set of atoms Γ, $\mathsf{sch}(\Gamma)$, is the set of all predicates that occur in Γ. A theory is *guarded* if all its (D)TGDs are guarded. An instance I *satisfies* a theory Σ, written $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$. For theories Σ, Σ', we say Σ entails Σ', written $\Sigma \models \Sigma'$, if for all instances I, $I \models \Sigma$ implies $I \models \Sigma'$.

For a database D, we write $I \models D$ if $D \subseteq I$. Given a database D and a theory Σ, a *model* of (Σ, D) is an instance I, denoted $I \models (\Sigma, D)$, such that $I \models D$ and $I \models \Sigma$. Given a ground atom α, we say α is *entailed* by Σ and D, written $(\Sigma, D) \models \alpha$, if $\alpha \in I$ for every model I of (Σ, D).

Queries. A *query* is a pair (Σ, q), where Σ is a theory and q is a predicate symbol. Given a query (Σ, q) and a database D, we let

$$\mathsf{ans}(\Sigma, q, D) = \{\vec{c} \in (\Delta_c)^n \mid (\Sigma, D) \models q(\vec{c})\},$$

where n is the arity of q. We call $\mathsf{ans}(\Sigma, q, D)$ the *(certain) answer* to (Σ, q) over D.

Normal Form. To simplify some technical constructions, we focus on guarded DTGDs with a restricted syntactic structure:

▶ **Definition 1.** (Normalized DTGDs) A theory Σ of DTGDs is in *normal form* if each $\sigma \in \Sigma$ is of one of the following forms:

$$B \rightarrow \exists \vec{y}.H \qquad \text{where } B, H \text{ are atoms with terms from } \Delta_v \tag{2}$$

$$\varphi \rightarrow \bigvee_{i=1}^{n} H_i \qquad \text{where each } H_i \text{ is an atom and } \varphi \text{ is a set of atoms over terms in } \Delta_v \tag{3}$$

In other words, a normalized theory Σ of DTGDs consists of a set of TGDs with one atom in the body and one atom in the head, and a set of guarded DTGDs without existentially quantified variables. By means of fresh predicate names, a theory of guarded DTGDs can be converted into the above normal form while preserving atom entailment (see, e.g., [18]). We state this more precisely next.

▶ **Proposition 2.** *Every query (Σ, q) can be transformed in polynomial time into a query (Σ', q) such that*

(a) $\mathsf{ans}(\Sigma, q, D) = \mathsf{ans}(\Sigma', q, D)$ *for every database D over* $\mathsf{sch}(\Sigma)$*;*

(b) Σ' *is in normal form;*

(c) *if Σ is disjunctive guarded, then Σ' is disjunctive guarded;*

(d) *if Σ has only guarded (non-disjunctive) TGDs, then Σ' also has only TGDs.*

Proof sketch. Assume a query (Σ, q). The idea is to replace every DTGD of the form (1) that appears in Σ by

- the DTGD $\forall\vec{x}(\varphi(\vec{x}) \rightarrow \bigvee_{i=1}^{n} F_i(\vec{x}_i))$, were $\vec{x}_i$ is a list of variables from $\vec{x}$ that appear in $\psi_i(\vec{x}, \vec{y}_i)$, and each F_i is a fresh predicate symbol with arity $|\vec{x}_i|$,
- for all $1 \leq i \leq n$, the TGD $\forall\vec{x}_i(F_i(\vec{x}_i) \rightarrow \exists\vec{y}_i.F_i'(\vec{x}_i, \vec{y}_i))$, where F_i' is a fresh predicate name of arity $|\vec{x}_i| + |\vec{y}_i|$, and
- for all $1 \leq i \leq n$ and each atom H in $\psi_i(\vec{x}, \vec{y}_i)$, the TGD $\forall\vec{x}_i\forall\vec{y}_i(F_i'(\vec{x}_i, \vec{y}_i) \rightarrow H)$.

The points *(b)*, *(c)*, and *(d)* follow easily. The point *(a)* holds because (Σ', D) is a *conservative extension* of (Σ, D) for any database D over $\mathsf{sch}(\Sigma)$. In particular, (i) any model of (Σ', D) is a model of (Σ, D), and (ii) every model of (Σ, D) that is over $\mathsf{sch}(\Sigma)$ can be extended to a model of (Σ', D) by properly populating the predicates F_i, F_i' introduced during the normalization process. ◀

In the rest of the paper, we consider only normalized guarded theories Σ. We let $\Sigma_\exists$ and $\Sigma_\forall$ denote the guarded DTGDs of the form (2) and of the form (3) that appear in Σ, respectively. The *width* of Σ, written *width*(Σ), is the maximal arity over the predicate in $\mathsf{sch}(\Sigma)$.

In the discussion below, and in particular in the complexity upper bounds, we often refer to (normalized) theories with *bounded width* (or *bounded predicate arity*), in which *width*(Σ) is bounded by a constant. We remark that the normalization process increases the predicate arities (the F_i and F_i' may exceed the width of the original theory). The width of a normalized theory, however, is bounded whenever there is only a bounded number of variables in each DTGD. It is also bounded if the original theory (before normalization) has bounded width, and in the consequent of DTGDs, $\psi_i(\vec{x}, \vec{y}_i)$ is a single atom rather than a conjunction of atoms. Therefore, the results below that refer to normalized theories of bounded width apply to these relevant cases.

Datalog with Disjunction (Datalog$^\vee$). A *rule* ρ is an expression of the form $H_1 \vee \ldots \vee H_n \leftarrow B_1, \ldots, B_k$, where $H_1, \ldots, H_n, B_1, \ldots, B_k$ are atoms with terms from $\Delta_v \cup \Delta_c$. The atoms $H_1, \ldots, H_n$ are called *head* atoms, and $B_1, \ldots, B_k$ are called *body* atoms. We require that each variable that appears in ρ also occurs in a body atom. A rule with no body atoms of the form $H \leftarrow$ is called a *fact*. A rule ρ with no head atoms of the form $\leftarrow B_1, \ldots, B_k$ is a *constraint*. A finite set P of rules is called a *program*. If every rule in a program P has at most one head atom, then P is called a (plain) Datalog program. The *grounding* $\mathsf{ground}(P)$ of a program P is the ground (i.e., variable-free) program that is obtained from P by replacing each rule ρ of P by its ground instances, i.e., rules that can be obtained from ρ by substituting its variables with constants of P.

A database D is a *model* of a program P, if $\{B_1, \ldots, B_k\} \subseteq D$ implies $D \cap \{H_1, \ldots, H_n\} \neq \emptyset$ for all rules $H_1 \vee \ldots \vee H_n \leftarrow B_1, \ldots, B_k$ in $\mathsf{ground}(P)$. A DATALOG$^\vee$ *query* is a pair (P, q), where P is a program, and q is a predicate symbol from P. We let $\mathsf{ans}(P, q, D)$ denote the set of all n-tuples $\vec{c}$ of values from Δ_c, where n is the arity of q, such that $q(\vec{c}) \in D'$ for all models D' of $P \cup \{\alpha \leftarrow \mid \alpha \in D\}$. If P is a plain DATALOG program, then (P, q) is a plain DATALOG query.

3 Counter Models

Assume Σ is a guarded theory of DTGDs. We want to decide whether $(\Sigma, D) \not\models \alpha$ for a given database D and a ground atom α. That is, we want to decide the existence of a model I of Σ and D such that $\alpha \notin I$. Rather than aiming at constructing such a (possibly infinite) I, we proceed as follows:

(1) We search for a 'small' part of such a possible I, which we call a *core instance* D_c for (Σ, D). Intuitively, core instances are databases that extend D to ensure the satisfaction of $\Sigma_\forall$ (but no nulls are added, and the satisfaction of $\Sigma_\exists$ is not guaranteed). They fix how the constants of D participate in all predicates, while ensuring that α is false.

(2) For each candidate core instance D_c, we verify if it can be extended to also satisfy $\Sigma_\exists$, while preserving satisfaction of $\Sigma_\forall$. When extending D_c, entailment of ground atoms is preserved, and hence so are the satisfaction of D and the non-entailment of α.

Core instances and their extensions are defined next:

▶ **Definition 3.** (Core instances) A *core instance* for (Σ, D) is a database D_c with predicates from $\mathsf{sch}(\Sigma)$ such that:

(c1) $\mathsf{dom}(D) = \mathsf{dom}(D_c)$, and

(c2) $D_c \models (\Sigma_\forall, D)$

An instance I is called an *extension* of D_c, if D_c is the result of restricting I to $\mathsf{dom}(D)$.

A core and its extensions coincide on the ground atoms they entail over $\mathsf{dom}(D)$. Hence, for a given query (Σ, q) and a tuple $\vec{c}$, deciding that $\vec{c} \notin \mathsf{ans}(\Sigma, q, D)$ amounts to deciding whether there is a core instance that does not entail $q(\vec{c})$, and that can be extended into a model of (Σ, D). Defining a disjunctive DATALOG program whose models are the core instances described above is not hard. The second part, that is, verifying if a core can be extended to a full model, is more challenging. In fact, it corresponds to testing satisfiability of a database (in this case, a candidate D_c) w.r.t. a theory Σ of guarded DTGDs, which is known to be EXPTIME-complete when predicate arities are bounded, and 2EXPTIME-complete otherwise [20, 15].

In order to obtain a set of rules that solves this problem (and that has polynomial size if *width*(Σ) is bounded), we characterize it as a game, revealing a simple algorithm that admits an elegant implementation in DATALOG$^\vee$. The game relies on *types* which we define next.

▶ **Definition 4.** (Types) For a theory Σ, we assume an order over some special variables $\mathbf{x}_1, \ldots, \mathbf{x}_w$, such that $w = \mathit{width}(\Sigma)$. A *type* τ over a theory Σ is a set of atoms over $\mathsf{sch}(\Sigma)$ such that $\mathsf{terms}(\tau) \subseteq \{\mathbf{x}_1, \ldots, \mathbf{x}_w\}$. We denote by *types*$(\Sigma)$ the set of all types over Σ.

Given an instance I and a tuple $\vec{c} = (c_1, \ldots, c_k)$ with $k \leq w$ and $\vec{c} \subseteq \mathsf{dom}(I)$, we let the type $\mathit{type}(\vec{c}, I) = \{R(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_j}) \mid R(c_{i_1}, \ldots, c_{i_j}) \in I, \{c_{i_1}, \ldots, c_{i_j}\} \subseteq \vec{c}\}$. A type τ is *realized* in I if there is some tuple $\vec{c} \subseteq \mathsf{dom}(I)$ such that $\mathit{type}(\vec{c}, I) = \tau$.

For a given Σ, the number of possible types that one can construct is bounded by $2^{n(w^w)}$, where n is the number of predicates occurring in $\mathsf{sch}(\Sigma)$, and w is the *width*(Σ); if w is

bounded, there are only exponentially many types. In the rest of the paper, we write $\mathbf{X}$ to denote the set of special variables $\{\mathbf{x}_1, \ldots, \mathbf{x}_w\}$ and $\vec{\mathbf{x}}$ to denote the tuple $(\mathbf{x}_1, \ldots, \mathbf{x}_w)$.

▶ **Definition 5.** Let $\tau \in \mathit{types}(\Sigma)$ be a type over the special variables $\mathbf{X}$ and let $\mathbf{Y} \subseteq \mathbf{X}$. We denote by $\tau|_{\mathbf{Y}} \subseteq \tau$ the type that contains each atom $R(\vec{t}) \in \tau$ with $\vec{t} \subseteq \mathbf{Y}$.

We now describe a game to decide whether a given core D_c can be extended into a model of (Σ, D). The game is played by Bob (the builder), who wants to extend D_c into a model, and Sam (the spoiler), who wants to spoil all of Bob's attempts. Sam starts by picking a tuple $\vec{c}$ such that $R(\vec{c}) \in D_c$. They look at its type $\mathit{type}(\vec{c}, D_c)$ and if it does not satisfy the DTGDs in $\Sigma_\forall$, Sam is declared the winner. Otherwise, in each turn Sam chooses a TGD $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ and a substitution h such that $h(B)$ is satisfied by the current type, forcing Bob to pick a type that satisfies H and that coincides with the current type on the shared variables of B and H. The game continues for as long as Bob can respond to the challenges of Sam.

Formally, for a theory Σ with $\Sigma = \Sigma_\exists \cup \Sigma_\forall$ and an instance I, we define the set $\mathsf{LC}(\Sigma, I)$ of *locally consistent types* as the set that contains each type $\tau \in \mathit{types}(\Sigma)$ such that the following condition holds.

(LC$_\forall$) For all DTGDs $\sigma \in \Sigma_\forall$ of the form $\varphi \to \bigvee_{i=1}^{n} H_i$, and for all substitutions h from $\mathsf{vars}(\varphi)$ to $\mathbf{X}$, $h(\varphi) \subseteq \tau$ implies that there exists $i \in [n]$ such that $h(H_i) \in \tau$.

The game on an instance I and a theory Σ starts by Sam choosing a tuple $\vec{c}$ such that $R(\vec{c}) \in I$ and $\tau = \mathit{type}(\vec{c}, I)$ is set to be the *current type*. Then:

(♦) If $\tau \notin \mathsf{LC}(\Sigma, I)$ then Sam is declared winner.
Otherwise, Sam chooses a TGD $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ and a substitution $h : \mathsf{vars}(B) \to \mathbf{X}$ such that $h(B) \in \tau$; if there is no such TGD in Σ, Bob is declared the winner. Otherwise, let $\mathbf{Z} = \mathsf{vars}(B) \cap \mathsf{vars}(H)$. Bob has to choose a type τ' such that the following conditions hold:

(C1) $\tau|_{h(\mathbf{Z})} = \tau'|_{h(\mathbf{Z})}$,
(C2) there exists a substitution $h' : \mathsf{vars}(H) \to \mathbf{X}$ with $h(\mathbf{Z}) = h'(\mathbf{Z})$ such that $h'(H) \in \tau'$.

The type τ' is set to be the current type, and the game continues with a new round, i.e. we go back to ♦.

A *run* of the game on an instance I is a (possibly infinite) sequence $\vec{c}\sigma_1 h_1 \tau_1 \sigma_2 h_2 \tau_2 \ldots$ where $\vec{c}$ is a tuple initially picked by Sam such that some atom $R(\vec{c}) \in I$, and each σ_i, h_i and τ_i are the TGD and the substitution picked by Sam and the type picked by Bob in round i, respectively. A *strategy for Bob*, to play on I and Σ, is a partial function *str* that maps a type τ, a TGD $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$, and a substitution $h : \mathsf{vars}(B) \to \mathbf{X}$ with $h(B) \in \tau$ to a type τ' that satisfies (C1) and (C2); intuitively, the strategy gives a move for Bob in response to the moves of Sam. A run $\vec{c}\sigma_1 h_1 \tau_1 \sigma_2 h_2 \tau_2 \ldots$ with $\mathit{type}(\vec{c}, I) = \tau_0$ *follows a strategy str* if $\tau_i = \mathit{str}(\tau_{i-1}, \sigma_i, h_i)$ for every $i \geq 1$.

For a finite run r, we let $\mathit{tail}(r) = \mathit{type}(\vec{c}, I)$ if $r = \vec{c}$, and $\mathit{tail}(r) = \tau_\ell$ if $r = \vec{c} \ldots \sigma_\ell h_\ell \tau_\ell$ with $\ell \geq 1$. The strategy *str* is called *non-losing* on I if for every finite run r that follows *str*:

(i) $\mathit{tail}(r) \in \mathsf{LC}(\Sigma, I)$, and
(ii) $\mathit{str}(\mathit{tail}(r), \sigma, h_\sigma)$ is defined for every $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ and every substitution $h_\sigma : \mathsf{vars}(B) \to \mathbf{X}$ with $h_\sigma(B) \in \mathit{tail}(r)$.

The correctness of the game is shown in the following theorem.

▶ **Theorem 6.** *Let Σ be a theory of guarded DTGDs, D a database, and α a ground atom. Then $(\Sigma, D) \not\models \alpha$ iff there is a core instance D_c for (Σ, D) such that:*
(1) *$\alpha \notin D_c$, and*
(2) *there is a non-losing strategy for Bob on D_c.*

Proof sketch. We focus on showing that for any given core D_c, there is a non-losing strategy *str* for Bob on D_c if and only if D_c can be extended into an instance I that satisfies (Σ, D).

For "$\Leftarrow$", assume an arbitrary model I of (Σ, D) that is an extension of D_c. We extract from I a non-losing strategy *str* for Bob as follows. First, let $rlz(I)$ be the set of all types realized in I. Observe that $rlz(I) \subseteq \mathsf{LC}(\Sigma, D_c)$ holds since the core D_c must satisfy all the rules in $\Sigma_\forall$. It suffices to set, for each type $\tau \in rlz(I)$ and each $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ with $h(B) \in \tau$, $str(\tau, \sigma, h) = \tau'$ for an arbitrarily chosen type $\tau' \in rlz(I)$ that satisfies (C1) and (C2) which exists because I is a model, thus it satisfies all the DTGDs in Σ.

For the "$\Rightarrow$" direction, from an arbitrary non-losing *str* for D_c, we build I as follows. We write $\vec{c}_\tau$ to denote a tuple of constants realizing τ; $rn(D_c, str)$ to denote the set of all finite runs $\vec{c}_\tau \sigma_1 h_1 \tau_1 \sigma_2 h_2 \tau_2 \dots$ that follow *str*; and $fv(\sigma)$ to denote the set $\mathsf{vars}(B) \cap \mathsf{vars}(H)$ of variables where $\sigma \in \Sigma$ is a TGD of the form $B \to \exists \vec{y}.H$. We let I be the set of atoms $R(t_1, \dots, t_\ell)$ such that one of the following holds:
(a) there exists a run $r = \vec{c}_\tau$ in $rn(D_c, str)$ with $R(\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_\ell}) \in \tau$ and $t_j = c_{i_j} \in \vec{c_\tau}$ for each $j \in [1, \ell]$ or
(b) there exists a run $r = \vec{c}_\tau \dots \sigma_i h_i \tau_i$ in $rn(D_c, str)$, where $i \geq 1$ and σ_i is of the form $B \to \exists \vec{y}.H$ such that $R(\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_\ell}) \in \tau_i$ and for each t_j with $j \in [1, \ell]$ the following holds:

(1) t_j is the labelled null $r'\mathbf{x}_{i_j}$, if
  (i) $r' = \vec{c}_\tau \dots \sigma_k h_k \tau_k$ and $r = r' \dots \sigma_i h_i \tau_i$ with $1 \leq k \leq i$,
  (ii) $\mathbf{x}_{i_j} \notin h_k(fv(\sigma_k))$, and
  (iii) $\mathbf{x}_{i_j} \in h_l(fv(\sigma_l))$ for each $k < l \leq i$ if $i \neq k$.
(2) t_j is the constant c_{i_j} if $\mathbf{x}_{i_j} \in h_l(fv(\sigma_l))$ for each $l \in [1, i]$.

This I satisfies (Σ, D) and it is an extension of D_c. For more details, see Appendix A. ◀

To decide whether Bob has a non-losing strategy on a given core, we use the type elimination procedure **Mark** presented in Algorithm 1. Intuitively, the algorithm *marks* all types from which Bob does not have a non-losing strategy. It takes as input a theory Σ, and an instance I which intuitively is the core being checked. The algorithm builds the set of all types, and then it starts marking the types that are not a winning choice for Bob. First, in step ($\mathrm{M}_\forall$), the algorithm marks the types that are not in $\mathsf{LC}(\Sigma, I)$. Then it iterates using ($\mathrm{M}_\exists$) to exhaustively mark types τ that allow Sam to pick a TGD for which Bob cannot reply with any type τ'.

The formal relationship between the game and the marking algorithm is established next.

▶ **Theorem 7.** *Let D_c be a core instance for (Σ, D). Then Bob has a non-losing strategy on D_c iff none of the types realized in D_c is marked by* **Mark**(Σ).

Proof sketch. For the "$\Rightarrow$" direction, we can show that if a type τ is marked, then it cannot occur in a non-losing *str* for Bob. The proof is by induction in the number of iterations that the algorithm **Mark**(Σ) requires to mark τ. For the "$\Leftarrow$" direction, a non-losing *str* for D_c is obtained by first taking all unmarked types $\tau \in types(\Sigma)$ (which are all contained in $\mathsf{LC}(\Sigma, I)$,

Algorithm 1: Mark.

input : theory Σ of guarded DTGDs

output : Set of (possibly) marked types

$N \leftarrow \{\tau \mid \tau \in \mathit{types}(\Sigma)\}$

($\mathrm{M}_\forall$) Mark each $\tau \in N$ such that there exists:
- $\sigma \in \Sigma_\forall$ of the form $\varphi \rightarrow \bigvee_{i=1}^{n} H_i$, and
- h from $\mathsf{vars}(\varphi)$ to $\mathbf{X}$, s.t. $h(\varphi) \subseteq \tau$, and $h(H_i) \notin \tau$ for each $i \in [n]$.

repeat

($\mathrm{M}_\exists$) Mark $\tau \in N$ if there exists a TGD $B \rightarrow \exists \vec{y}.H \in \Sigma_\exists$ and a substitution $h : \mathsf{vars}(B) \rightarrow \mathbf{X}$ with $h(B) \in \tau$ s.t. for each $\tau' \in N$, at least one of the following holds:

($\mathrm{C0}$) τ' is marked,
($\mathrm{C1}'$) $\tau|_{h(\mathbf{Z})} \neq \tau'|_{h(\mathbf{Z})}$, or
($\mathrm{C2}'$) $h'(H) \notin \tau'$ for each $h' : \mathsf{vars}(H) \rightarrow \mathbf{X}$ with $h(\mathbf{Z}) = h'(\mathbf{Z})$

where $\mathbf{Z} = \mathsf{vars}(B) \cap \mathsf{vars}(H)$.

until *no new type is marked*

return N

as otherwise they would be marked by ($\mathrm{M}_\forall$). Then, for each unmarked type τ, each TGD $\sigma \in \Sigma_\exists$ of the form $B \rightarrow \exists \vec{y}.H$ and each substitution $h : \mathsf{vars}(B) \rightarrow \mathbf{X}$ with $h(B) \in \tau$, we set $\mathit{str}(\tau, \sigma, h) = \tau'$ for an arbitrarily chosen unmarked type τ' that satisfies (C1) and (C2). ◀

4 Translation into Datalog$^\vee$

The goal of this section is to build, from a given query (Σ, q), a DATALOG$^\vee$ program P such that (Σ, q) and (P, q) have the same answers for all databases D over $\mathsf{sch}(\Sigma)$. Moreover, the size of P is polynomial in Σ whenever $\mathit{width}(\Sigma)$ is bounded. P has three major components:

(a) rules that non-deterministically generate a core instance D_c for (Σ, D), where D is an input database;

(b) rules that implement the marking algorithm presented in the previous section;

(c) rules that 'glue' (a) and (b), by ensuring that all types that occur in D_c are not marked.

We emphasize that the construction of P depends exclusively on (Σ, q), and is independent from a particular database D. In what follows, we let $w = \mathit{width}(\Sigma)$, and let n be the number of distinct predicate symbols occurring in $\mathsf{sch}(\Sigma)$.

(I) Collecting the constants. We first add rules to collect in the unary predicate const all the constants that occur in the input database. For each $R \in \mathsf{sch}(\Sigma)$ with arity ℓ, and for each $1 \leq i \leq \ell$ we add the rule:

$$\mathsf{const}(u_i) \leftarrow R(u_1, \ldots, u_\ell)$$

For each $1 < i \leq w$ we have an i-ary version of const that stores i-ary tuples of constants:

$$\mathsf{const}^i(u_1, \ldots, u_i) \leftarrow \mathsf{const}(u_1), \ldots, \mathsf{const}(u_i)$$

(II) Generating core instances. We use a fresh predicate $\overline{R}$ for each $R \in \mathsf{sch}(\Sigma)$, and add the following rules to P for each ℓ-ary predicate $R \in \mathsf{sch}(\Sigma)$, where $\vec{u}$ is an ℓ-ary tuple of variables:

$$R(\vec{u}) \vee \overline{R}(\vec{u}) \leftarrow \mathsf{const}^{\ell}(\vec{u}) \qquad\qquad \leftarrow R(\vec{u}), \overline{R}(\vec{u})$$

To ensure (c2) in Definition 3, for each $\sigma \in \Sigma_{\forall}$ of the form $\varphi \rightarrow \bigvee_{i=1}^{n} H_i$ we add the rule $\bigvee_{i=1}^{n} H_i \leftarrow \varphi$. For any input D, the models of the above rules (when restricted to the predicates in $\mathsf{sch}(\Sigma)$) are the core instances D_c of (Σ, D).

Towards checking whether D_c can be extended to satisfy all rules in Σ, we implement the algorithm **Mark** from Section 3. We assume a fixed, arbitrary enumeration $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_m$ of all the atoms that can be constructed over the predicate symbols in $\mathsf{sch}(\Sigma)$ and the special variables in $\mathbf{X}$. The set of all these atoms $\{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_m\}$ is denoted by $\mathbf{A}$. Note that $1 \leq m \leq n(w^w)$, hence if w is bounded by a constant, then $|\mathbf{A}|$ is polynomially bounded.

We assume also a pair $0, 1$ of special constants. Intuitively, we use an m-ary predicate symbol $\mathsf{Type} = \{0,1\}^m$ to store all types in $\mathit{types}(\Sigma)$. For instance, a tuple $(r_1, \ldots, r_m) \in \mathsf{Type}$, encodes the type $\tau = \{\boldsymbol{\alpha}_i \mid r_i = 1, 1 \leq i \leq m\}$. This relation is the basis to compute another m-ary relation $\mathsf{Marked} \subseteq \{0,1\}^m$ that contains precisely the types marked by the **Mark** algorithm. We next define the rules to compute Type and Marked, and other relevant relations.

(III) A linear order over types. The first ingredient is a linear order over all possible types. Of course, we could hardcode the list of exponentially many types in the Datalog program, but then the program would not be of polynomial size, and, therefore, we need to compute it instead. To this end, for every $1 \leq i \leq m$, we inductively define i-ary relations first^i and last^i, and a $2i$-ary relation next^i, which provide the first, the last, and the successor elements from a linear order on $\{0,1\}^i$. In particular, given $\vec{u}, \vec{v} \in \{0,1\}^i$, the fact $\mathsf{next}^i(\vec{u}, \vec{v})$ will be true if $\vec{v}$ follows $\vec{u}$ in the ordering of $\{0,1\}^i$. The rules to populate next^i are quite standard (see, e.g., Theorem 4.5 in [5]). For the case $i = 1$, we simply add the following facts:

$$\mathsf{first}^1(0) \leftarrow \qquad\qquad \mathsf{last}^1(1) \leftarrow \qquad\qquad \mathsf{next}^1(0,1) \leftarrow$$

Then, for all $1 < i \leq m - 1$ we add the following rules:

$$\begin{aligned}
\mathsf{next}^{i+1}(0, \vec{u}, 0, \vec{v}) &\leftarrow \mathsf{next}^i(\vec{u}, \vec{v}) \qquad & \mathsf{first}^{i+1}(0, \vec{u}) &\leftarrow \mathsf{first}^i(\vec{u}) \\
\mathsf{next}^{i+1}(1, \vec{u}, 1, \vec{v}) &\leftarrow \mathsf{next}^i(\vec{u}, \vec{v}) & \mathsf{last}^{i+1}(1, \vec{u}) &\leftarrow \mathsf{last}^i(\vec{u}) \\
\mathsf{next}^{i+1}(0, \vec{u}, 1, \vec{v}) &\leftarrow \mathsf{last}^i(\vec{u}), \mathsf{first}^i(\vec{v}) &&
\end{aligned}$$

We can now collect in the relation Type all types in $\mathit{types}(\Sigma)$:

$$\mathsf{Type}(\vec{u}) \leftarrow \mathsf{first}^m(\vec{u}) \qquad\qquad \mathsf{Type}(\vec{v}) \leftarrow \mathsf{next}^m(\vec{u}, \vec{v})$$

(IV) Implementing Step ($\mathrm{M}_{\forall}$). First, we add the auxiliary facts $\mathsf{F}(0) \leftarrow$ and $\mathsf{T}(1) \leftarrow$ to P. For an m-tuple of variables $\vec{u}$ and an atom $\boldsymbol{\alpha}_j$ in the enumeration $\boldsymbol{\alpha}_1, \ldots \boldsymbol{\alpha}_m$, we use $\boldsymbol{\alpha}_j \in \vec{u}$ to denote the atom $\mathsf{T}(u_j)$, and $\boldsymbol{\alpha}_j \notin \vec{u}$ to denote the atom $\mathsf{F}(u_j)$. Then the step ($\mathrm{M}_{\forall}$), which marks types violating some rule of $\Sigma_{\forall}$, is implemented using the following rule for every DTGD $\varphi \rightarrow \bigvee_{i=1}^{n} H_i \in \Sigma_{\forall}$. Assume φ is a set of atoms $B_1, \ldots B_l$. Then, for all substitutions $h : \mathsf{vars}(\varphi) \rightarrow \mathbf{X}$, add the rule:

$$\mathsf{Marked}(\vec{u}) \leftarrow \mathsf{Type}(\vec{u}), h(B_1) \in \vec{u}, \ldots, h(B_l) \in \vec{u}, h(H_1) \notin \vec{u}, \ldots, h(H_n) \notin \vec{u}$$

There are at most w^k possible substitutions, where k is the arity of the guard atom in φ.

(V) Implementing Step ($\mathrm{M}_\exists$). The following rules are added for all TGDs $\sigma = B \to \exists \vec{y}.H \in \Sigma_\exists$ and for all substitutions $h : \mathsf{vars}(B) \to \mathbf{X}$. Assume $\mathbf{Z} = \mathsf{vars}(B) \cap \mathsf{vars}(H)$. Recall that we need to mark a type τ if there exists a substitution $h : \mathsf{vars}(B) \to \mathbf{X}$ such that $h(B) \in \tau$, and *for each* type τ', either τ' is marked or at least one of (C1′) or (C2′) holds. First, for each TGD σ and substitution h, we use an auxiliary $2m$-ary relation $\mathsf{MarkedOne}_{\sigma,h}$ to collect all such types τ'.

For collecting each τ' that satisfies condition (C0), we add:

$$\mathsf{MarkedOne}_{\sigma,h}(\vec{u},\vec{v}) \leftarrow \mathsf{Type}(\vec{u}), \mathsf{Marked}(\vec{v})$$

For (C1′) we add a rule for each $R(\vec{x})$, such that $\vec{x} \subseteq h(\mathbf{Z})$:

$$\begin{aligned} \mathsf{MarkedOne}_{\sigma,h}(\vec{u},\vec{v}) &\leftarrow \mathsf{Type}(\vec{u}), \mathsf{Type}(\vec{v}), R(\vec{x}) \in \vec{u}, R(\vec{x}) \notin \vec{v} \\ \mathsf{MarkedOne}_{\sigma,h}(\vec{u},\vec{v}) &\leftarrow \mathsf{Type}(\vec{u}), \mathsf{Type}(\vec{v}), R(\vec{x}) \notin \vec{u}, R(\vec{x}) \in \vec{v} \end{aligned}$$

To implement condition (C2′), we collect each τ' such that $h'_i(H) \notin \tau$ for each substitution $h'_i : \mathsf{vars}(H) \to \mathbf{X}$ with $h(\mathbf{Z}) = h'_i(\mathbf{Z})$. Notice that for k variables in $\vec{y}$, there are w^k possible substitutions, that is $1 \leq i \leq w^k$. We add:

$$\mathsf{MarkedOne}_{\sigma,h}(\vec{u},\vec{v}) \leftarrow \mathsf{Type}(\vec{u}), \mathsf{Type}(\vec{v}), h'_1(H) \notin \vec{v}, \ldots, h'_{w^k}(H) \notin \vec{v}$$

Intuitively, we want to infer $\mathsf{Marked}(\vec{u})$ if $h(\mathsf{B})$ is set to true in $\vec{u}$ and $\mathsf{MarkedOne}_{\sigma,h}(\vec{u},\vec{v})$ is true for all types (bit vectors) $\vec{v}$. To this aim, we need another $2k$-ary relation $\mathsf{MarkedUntil}_{\sigma,h}$. We add:

$$\begin{aligned} \mathsf{MarkedUntil}_{\sigma,h}(\vec{u},\vec{t}) &\leftarrow \mathsf{MarkedOne}_{\sigma,h}(\vec{u},\vec{t}), \mathsf{first}^m(\vec{t}) \\ \mathsf{MarkedUntil}_{\sigma,h}(\vec{u},\vec{v}) &\leftarrow \mathsf{MarkedUntil}_{\sigma,h}(\vec{u},\vec{t}), \mathsf{next}^m(\vec{t},\vec{v}), \mathsf{MarkedOne}_{\sigma,h}(\vec{u},\vec{v}) \end{aligned}$$

Intuitively, with the above rules we traverse all types checking the conditions (C0), (C1′) and (C2′) described in ($\mathrm{M}_\exists$). If we manage to reach the last type, and if $h(\mathsf{B}) \in \vec{u}$, we know the condition is satisfied. We add the following rule:

$$\mathsf{Marked}(\vec{u}) \leftarrow \mathsf{MarkedUntil}_{\sigma,h}(\vec{u},\vec{t}), h(\mathsf{B}) \in \vec{u}, \mathsf{last}^m(\vec{t})$$

(VI) Forbidding marked types in the core. Finally, we need to forbid each tuple of constants in the generated core database from having a type from Marked. For all $0 \leq j \leq m$ we take a fresh $(j+w)$-ary relation symbol Proj^j. We first add:

$$\mathsf{Proj}^m(\vec{\mathbf{x}},\vec{u}) \leftarrow \mathsf{const}^w(\vec{\mathbf{x}}), \mathsf{Marked}(\vec{u})$$

Intuitively, we collect in $\mathsf{const}^w(\vec{\mathbf{x}})$ all the possible w-ary tuples of constants from the core database. We now project away bits from the Proj^j relations by looking at the actual types of tuples of constants. We add the following rules for all $1 \leq j \leq m$:

$$\mathsf{Proj}^{j-1}(\vec{\mathbf{x}},\vec{u}) \leftarrow \mathsf{Proj}^j(\vec{\mathbf{x}},\vec{u},1), \boldsymbol{\alpha}_j \qquad \mathsf{Proj}^{j-1}(\vec{\mathbf{x}},\vec{u}) \leftarrow \mathsf{Proj}^j(\vec{\mathbf{x}},\vec{u},0), \overline{\boldsymbol{\alpha}_j}$$

Finally, we need to rule out situations when the extracted type of a tuple of constants over the core database is marked by adding the constraint $\leftarrow \mathsf{Proj}^0(\vec{\mathbf{x}})$.

This completes the polynomial translation of a query (Σ, q) where Σ is a set of guarded DTGDs to a Datalog$^\vee$ query (P, q). Next, we provide a theorem that captures the correctness of the above translation.

▶ **Theorem 8.** *For a query (Σ, q), where Σ is a normalized theory of guarded DTGDs, we can build a query (P, q), where P is a set of disjunctive* Datalog *rules, such that $\mathsf{ans}(\Sigma, q, D) = \mathsf{ans}(P, q, D)$ for any given database D over $\mathsf{sch}(\Sigma)$. The construction requires only polynomial time whenever width(Σ) is bounded.*

From the complexity of Datalog$^\vee$ [12], we infer a coNExpTime upper bound in combined complexity for guarded DTGDs whose normalized form uses only predicates whose arities are bounded by a constant. However, we use disjunction in a limited way: only in the rules in (II), whose heads have only two atoms over predicates of bounded arity. The resulting program falls into a class of programs that can be evaluated in (deterministic) exponential time, see Appendix A for more details. This matches the ExpTime-completeness of satisfiability of guarded DTGDs with bounded predicate arities [20].

▶ **Proposition 9.** *Deciding $\vec{c} \in \mathsf{ans}(P, q, D)$, where P is the disjunctive* Datalog *program obtained from the above translation of a set Σ of guarded DTGDs, can be done in* ExpTime.

5 Non-Disjunctive TGDs

Now we consider the case of plain guarded TGDs, that have no disjunction. Recall that in normalized theories of TGDs, each σ takes the form $B \to \exists \vec{y}.H$ or $\varphi \to H$, where B, H are atoms and φ is a set of atoms, all with terms from Δ_v.

In this section, we provide a rewriting of queries given by a theory of TGDs into plain Datalog. That is, given a query of the form (Σ, q), where Σ is a theory of TGDs, we obtain (P, q), where P is a (non-disjunctive) plain Datalog program. The resulting program is of polynomial size whenever *width*(Σ) is bounded, unlike previous rewritings that exist in the literature for related query languages.

5.1 From Marked Types to Horn Rules

To obtain a non-disjunctive rewriting for TGDs, we leverage to a great extent the results of the previous section, but proceed somewhat differently. We start by observing that if in the input (Σ, q), the TGDs in Σ contain no disjunction, almost all rules of the program described above are in plain Datalog. However, the rules in step **(II)** are disjunctive, and this is a major issue. Indeed, the basic strategy that we described in Section 3, namely, to obtain a core and then check if it can be suitably extended, is intrinsically based on the ability to use disjunction to generate different core instances. Therefore, to obtain a program without disjunction, we must proceed differently. Our strategy is to use the marked types to obtain a new set of non-disjunctive TGDs Σ_{Horn}, which correctly capture the consequences of Σ. This TGDs are *full*, that is, they have no existentially quantified variables, and they can be easily realized by Datalog rules.

Towards obtaining Σ_{Horn}, let $\mathsf{Marked}(\Sigma)$ denote the set of all the types that are marked by Algorithm 1 executed on Σ. We show that (independently of whether Σ contains or not disjunctive heads), $\mathsf{Marked}(\Sigma)$ defines a set of full DTGDs Σ_M that completely captures the consequences of Σ. Therefore, we can replace Σ by Σ_M, while preserving the entailment of facts. Later we show that Σ_M can in turn be replaced by the desired Σ_{Horn}.

To define Σ_M, recall that $\mathbf{A}$ is the set of all atoms that can occur in *types*(Σ). Each type τ defines a full DTGD *rule*(τ) as follows:

$$\tau \to \bigvee_{\boldsymbol{\alpha} \in \mathbf{A} \setminus \tau} \boldsymbol{\alpha}$$

Consider a core instance D_c, which we would like to extend to satisfy Σ. Intuitively, if $\tau \in \mathsf{Marked}(\Sigma)$, such a rule expresses that if all the atoms in τ are true in D_c, then some atom not in τ must be true as well, as otherwise the marked type τ would be realized.

Formally, we can show the following. We remark that we formulate the lemma for TGDs, but it holds also for DTGDs.

▶ **Lemma 10.** *Let Σ be a set of guarded TGDs and let Σ_M be the set of all rule(τ) with $\tau \in \mathsf{Marked}(\Sigma)$. Then, for any given atom $\boldsymbol{\alpha}$ and database D, $(D, \Sigma) \not\models \boldsymbol{\alpha}$ iff $(D, \Sigma_M) \not\models \boldsymbol{\alpha}$.*

Proof. For the "⇒" direction, let I be an instance such that $I \models (D, \Sigma)$ and $\boldsymbol{\alpha} \notin I$. We show that $I \models (D, \Sigma_M)$. Let D_c be the restriction of I to constants in $\mathsf{dom}(D)$, that is D_c is a core instance for (D, Σ). The latter together with Theorem 6 imply that Bob has a non-losing strategy on D_c, which together with Theorem 7 imply that none of the types realized in D_c is marked by $\mathbf{Mark}(\Sigma)$. We claim that $D_c \models (D, \Sigma_M)$. Towards a contradiction assume there there exists a *rule*(τ) $\in \Sigma_M$ such that $D_c \not\models \tau \rightarrow \bigvee_{\boldsymbol{\alpha} \in \mathbf{A} \setminus \tau} \boldsymbol{\alpha}$. That D_c doesn't satisfy $\tau \rightarrow \bigvee_{\boldsymbol{\alpha} \in \mathbf{A} \setminus \tau} \boldsymbol{\alpha}$ intuitively means that one can match τ to D_c, but cannot match any of $\boldsymbol{\alpha}$ to D_c. The latter imply that τ must be realized in D_c, but this contradicts that $\tau \in \mathsf{Marked}(\Sigma)$ by definition of *rule*(τ).

For the "⇐" direction, let I be an instance such that $I \models (D, \Sigma_M)$ and $\boldsymbol{\alpha} \notin I$. Recall that no rule in Σ_M enforces the invention of nulls, that is w.l.o.g. we assume $\mathsf{terms}(I) = \mathsf{dom}(D)$. One could easily show that I is a core instance for (D, Σ). Observe that none of the types realized in I are marked by $\mathbf{Mark}(\Sigma)$ as otherwise, by definition of $\mathsf{Marked}(\Sigma)$, the corresponding type would appear in the body of a rule in Σ_M which together with $I \models \Sigma_M$ contradict the assumption that the corresponding type is realized in I. The latter and Theorem 7 imply Bob has a non-losing strategy on I which together with assumption $\boldsymbol{\alpha} \notin I$, and by Theorem 6 imply $(D, \Sigma) \not\models \boldsymbol{\alpha}$. ◀

In general, Σ_M is a set of full DTGDs, but if Σ is not disjunctive, then we can obtain a set of non-disjunctive full TGDs that achieves the same effect. The core intuition is that, for TGDs, the disjunctive heads express choices between atoms that are *irrelevant*, and one can disregard such choices, obtaining full TGDs that enforce only the necessary atoms. To identify the necessary and the irrelevant atoms, we rely on what we call *abstract types*, which are types augmented with a set τ_{ir} of irrelevant atoms.

▶ **Definition 11.** An *abstract type* (over Σ) is a pair (τ_c, τ_{ir}) of (possibly empty) disjoint subsets of $\mathbf{A}$.

Intuitively, an atom $\boldsymbol{\alpha}$ is irrelevant for τ_c, with $\boldsymbol{\alpha} \notin \tau_c$, if both τ_c and $\tau_c \cup \{\boldsymbol{\alpha}\}$ are marked. The algorithm **Horn** takes as input the set of all abstract types $(\tau, \emptyset)$ where $\tau \in \mathsf{Marked}(\Sigma)$, and outputs a set of abstract types. The pair (τ_c, τ_{ir}) will be returned by the algorithm if $\tau = \tau_c \cup \tau'_{ir}$ is marked for each $\tau'_{ir} \subseteq \tau_{ir}$, that is, if the presence of the atoms in τ_{ir} is irrelevant for the marking of τ_c. We use $\mathsf{Horn}(\Sigma)$ to denote the result of running Algorithm 2 on $\mathsf{Marked}(\Sigma)$ for a given theory Σ.

▶ **Lemma 12.** *Let (τ_c, τ_{ir}) be an abstract type. Then $(\tau_c, \tau_{ir}) \in \mathsf{Horn}(\Sigma)$ if and only if $\tau_c \cup \tau'_{ir} \in \mathsf{Marked}(\Sigma)$ for each $\tau'_{ir} \subseteq \tau_{ir}$.*

If $(\tau_c, \tau_{it}) \in \mathsf{Horn}(\Sigma)$, then the irrelevant atoms in τ_{ir} can be omitted from the head of *rule*(τ_c), as shown next.

▶ **Lemma 13.** *Let (τ_c, τ_{ir}) be an abstract type. Then $(\tau_c, \tau_{ir}) \in \mathsf{Horn}(\Sigma)$ if and only if $\Sigma \models \tau_c \rightarrow \bigvee_{\boldsymbol{\alpha} \in \mathbf{A} \setminus (\tau_c \cup \tau_{ir})} \boldsymbol{\alpha}$.*

Algorithm 2: Horn.

input : a set M_0 of types

output : set of abstract types

$N \leftarrow \{(\tau_c, \emptyset) \mid \tau_c \in M_0\}$

repeat

  if *there exist* $\{(\tau_c, \tau_{ir}), (\tau'_c, \tau_{ir})\} \subseteq N$ *with* $\tau_c = \tau'_c \cup \{\boldsymbol{\alpha}\}$ *for some atom* $\boldsymbol{\alpha} \notin \tau'_c$ **then**

    add the pair $(\tau'_c, \tau_{ir} \cup \{\boldsymbol{\alpha}\})$ to N

until *no further changes*

return N

These rules are still disjunctive, however, we can show that for every marked type, there exists a corresponding abstract type that covers all atoms over the signature of the theory, except for one, and hence it induces a full TGD with only one atom in the head. The proof of the next lemma uses the well-known property that (non-disjunctive) TGDs are convex, that is, $\Sigma \models \bigwedge_{i=1}^{k} \boldsymbol{\alpha}_i \rightarrow \bigvee_{j=1}^{n} \boldsymbol{\alpha}_j$ implies that there exists $l \in \{1, \ldots, n\}$ such that $\Sigma \models \bigwedge_{i=1}^{k} \boldsymbol{\alpha}_i \rightarrow \boldsymbol{\alpha}_l$ (see Lemma 2 in [3]).

▶ **Lemma 14.** *Let Σ be a set of TGDs, $M = \mathsf{Mark}(\Sigma)$, and let $N = \mathsf{Horn}(\Sigma)$. Then $\tau_c \in M$ implies that there exists a pair $(\tau_c, \tau_{ir}) \in N$ such that $\tau_c \cup \tau_{ir} \cup \{\boldsymbol{\alpha}\} = \mathbf{A}$ for some $\boldsymbol{\alpha} \in \mathbf{A}$.*

Now we state the central result of this section: from $\mathsf{Horn}(\Sigma)$ one can construct the desired set of full TGDs Σ_{Horn}, which has the same atomic consequences as the original Σ.

▶ **Theorem 15.** *Let Σ be a set of TGDs, and let Σ_{Horn} be the set of all full TGDs $\tau_c \rightarrow \boldsymbol{\alpha}$ such that there is an abstract type (τ_c, τ_{ir}) in $\mathsf{Horn}(\Sigma)$ with $\tau_c \cup \tau_{ir} \cup \{\boldsymbol{\alpha}\} = \mathbf{A}$. Then, for any given atom α and database D, we have $D \cup \Sigma \models \alpha$ iff $D \cup \Sigma_{\mathsf{Horn}} \models \alpha$.*

5.2 Translation into Datalog

Assume a normalized theory Σ of guarded TGDs. Also, assume $w = \mathit{width}(\Sigma)$, and n is the number of distinct predicate symbols occurring in $\mathsf{sch}(\Sigma)$. Similarly as we did in Section 4, we build next a program P such that the queries (Σ, q) and (P, q) have the same answers for all databases D over $\mathsf{sch}(\Sigma)$. As before, P is polynomial under the assumption that w is bounded by a constant. Crucially, in this case, P will be a plain DATALOG program.

First of all, P contains the rules described in **(I)**, **(III)**, **(IV)**, and **(V)** in Section 4. These rules that collect the constants and implement the marking algorithm from the previous translation do not use the rules with disjunction in **(II)**. Additionally, we include the following rules that implement the algorithm **Horn** and the inferences from Σ_{Horn}.

(VII) Generating abstract types from marked types. We add rules that construct the set $\mathsf{Horn}(\Sigma)$ from Algorithm 2, creating abstract types from the marked types. Recall that we represent (regular) types as m-sequences of 1 and 0, where the i-th value indicates whether the atom $\boldsymbol{\alpha}_i$ in the enumeration is contained in the type or not. We extend this representation using an additional constant 2 in position i to indicate that an atom $\boldsymbol{\alpha}_i$ is in the set τ_{ir} of an abstract type (τ_c, τ_{ir}). We will write $\vec{u}_{[i,j]}$ to denote the tuple $(u_i, u_{i+1}, \ldots, u_j)$; if $j < i$ then $\vec{u}_{[i,j]}$ refers to the empty tuple. For all $1 \leq i \leq m$, we add the rules:

$$\mathsf{Marked}(\vec{u}_{[1,i-1]}, 2, \vec{u}_{[i+1,m]}) \leftarrow \mathsf{Marked}(\vec{u}_{[1,i-1]}, 0, \vec{u}_{[i+1,m]}), \mathsf{Marked}(\vec{u}_{[1,i-1]}, 1, \vec{u}_{[i+1,m]})$$

(VIII) Constructing all horn types. We call a type *horn* if only one bit is set to 0, and the other bits to 1 or 2. Intuitively, horn types stand for abstract types of the form (τ_c, τ_{ir}) that can be converted into full TGDs of the form $\tau_c \to \boldsymbol{\alpha}$, where the atoms in τ_c are represented with 1, the atoms in τ_{ir} with 2, and atom $\boldsymbol{\alpha}$ with 0. First, we use a new m-ary predicate symbol Horn to construct all possible Horn types. To this end, we first add the following facts for all $0 \leq i < m$:

$$\mathsf{Horn}(\underbrace{1,\ldots,1}_{i},0,\underbrace{1,\ldots,1}_{m-i-1}) \leftarrow$$

To construct all possible alternations of 1 and 2, we add the following rule for all $1 \leq i \leq m$:

$$\mathsf{Horn}(\vec{u}_{[1,i-1]},2,\vec{u}_{[i+1,m]}) \leftarrow \mathsf{Horn}(\vec{u}_{[1,i-1]},1,\vec{u}_{[i+1,m]})$$

Finally, we use another m-ary relation $\mathsf{MarkedHorn}$ to collect all types from $\mathsf{Horn}(\Sigma)$ that are marked and that can be converted into full TGDs. The rule is as follows:

$$\mathsf{MarkedHorn}(\vec{u}) \leftarrow \mathsf{Marked}(\vec{u}),\mathsf{Horn}(\vec{u})$$

(IX) Implementing the full TGDs from Σ_{Horn}. Finally, we guarantee each $\tau_c \to \boldsymbol{\alpha} \in \Sigma_{\mathsf{Horn}}$ is satisfied. For each Horn type in $\mathsf{MarkedHorn}$, whenever the atoms in positions with 1 are made true by some instantiation of the variables with constants, we infer the atom in the 0 position. For all $1 \leq i,j \leq m$ we take a fresh $(j+1)$-ary predicate symbol Proj_i^j, where i indicates the position of the bit labelled with 0. For all $1 \leq i \leq m$, we add:

$$\mathsf{Proj}_i^m(\vec{\mathbf{x}},\vec{u}_{[1,i-1]},0,\vec{u}_{[i+1,m]}) \leftarrow \mathsf{MarkedHorn}(\vec{u}_{[1,i-1]},0,\vec{u}_{[i+1,m]}),\mathsf{const}^w(\vec{\mathbf{x}})$$

We will now project away bits from the Proj_i^j relations. We add the following rules for all $1 \leq j \leq m$ and for all $1 \leq i \leq m$:

$$\begin{aligned}
\mathsf{Proj}_i^{j-1}(\vec{\mathbf{x}},\vec{u}) &\leftarrow \mathsf{Proj}_i^j(\vec{\mathbf{x}},\vec{u},1),\boldsymbol{\alpha}_j\\
\mathsf{Proj}_i^{j-1}(\vec{\mathbf{x}},\vec{u}) &\leftarrow \mathsf{Proj}_i^j(\vec{\mathbf{x}},\vec{u},2)\\
\mathsf{Proj}_i^{j-1}(\vec{\mathbf{x}},\vec{u}) &\leftarrow \mathsf{Proj}_i^j(\vec{\mathbf{x}},\vec{u},0)
\end{aligned}$$

If we reach the last bit, we add the atom whose bit is set to 0. For all $1 \leq i \leq m$, we add:

$$\boldsymbol{\alpha}_i \leftarrow \mathsf{Proj}_i^0(\vec{\mathbf{x}})$$

This completes the polynomial translation of a query (Σ, q), where Σ is a set of guarded TGDs (without disjunction), to a (plain) Datalog query (P, q). Next, we provide a theorem that captures the correctness of the above translation.

▶ **Theorem 16.** *For a query (Σ, q), where Σ is a normalized theory of guarded TGDs, we can build a query (P, q), where P is a set of (plain)* Datalog *rules such that $\mathsf{ans}(\Sigma, q, D) = \mathsf{ans}(P, q, D)$ for any given database D over $\mathsf{sch}(\Sigma)$. Moreover, if width(Σ) is bounded, then (P, q) can be built in polynomial time.*

6 Discussion and Conclusions

In this paper, we have studied a very rich query language, where a query predicate is given together with a set of guarded (D)TGDs expressing domain knowledge. For queries (Σ, q) with q an instance query, we have proposed rewritings to Datalog$^\vee$ for Σ a set of guarded

DTGDs, and to (plain) Datalog for Σ a set of guarded (non-disjunctive) TGDs. To our best knowledge, these are the first such rewritings that are *polynomial* under the relevant, useful restriction that the maximal number of variables in the guarded (D)TGDs is bounded by a constant. If the number of variables per rule is not bounded, the rewriting is exponential in the size of the query. This is natural: answering guarded (D)TGD queries is 2ExpTime-hard [4], and cautious inference from Datalog$^\vee$ queries is coNExpTime-complete, and from Datalog is ExpTime-complete [12]. For the case with disjunction, a polynomial rewriting would imply coNExpTime = 2ExpTime, and for the case without disjunction a polynomial rewriting cannot exist since ExpTime $\neq$ 2ExpTime.

Our results can easily be generalized to guarded (D)TGDs with *acyclic conjunctive queries* or with *quantifier-free conjunctive queries*, that is conjunctive queries with no existential variables. Acyclic conjunctive queries can be rewritten in polynomial time into a set of rules that fall into the guarded (D)TGD fragment (see, e.g., the proof of Theorem 6.2 in [8]) and the quantifier-free conjunctive queries are syntactically restricted to ensure that the relevant variable assignments only map into the constants of the input database. Additionally, our techniques can be easily extended to support *nearly guarded (D)TGDs* [18], which are a strict extension of the guarded variants with non-guarded rules. Indeed, the latter rules can 'operate' only on constants from the input database and we just need to add a simple condition that ensures the satisfaction of these rules by the core instances. Identifying other cases that are in ExpTime and that allow for polynomial Datalog rewritings is an open research problem for future work.

Extending our approach to support other query languages is also an interesting direction for the future. We remark, however, that generalizing our translation to guarded DTGDs in the presence of arbitrary conjunctive queries while remaining polynomial is out of reach under common assumptions in complexity theory. Indeed, the associated query answering problem is 2ExpTime-hard in combined complexity already for a restricted fragment of arity two [24]. For the case of TGDs without disjunction a polynomial rewriting to Datalog for conjunctive queries may be feasible. However, although adapting the ideas in this work to conjunctive queries may be possible, and at the cost of an exponential blow-up for DTGDs, it seems a challenging task. The guarded DTGDs we studied in this paper do not allow for constants. Extending our translation to support constants remains for future work.

References

1 Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. Polynomial datalog rewritings for expressive description logics with closed predicates. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 878–885. IJCAI/AAAI Press, 2016. URL: `http://www.ijcai.org/Abstract/16/129`.

2 Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philosophical Logic*, 27(3):217–274, 1998. `doi:10.1023/A:1004275029985`.

3 Vince Bárány, Michael Benedikt, and Balder ten Cate. Rewriting guarded negation queries. In *Proc. of MFCS' 13*, pages 98–110. ACM, 2013.

4 Vince Bárány, Georg Gottlob, and Martin Otto. Querying the guarded fragment. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 1–10. IEEE Computer Society, 2010. `doi:10.1109/LICS.2010.26`.

5 Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *PVLDB*, 5(11):1328–1339, 2012. URL: http://vldb.org/pvldb/vol5/p1328_vincebarany_vldb2012.pdf.

6 Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984. doi:10.1145/1634.1636.

7 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014. doi:10.1145/2661643.

8 Pierre Bourhis, Marco Manna, Michael Morak, and Andreas Pieris. Guarded-based disjunctive tuple-generating dependencies. *ACM Trans. Database Syst.*, 41(4):27:1–27:45, 2016. doi:10.1145/2976736.

9 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013. doi:10.1613/jair.3873.

10 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007. doi:10.1007/s10817-007-9078-x.

11 Alin Deutsch and Val Tannen. Reformulation of XML queries and constraints. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings*, volume 2572 of *Lecture Notes in Computer Science*, pages 225–241. Springer, 2003. doi:10.1007/3-540-36285-1_15.

12 Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997. doi:10.1145/261124.261126.

13 Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for horn-shiq plus rules. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4931.

14 Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artif. Intell.*, 213:42–59, 2014. doi:10.1016/j.artint.2014.04.004.

15 Georg Gottlob, Marco Manna, Michael Morak, and Andreas Pieris. On the complexity of ontological reasoning under disjunctive existential rules. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2012. doi:10.1007/978-3-642-32589-2_1.

16 Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial combined rewritings for existential rules. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014.* AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7973.

17 Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial rewritings for linear existential rules. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2992–2998. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/423.

18 Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. Expressiveness of guarded existential rule languages. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 27–38. ACM, 2014. doi:10.1145/2594538.2594556.

19 Georg Gottlob and Thomas Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4510.

20 Erich Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.

21 Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reasoning*, 39(3):351–384, 2007. doi:10.1007/s10817-007-9080-3.

22 Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Datalog rewritability of disjunctive datalog programs and its applications to ontology reasoning. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1077–1083. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8201.

23 Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to ontology-based data access. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2656–2661. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-442.

24 Carsten Lutz. Inverse roles make conjunctive queries hard. In Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris, editors, *Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8-10 June, 2007*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. URL: http://ceur-ws.org/Vol-250/paper_3.pdf.

25 Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-based data access with closed predicates is inherently intractable(sometimes). In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1024–1030. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6870.

26 Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-based data access with closed predicates is inherently intractable(sometimes). In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1024–1030. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6870.

27 Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Worst-case optimal reasoning for the horn-dl fragments of OWL 1 and 2. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press, 2010. URL: http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1296.

28 Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *J. Applied Logic*, 8(2):186–209, 2010. doi:10.1016/j.jal.2009.09.004.

29 Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49, 2015. doi:10.1016/j.websem.2015.02.001.

A Appendix

A.1 Proofs of Section 3

To prove Theorem 6, we first introduce the following lemma.

▶ **Lemma 17.** *Assume a theory Σ, a database D and an assertion α. Then $(\Sigma, D) \not\models \alpha$ iff there exists a core instance D_c for (Σ, D) s.t.*
(1) *$\alpha \notin D_c$, and*
(2) *there exists an extension I of D_c s.t. $I \models (\Sigma, D)$.*

Proof. For the "$\Leftarrow$" direction, it is clear that I is the desired instance that satisfies (Σ, D) (condition *(2)*) and is such that $\alpha \notin I$. The latter is true from condition *(1)* and the fact that I is an extension of D_c. For the "$\Rightarrow$" direction, assume I with $\alpha \notin I$ is an instance that satisfies (Σ, D). W.l.o.g. assume that $\mathsf{sch}(I) \subseteq \mathsf{sch}(\Sigma)$. We construct a core database D_c such that $D_c = \{R(\vec{t}) \mid R(\vec{t}) \in I, \vec{t} \subseteq \mathsf{dom}(D)\}$. We claim that D_c is a core interpretation for (Σ, D). Condition *(c1)* of Definition 3 holds by construction of D_c. Also, D_c satisfies all the rules from $\Sigma_\forall \subseteq \Sigma$ as these rules can only create new atoms with terms that range over constants from $\mathsf{dom}(D)$, and thus condition *(c2)* is also satisfied by D_c. ◀

Proof of Theorem 6. By Lemma 17, we only need to show that, for any given core D_c, there is a non-losing strategy *str* for Bob on D_c if and only if D_c can be extended into an instance I that satisfies (Σ, D).

For the "$\Rightarrow$" direction, from an arbitrary non-losing *str* for D_c, we build I as follows. We write $\vec{c}_\tau$ to denote a tuple of constants realizing τ, that is $\tau = \mathit{type}(\vec{c}_\tau, D_c)$. In the following, we denote by $\mathit{rn}(D_c, \mathit{str})$ the set of all finite runs $\vec{c}_\tau \sigma_1 h_1 \tau_1 \sigma_2 h_2 \tau_2 \dots$ that follow *str*. Additionally, we will denote with $\mathit{fv}(\sigma)$ the set $\mathsf{vars}(B) \cap \mathsf{vars}(H)$ of variables where $\sigma \in \Sigma$ is a TGD of the form $B \to \exists \vec{y}.H \in \Sigma$.

We let I be the set of atoms $R(t_1, \dots, t_\ell)$ such that one of the following holds:
(a) there exists a run $r = \vec{c}_\tau$ in $\mathit{rn}(D_c, \mathit{str})$ with $R(\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_\ell}) \in \tau$ and $t_j = c_{i_j} \in \vec{c}_\tau$ for each $j \in [1, \ell]$, or
(b) there exists a run $r = \vec{c}_\tau \dots \sigma_i h_i \tau_i$ in $\mathit{rn}(D_c, \mathit{str})$ for $i \geq 1$ with $R(\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_\ell}) \in \tau_i$, where σ_i is a TGD of the form $B \to \exists \vec{y}.H \in \Sigma$ such that, for each t_j with $j \in [1, \ell]$, the following holds:
  (1) t_j is the labelled null $r'\mathbf{x}_{i_j}$, if
    (i) $r' = \vec{c}_\tau \dots \sigma_k h_k \tau_k$ and $r = r' \dots \sigma_i h_i \tau_i$ with $1 \leq k \leq i$,
    (ii) $\mathbf{x}_{i_j} \notin h_k(\mathit{fv}(\sigma_k))$, and
    (iii) $\mathbf{x}_{i_j} \in h_l(\mathit{fv}(\sigma_l))$ for each $k < l \leq i$ if $i \neq k$.
  (2) t_j is the constant c_{i_j} if:
    (i) $\mathbf{x}_{i_j} \in h_l(\mathit{fv}(\sigma_l))$ for each $l \in [1, i]$.

First, we show that I is an extension of D_c, that is for each predicate $R \in \mathsf{sch}(\Sigma)$ and for each tuple $\vec{c}$ of constants from $\mathsf{dom}(D_c)$, $R(\vec{c}) \in D_c$ if and only if $R(\vec{c}) \in I$. Observe that, by construction of I, condition (a), D_c is contained in I. That no other ground atom (that does not appear in D_c) is added to I is ensured by *(b)* point (2) and by definition of a strategy, more precisely by condition (C1). That is (C1) guarantees that each type τ_i in a run $r = \vec{c}_\tau \dots \sigma_i h_i \tau_i$ in $\mathit{rn}(D_c, \mathit{str})$ coincides with the type τ_{i-1} on the shared variables

$h_i(fv(\sigma_i))$. Thus τ_i coincides with τ on the variables propagated throughout the run up to i and that those variables are mapped to the same constants in the core is ensured by *(b)* point (2) in the model I constructed above.

We now show that I models (Σ, D). That $I \models D$ results by definition of a core instance D_c. To prove $I \models \Sigma$, we first show that I satisfies all guarded DTGDs in $\Sigma_\forall$. Assume an arbitrary guarded DTGD σ of the form $\varphi \to \bigvee_{i=1}^{n} H_i$ in $\Sigma_\forall$ and let h be an arbitrary substitution from $\mathsf{vars}(\varphi)$ to $\mathsf{terms}(I)$ such that $h(\varphi) \subseteq I$. Recall that the guard atom in φ contains all the variables that appear in σ. We show that there must exist an $i \in [n]$ with $h(H_i) \in I$. We distinguish the following two cases:

(i) If $h(\varphi) \subseteq D_c$, then by definition of a core instance, $D_c \models \Sigma_\forall$. In particular D_c satisfies σ, that is there exists an $i \in [n]$ with $h(H_i) \in D_c$. The latter together with $D_c \subseteq I$ implies $h(H_i) \in I$.

(ii) Otherwise, let the ℓ-ary atom $R(x_1, \ldots, x_\ell)$ be the guard atom in φ and let $h(x_i) = t_i$ for each $i \in [1, \ell]$, such that $R(t_1, \ldots, t_\ell) \in I$ and $R(t_1, \ldots, t_\ell) \notin D_c$, that is there exists an $i \in [1, \ell]$ such that t_i is a labelled null. Observe that each t_i is either a labelled null $r\mathbf{x}_{i_j}$, or a constant c_{i_j}, or a labelled null $r'\mathbf{x}_{i_j}$ where r' is such that $r = r' \ldots \sigma_i h_i \tau_i$. Let $t_j \in \{t_1, \ldots, t_\ell\}$ be the labelled null $r\mathbf{x}_{i_j}$ with the longest prefix run $r \in rn(D_c, str)$, that is there is no $k \in [1, \ell]$ such that $t_k = r'\mathbf{x}_{i_k}$, $r' = rr''$, and $r' \neq r$ and assume r is of the form $\vec{c}_\tau \ldots \sigma_i h_i \tau_i$ and note that $i \geq 1$. First we claim that the corresponding $R(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}) \in \mathit{tail}(r) = \tau_i$, where each $\mathbf{x}_{i_j}$ is in the position of a labelled null $t_j = r\mathbf{x}_{i_j}$ or the labelled null $t_j = r'\mathbf{x}_{i_j}$ with $r = r'r''$ or the constant $c_{i_j} \in \vec{c}_\tau$. Towards a contradiction assume $R(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}) \notin \tau_i$. Then by construction of I, there exists a run $r_l \in rn(D_c, str)$ of the form $r \ldots \sigma_l h_l \tau_l$ such that $R(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}) \in \mathit{tail}(r_l)$ and each variable $\mathbf{x}_{i_1}$ with $t_j = r\mathbf{x}_{i_j}$ must have been shared throughout the run up to τ_l from τ_i and the other variables must come from earlier types in r. But then by condition (C1) of the game all types appearing in the run r_l from $\mathit{tail}(r) = \tau_i$ to $\mathit{tail}(r_l) = \tau_l$ must coincide on these shared variables and hence the atom $R(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell})$ must be in each type from τ_l to τ_i. The latter contradicts the assumption that $R(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}) \notin \tau_i$.

That $R(x_1, \ldots, x_\ell)$ is the guard atom in φ and that $h(\varphi) \subseteq I$ imply $R'(t_{i_j}, \ldots, t_{i_k}) \in I$ for all atoms $R'(x_{i_j}, \ldots, x_{i_k})$ in φ where $\{t_{i_j}, \ldots, t_{i_k}\} \subseteq \{t_1, \ldots, t_\ell\}$. Assume an arbitrary atom $R'(x_{i_j}, \ldots, x_{i_k})$ in φ that is not the guard atom, that is $R'(t_{i_j}, \ldots, t_{i_k}) \in I$. We claim that the corresponding $R'(\mathbf{x}_{i_j}, \ldots, \mathbf{x}_{i_k})$ is also in τ_i. Let $t_{i_l} \in \{t_{i_j}, \ldots, t_{i_k}\}$ be $r'\mathbf{x}_{i_l}$ with the longest prefix run r' from the terms of $R'(t_{i_j}, \ldots, t_{i_k})$. Indeed, if $r' = r$ then $R'(\mathbf{x}_{i_l}, \ldots, \mathbf{x}_{i_k}) \in \tau_i$ by construction of I. Otherwise observe that $r = r' \ldots \sigma_i h_i \tau_i$ since $t_{i_l} \in \{t_1, \ldots, t_\ell\}$. Then it is clear by construction of I that the atom $R'(t_{i_l}, \ldots, t_{i_k})$ is added to I because $R'(\mathbf{x}_{i_l}, \ldots, \mathbf{x}_{i_k}) \in \mathit{tail}(r')$. We claim that $R'(\mathbf{x}_{i_l}, \ldots, \mathbf{x}_{i_k}) \in \tau_j$ for each type τ_j occurring in the run r from $\mathit{tail}(r')$ to τ_i. By assumption that the guard $R(\mathbf{x}_{i_l}, \ldots, \mathbf{x}_{i_\ell}) \in \tau_i$ and by construction points (1) and (2), each $\mathbf{x}_{i_j}$ appears in each type throughout the run from $\mathit{tail}(r')$ to τ_i which together with condition (C2) imply $R'(\mathbf{x}_{i_l}, \ldots, \mathbf{x}_{i_k})$ must have be carried throughout the run and in particular $R'(\mathbf{x}_{i_l}, \ldots, \mathbf{x}_{i_k})$ must be in τ_i. Finally, with a similar argument one prove the claim if each t_{i_j} is a constant from the database.

The above implies that there is a substitution $h' : \{x_1, \ldots, x_\ell\} \to \{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}\}$ such that $h'(\varphi) \subseteq \tau_i$. Since by construction each $r \in rn(D_c, str)$ follows the strategy str which is non-losing, thus $\tau_i \in \mathsf{LC}(\Sigma, I)$, that is τ_i satisfies the condition $\mathsf{LC}_\forall$, thus there exists $i \in [n]$ with $H_i(\mathbf{x}_{i_j}, \ldots, \mathbf{x}_{i_k}) \in \tau_i$, hence by construction of I also the corresponding $H_i(t_j, \ldots, t_k) \in I$ and $H_i(t_j, \ldots, t_k) = h(H_i(x_j, \ldots, x_k))$, which shows the claim.

It is left to show that I satisfies all the TGDs σ in $\Sigma_\exists$ of the form $B \to \exists \vec{y}.H$. Recall that $fv(\sigma)$ is the set of shared variables $\mathsf{vars}(B) \cap \mathsf{vars}(H)$. Let h be an arbitrary substitution from $\mathsf{vars}(B)$ to $\mathsf{terms}(I)$ such that $h(B) \in I$. We show that there exists a substitution h' from $\mathsf{vars}(H)$ to $\mathsf{terms}(I)$ such that $h'(H) \in I$ and $h'(\mathbf{x}) = h(\mathbf{x})$ for all $\mathbf{x} \in fv(\sigma)$. Intuitively, that B is mapped to I implies there is a run with the corresponding atom in the tail type and hence there is a substitution from B to the type. This makes the TGD σ applicable to the type, and by the strategy there exists a new type that satisfies the head preserving the atoms over the shared variables, and hence there is an h' that satisfies the above properties. Formally, let $h(B)$ be the atom $B(t_1, \ldots, t_\ell) \in I$ such that $B(t_1, \ldots, t_\ell) \notin D_c$, that is at least one of $t_1, \ldots, t_\ell$ is a labelled null. The case when each t_i is a constant, that is $B(t_1, \ldots, t_\ell) \in D_c$, is analogous. Let $t_j \in \{t_1, \ldots, t_\ell\}$ be the labelled null $r\mathbf{x}_{i_j}$ that has the longest prefix run $r \in rn(D_c, str)$, that is there is no $k \in [1, \ell]$ such that $t_k = r'\mathbf{x}_{i_k}$, $r' = rr''$, and $r' \neq r$. Then by construction of I there exists a run $r \in rn(D_c, str)$ of the form $r = \vec{c}_\tau \ldots \sigma_i h_i \tau_i$ with $i \geq 1$, such that $B(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}) \in \tau_i$ where each $\mathbf{x}_{i_j}$ is in the position of the labelled null $t_j = r'\mathbf{x}_{i_j}$ or the constant c_{i_j} in $R(t_1, \ldots, t_\ell)$. Then let h_σ be the substitution from $\mathsf{vars}(B)$ to $\{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}\}$ such that for each $x \in \mathsf{vars}(B)$, $h_\sigma(x) = \mathbf{x}_{i_j}$ if $h(x)$ is the labelled null $r'\mathbf{x}_{i_j}$ or the constant $c_{i_j} \in \vec{c}_\tau$. Observe that $h_\sigma(B) = B(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell}) \in \tau_i$. Since str is a non-losing strategy, $str(\tau_i, \sigma, h_\sigma)$ is defined and hence there exists a type $\tau_i' = str(\tau_i, \sigma, h_\sigma)$ and a substitution $h'_\sigma : \mathsf{vars}(H) \to \mathbf{X}$ such that $h'_\sigma(H) \in \tau_i'$. Thus the run $r' = r\sigma h_\sigma \tau_i'$ is in $rn(D_c, str)$ with $h'_\sigma(H) \in \tau_i'$. Then, from $h_\sigma(fv(\sigma)) = h'_\sigma(fv(\sigma))$ (condition (C2) in the game) and by construction of I, $h'(H) \in I$, where h' is the substitution which coincided with h on the shared variables $fv(\sigma)$ and maps the other variables $x \in \mathsf{vars}(H) \setminus fv(\sigma)$ to $r'h'_\sigma(x)$.

For "$\Leftarrow$", assume an arbitrary model I of (Σ, D) that is an extension of D_c. We can extract from it a non-losing strategy for Bob as follows. First, let $rlz(I)$ be a the set of all types realized in I, and observe that $rlz(I) \subseteq \mathsf{LC}(\Sigma, D_c)$ and in particular that, for all $R(\vec{c}) \in D_c$, $type(\vec{c}, I) \in rlz(I)$. Then it suffices to set, for each type $\tau \in rlz(I)$ and each TGD $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ with $h(B) \in \tau$, $str(\tau, \sigma, h) = \tau'$ for an arbitrarily chosen type $\tau' \in rlz(I)$ that satisfies (C1) and (C2) which exists because I is a model, hence it satisfies all DTGDs in Σ. This str is a non-losing strategy for Bob. ◀

Proof. (Proof of Theorem 7.) For the "$\Rightarrow$" direction, let str be a non-losing strategy on D_c. It suffices to show that if a type is marked then it cannot occur in a run that follows str, which implies that if a type realized in D_c is marked by the algorithm **Mark**(Σ), then there is no non-losing strategy for D_c. The proof is by induction in the number of iterations needed to mark a type. For the base case, $k = 0$, assume a type $\tau \in types(\Sigma)$ is marked by ($\mathrm{M}_\forall$) in the algorithm **Mark**(Σ), then τ doesn't satisfy ($\mathrm{LC}_\forall$, that is $\tau \notin \mathsf{LC}(\Sigma, D_c)$. Hence there cannot exist a run r with $tail(r) = \tau$ since this contradicts the definition of non-losing str.

For the induction hypothesis, assume the claim holds for the first k iterations, that is assume that every type marked by the algorithm **Mark**(Σ) from ($\mathrm{M}_\forall$) or in the first k iterations of ($\mathrm{M}_\exists$) doesn't occur in a run that follows str. We show that the claim holds also for the next type that gets marked by (M_Σ). Thus for the inductive case, let $\tau \in types(\Sigma)$ be the first type marked by the algorithm after some iteration $j > k$ of ($\mathrm{M}_\exists$) and let $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ and $h : \mathsf{vars}(B) \to \mathbf{X}$ with $h(B) \in \tau$, be the TGD and the substitution, respectively, chosen at that step. Let $fv(\sigma) = \mathsf{vars}(B) \cap \mathsf{vars}(H)$. Towards a contradiction, assume there exists a run r that follows str and $tail(r) = \tau$. By assumption and definition of non-losing str, then there must exist a type $\tau' = str(\tau, \sigma, h)$ s.t. $\tau' \in \mathsf{LC}(\Sigma, D_c)$, and it satisfies conditions (C1) and (C2) of the game. But since τ gets marked then for each $\tau_i \in types(\Sigma)$, either τ_i is marked at some earlier step in the algorithm **Mark**(Σ), and then by hypothesis this type cannot appear in a run, or τ_i doesn't satisfy conditions (C1) or (C2).

It follows that there exists no τ_i that can define $str(\tau, \sigma, h)$ which contradicts that str is a non-losing strategy.

For the "$\Leftarrow$" direction, let $good(\Sigma) \subseteq types(\Sigma)$ be the set of all types that are not marked by the algorithm $\mathbf{Mark}(\Sigma)$, that is $good(\Sigma) \subseteq \mathsf{LC}(\Sigma, D_c)$. By assumption the types realized in D_c are not marked. We can build a strategy str on D_c which maps all pairs of a type $\tau \in good(\Sigma)$ and each TGD $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ and substitution $h : \mathsf{vars}(B) \to \mathbf{X}$ with $h(B) \in \tau$ to an arbitrary type $\tau' \in good(\Sigma)$ such that (C1) and (C2) hold, that is $\tau|_{h(fv(\sigma))} = \tau'|_{h(fv(\sigma))}$, and there exist a substitution $h' : \mathsf{vars}(H) \to \mathbf{X}$ with $h(fv(\sigma)) = h'(fv(\sigma))$ such that $h'(H) \in \tau'$. Such construction of str is possible since if there would not exists a $\tau' = str(\tau, \sigma, h)$ then τ would be marked by the algorithm as some iteration of ($\mathrm{M}_\exists$). We claim that str is a non-losing strategy on D_c. Towards a contradiction, assume str is not a non-losing strategy on D_c, that is there exists a finite run r with $tail(r) = \tau$ that follows str, such that either $tail(r) \notin \mathsf{LC}(\Sigma, D_c)$ or there exists some TGD $\sigma \in \Sigma_\exists$ of the form $B \to \exists \vec{y}.H$ and a substitution $h : \mathsf{vars}(B) \to \mathbf{X}$ with $h(B) \in \tau$ and $str(\tau, \sigma, h)$ is not defined. By construction of str, $tail(r) \in good(\Sigma)$, hence it cannot be the case that $tail(r) \notin \mathsf{LC}(\Sigma, D_c)$, because then τ doesn't satisfy ($\mathrm{LC}_\forall$) and as a consequence τ would be marked by ($\mathrm{M}_\forall$) in the algorithm, which then contradicts our assumption that τ is not marked. For the other case, using similar reasoning as for the "$\Rightarrow$" direction, if $str(\tau, \sigma, h)$ is not defined then there exists some iteration of ($\mathrm{M}_\exists$) that would mark $tail(r)$, which again contradicts our assumption. Hence, str constructed as above is a non-losing strategy for Bob on D_c. ◀

A.2 Proofs of Section 4

Proof. (Proof of Proposition 9.) This proof is similar to the proof of Theorem 3 in [1]. Program P can be partitioned into programs P_1 and P_2 as follows:

- P_1 consists of all rules in **(I)** and **(II)**. P_1 is a positive disjunctive program with at most w variables in each rule.
- P_2 consists of the remaining rules, and is disjunction-free.

Note that P_2 does not define any relations used in P_1, i.e. none of the relation symbols of P_1 occurs in the head of a rule in P_2. Assume a set F of facts over the signature of P_1. Due to the above properties, the successful runs of the following non-deterministic procedure generate the set of all models of $P \cup F$:

(S1) Compute a minimal model I_1 of $P_1 \cup F$.

(S2) Compute the least model I_2 of $I_1 \cup P_2^{I_1}$. If I_2 does not exist due to a constraint violation, then return *failure*.

Since P_1 has at most w-variables in every rule, each minimal model I_1 of $P_1 \cup F$ is of polynomial size in the size of $P_1 \cup F$, and the set of all such models can be traversed in polynomial space. For a given I_1, performing step (S2) is feasible in (deterministic) exponential time, because $P_2^{I_1}$ is a ground disjunction-free positive program of exponential size. It follows that computing the answers to (P, q) for any given input database D over the schema $\mathsf{sch}(\Sigma)$ requires single deterministic exponential time and is coNP in data complexity. ◀

A.3 Proofs of Section 5

Proof. (Proof of Lemma 12.) To show the "$\Rightarrow$" direction, we consider an arbitrary run of the algorithm that starts at $N_0 = \{(\tau_c, \emptyset) \mid \tau_c \in M\}$ with $M = \mathsf{Marked}(\Sigma)$, and we let N_i be the set N after the i-th iteration of the algorithm. (Note that although different runs may generate different sequences of N_is, this is irrelevant for the proof, and the final $\mathsf{Horn}(\Sigma)$ is unique). We show that $(\tau_c, \tau_{ir}) \in N_i$ implies that $\tau_c \cup \tau'_{ir} \in M$ for each $\tau'_{ir} \subseteq \tau_{ir}$, by

induction on the minimal i such that $(\tau_c, \tau_{ir}) \in N_i$. The claim holds trivially for $i = 0$. Next, assume $(\tau'_c, \tau'_{ir}) \in N_{i+1}$ but $(\tau'_c, \tau'_{ir}) \notin N_i$. Then there are $\{(\tau_c, \tau_{ir}), (\tau'_c, \tau_{ir})\} \subseteq N_i$ with $\tau_c = \tau'_c \cup \{\boldsymbol{\alpha}\}$ for some $\boldsymbol{\alpha} \notin \tau_c$, and $\tau'_{ir} = \tau_{ir} \cup \{\boldsymbol{\alpha}\}$. By hypothesis $\tau'_c \in M$ and $\tau_c = \tau'_c \cup \{\boldsymbol{\alpha}\} \in M$. Moreover, for any subset $\tau''_{ir} \subseteq \tau'_{ir}$: if $\{\boldsymbol{\alpha}\} \notin \tau''_{ir}$ then $\tau''_{ir} \subseteq \tau_{ir}$ and by hypothesis $\tau'_c \cup \tau''_{ir} \in M$; otherwise if $\{\boldsymbol{\alpha}\} \in \tau''_{ir}$ then $\tau'_c \cup \{\boldsymbol{\alpha}\} = \tau_c$ and $\tau''_{ir} \setminus \{\boldsymbol{\alpha}\} \subseteq \tau_{ir}$ and again by hypothesis $\tau_c \cup (\tau''_{ir} \setminus \{\boldsymbol{\alpha}\}) \in M$.

We show the "$\Leftarrow$" direction by induction on the size of τ_{ir}. Let $M = \mathsf{Marked}(\Sigma)$ and $N = \mathsf{Horn}(\Sigma)$. For $\tau_{ir} = \emptyset$, by construction $(\tau_c, \emptyset) \in N$. Assume the claim holds for all abstract types with τ_{ir} of size $k \leq |\mathbf{A}| - |\tau_c| - 1$. We show it for $k+1$. Assume $\tau_c \cup \tau'_{ir} \in M$ for each $\tau'_{ir} \subseteq \tau_{ir}$ with $|\tau_{ir}| = k+1$. In particular, it must hold for an arbitrary $\tau'_{ir} \subseteq \tau_{ir}$ with $|\tau'_{ir}| = k$. Thus by hypothesis $(\tau_c, \tau'_{ir}) \in N$. Now, let $\{\boldsymbol{\alpha}\} = \tau_{ir} \setminus \tau'_{ir}$. By assumption $\tau_c \cup \{\boldsymbol{\alpha}\} \cup \tau''_{ir} \in M$ for each $\tau''_{ir} \subseteq \tau'_{ir}$. As $|\tau'_{ir}| = k$, again by hypothesis $(\tau_c \cup \{\boldsymbol{\alpha}\}, \tau'_{ir})$ must also be in the set N. But then $(\tau_c \cup \{\boldsymbol{\alpha}\}, \tau'_{ir}) \in N$ and $(\tau_c, \tau'_{ir}) \in N$ implies $(\tau_c, \tau'_{ir} \cup \{\boldsymbol{\alpha}\}) \in N$ by algorithm Horn, that is $(\tau_c, \tau_{ir}) \in N$. ◀

Let φ be a disjunction of atoms. In what follows, for an instance I, we write $I \models \varphi$ if some atom from φ is in I, and $I \not\models \varphi$ otherwise.

Proof. (Proof of Lemma 13.) Let $M = \mathsf{Marked}(\Sigma)$ and $N = \mathsf{Horn}(\Sigma)$. For convenience, we denote the atoms in τ_{ir} by $\boldsymbol{\alpha}_1^{ir}, \ldots, \boldsymbol{\alpha}_l^{ir}$, and the disjunction of the atoms in $\mathbf{A} \setminus (\tau_c \cup \tau_{ir})$ by $\mathbf{A}^H_\vee$. For the "$\Rightarrow$" direction, as $(\tau_c, \tau_{ir}) \in N$, we know from Lemma 12 that $\tau_c \cup \tau_{ir} \in M$, and by Lemma 10, $\Sigma \models \tau_c, \tau_{ir} \rightarrow \mathbf{A}^H_\vee$. To show that $\Sigma \models \tau_c \rightarrow \mathbf{A}^H_\vee$, we argue that one can always remove any of the atoms $\boldsymbol{\alpha}_i^{ir}$ with $1 \leq i \leq \ell$ from the rule and the new rule obtained will still be entailed by the theory. The claim is trivial for $l = 0$. Let $\boldsymbol{\alpha}_i^{ir}$ be an arbitrary atom in τ_{ir} and let $\tau'_{ir} = \tau_{ir} \setminus \{\boldsymbol{\alpha}_i^{ir}\}$. From Lemma 12, we know that $\tau_c \cup \tau'_{ir} \in M$, thus $\Sigma \models \tau_c, \tau'_{ir} \rightarrow \mathbf{A}^H_\vee \vee \boldsymbol{\alpha}_i^{ir}$ (*). But by assumption $\Sigma \models \tau_c, \tau'_{ir}, \boldsymbol{\alpha}_i^{ir} \rightarrow \mathbf{A}^H_\vee$ (**). Then one can eliminate the atom $\boldsymbol{\alpha}_i^{ir}$ and the resulting rule will is still entailed by the theory, that is $\Sigma \models \tau_c, \tau'_{ir} \rightarrow \mathbf{A}^H_\vee$. Assume towards a contradiction that there exists a model I of Σ, and a substitution h such that $h(\tau_c \cup \tau'_{ir}) \subseteq I$, but $I \not\models h(\mathbf{A}^H_\vee)$. From this and (*), it follows that $h(\boldsymbol{\alpha}_i^{ir}) \subseteq I$, but then by (**), $I \not\models h(\mathbf{A}^H_\vee(a))$, a contradiction. So $\Sigma \models \tau_c, \tau'_{ir} \rightarrow \mathbf{A}^H_\vee$.

For the "$\Leftarrow$" direction, $\Sigma \models \tau_c \rightarrow \mathbf{A}^H_\vee$ implies $\Sigma \models \tau_c \rightarrow \tau_{ir} \vee \mathbf{A}^H_\vee$. But then by Lemma 10, $\tau_c \in M$. Now, it easy to see that for every $\tau'_{ir} \subseteq \tau_{ir}$, we have $\Sigma \models \tau_c, \tau'_{ir} \rightarrow \mathbf{A}^H_\vee \vee \tau''_\vee$ (***), where $\tau''_\vee$ is the disjunction of all atoms in $\tau_{ir} \setminus \tau'_{ir}$. Indeed, if this were not the case, then there would exist a model I of Σ and a substitution h such that $h(\tau_c, \tau'_{ir}) \subseteq I$ and $I \not\models h(\mathbf{A}^H_\vee \vee \tau''_\vee)$. But then $I \not\models h(\mathbf{A}^H_\vee)$, which contradicts our assumption. Now, (***) implies $\tau_c \cup \tau'_{ir}$ is a marked type for each $\tau'_{ir} \subseteq \tau_{ir}$, and thus by Lemma 12, $(\tau_c, \tau_{ir}) \in N$ follows. ◀

Proof. (Proof of Lemma 14.) Let $|\mathbf{A}| = n$. First, notice that $|\tau_c| \geq n - 1$, implies that $(\tau_c, \emptyset) \in M$ is the desired abstract type. Now, assume $|\tau_c| < n - 1$. Then by Lemma 10, $\Sigma \models \tau_c \rightarrow \boldsymbol{\alpha}_1 \vee \ldots \vee \boldsymbol{\alpha}_\ell$ with $\{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n\} = \mathbf{A} \setminus \tau_c$. Since Σ is a set of (non-disjunctive) TGDs, by the property of convexity there exists $i \in \{1, \ldots, \ell\}$ such that $\Sigma \models \tau_c \rightarrow \boldsymbol{\alpha}_i$. Then, by Lemma 13, we have $(\tau_c, \{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_\ell\} \setminus \{\boldsymbol{\alpha}_i\}) \in N$, which proves the claim. ◀

Proof. (Proof of Theorem 15.) Let $M = \mathsf{Marked}(\Sigma)$ and $N = \mathsf{Horn}(\Sigma)$, and let Σ_M be the set of all $\mathit{rule}(\tau)$ with $\tau \in M$. By Lemma 10, it is sufficient to show that $\Sigma \models \Sigma_M$ if and only if $\Sigma \models \Sigma_{\mathsf{Horn}}$. For the "$\Rightarrow$" direction, it suffices to show that for every rule in Σ_M there exists a more restricted rule in Σ_{Horn} that is also entailed by the theory. To this end, let $\tau_c \rightarrow \mathbf{A}^H_\vee$ be an arbitrary rule in Σ_M, where $\tau_c \in M$ and $\mathbf{A}^H_\vee = \bigvee_{\boldsymbol{\alpha} \in \mathbf{A} \setminus \tau_c} \boldsymbol{\alpha}$. By Lemma 14 there exists an abstract type (τ_c, τ_{ir}) in M with $|\tau| + |\tau_{ir}| \geq |\mathbf{A}| - 1$. Hence $\tau_c \rightarrow \boldsymbol{\alpha} \in \Sigma_{\mathsf{Horn}}$ for $\{\boldsymbol{\alpha}\} = \mathbf{A} \setminus (\tau \cup \tau_{ir})$, and by Lemma 13, $\Sigma \models \tau_c \rightarrow \boldsymbol{\alpha}$. For the "$\Leftarrow$" direction, it is easy to see

that for every rule in Σ_{Horn} there exists a more general rule in Σ_M. That is, for an arbitrary $\tau_c \to \mathbf{A}^H_\vee$ in Σ_{Horn} with $\mathbf{A}^H_\vee$ a disjunction over the set of atoms $\mathbf{A}^H$, by Lemma 13, we have $(\tau_c, \tau_{ir}) \in N$ for $\tau_{ir} = \mathbf{A} \setminus (\tau_c \cup \mathbf{A}^H)$. Then by Lemma 12, $\tau_c \in M$, hence $\tau_c \to \mathbf{A}^H_\vee \vee \tau''_\vee$ is in Σ_M, where $\tau''_\vee$ is the disjunction of the atoms in τ_{ir}. ◀