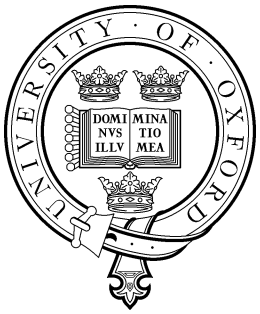
The Impact of Disjunction on Reasoning under Existential Rules



Michael Morak
St Anne's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2014

To my parents,

who have been giving me an unending amount of support.

Thank you.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Georg for the support of my DPhil studies, for his motivation, fruitful collaborations, and for sound advice whenever I needed it. I would also like to thank my friend, colleague, and inofficial advisor Andreas for his support and excellent guidance on conducting my DPhil studies and research activities.

Further, I would like give my thanks to my examiners, Thomas and Carsten, for their insightful comments, encouragement, fruitful discussions, and critical questions, which helped to improve this thesis substantially and spark new research interests and ideas.

My sincere thanks, together with Andreas, also go to my colleagues Pierre and Marco for many stimulating discussions, hard research work, sleepless nights before deadlines, and all the other good times we had. Also, thank you Lei, Helen, and Victor, for all the fun we had in the attic.

For their support throughout my time in Oxford I would like to sincerely thank my parents Brunhilde and Josef, my brother Niklas, and my partner Carina. Thanks for always being there for me.

Finally, I would like to thank the University of Oxford in general and the Department of Computer Science in particular, for providing me a wonderful working environment, and, together with the Engineering and Physical Sciences Research Council, Mrs Jane Ledig-Rowohlt (née Scatcherd) and the Austrian Academy of Sciences for their generous financial support, without which it would not have been possible to conduct the research for this thesis.

Statement of Originality

I hereby declare that no part of this thesis has been accepted or is currently being submitted for any degree, diploma or certificate or other qualification in the University of Oxford or elsewhere. I declare that this thesis is wholly my own work, that I have cited all source materials and that I have clearly marked all parts of the thesis which have been quoted from other works.

SIGNATURINFORMATIONEN	**Unterzeichner**	Michael Morak
	Datum/Zeit-UTC	2015-11-03T11:48:14+01:00
	Prüfinformation	Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at
Hinweis	Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument ist gemäß § 4 Abs. 1 Signaturgesetz einem handschriftlich unterschriebenen Dokument rechtlich gleichgestellt.	

Michael Morak

Abstract

Ontological database management systems are a powerful tool that combine traditional database techniques with ontological reasoning methods. In this setting, a classical extensional database is enriched with an ontology, or a set of logical assertions, that describe how new, intensional knowledge can be derived from the extensional data. Conjunctive queries are therefore answered against this combined *knowledge base* of extensional and intensional data.

Many languages that represent ontologies have been introduced in the literature. In this thesis we will focus on existential rules (also called tuple-generating dependencies or Datalog$^{\pm}$ rules), and three established languages in this area, namely guarded-based rules, sticky rules and weakly-acyclic rules. The main goal of the thesis is to enrich these languages with non-deterministic constructs (i.e. disjunctions) and investigate the complexity of the answering conjunctive queries under these extended languages. As is common in the literature, we will distinguish between *combined complexity*, where the database, the ontology and the query are considered as input, and *data complexity*, where only the database is considered as input. The latter case is relevant in practice, as usually the ontology and the query can be considered as fixed, and are usually much smaller than the database itself.

After giving appropriate definitions to extend the considered languages to disjunctive existential rules, we establish a series of complexity results, completing the complexity picture for each of the above languages, and four different query languages: arbitrary conjunctive queries, bounded (hyper-)treewidth queries, acyclic queries and atomic queries. For the guarded-based languages, we show a strong 2ExpTime lower bound for general queries that holds even for fixed ontologies, and extablishes 2ExpTime-completeness of the query answering problem in this case. For acyclic queries, the complexity can be reduced to ExpTime, if the predicate arity is bounded, and the problem even becomes tractable for certain restricted languages, if only atomic queries are used. For ontologies represented by sticky disjunctive rules, we show that the problem becomes undecidable, even in the case of data complexity and atomic queries. Finally, for weakly-acyclic rules, we show that the complexity increases from 2ExpTime to co-N2ExpTime in general, and from tractable to co-NP in case of the data complexity, independent of which query language is used.

After answering the open complexity questions, we investigate applications and relevant consequences of our results for description logics and give two generic complexity statements, respectively, for acyclic and general conjunctive query answering over description logic knowledge bases. These generic results allow for an easy determination of the complexity of this reasoning task, based on the expressivity of the considered description logic.

Contents

Contents i

List of Figures v

List of Tables vii

1 Introduction 1

1.1 Research Challenges . . . 2
1.2 Research Goals . . . 4
1.3 Summary of Contributions . . . 5
1.3.1 Query Answering under Guarded-based Classes of DTGDs . . . 5
1.3.2 Query Answering under Sticky Sets of DTGDs . . . 8
1.3.3 Query Answering under Weakly-Acyclic Sets of DTGDs . . . 9
1.3.4 Applications to Description Logics . . . 10
1.4 Road Map . . . 10

2 Preliminaries 13

2.1 Technical Definitions . . . 13
2.1.1 Trees, Hypertrees and Decompositions . . . 13
2.1.2 Alphabets . . . 14
2.1.3 The Relational Model . . . 15
2.1.4 Substitutions and Homomorphisms . . . 16
2.1.5 (Boolean) Conjunctive Queries . . . 16
2.1.6 Disjunctive Tuple-generating Dependencies . . . 17
2.2 Query Answering under Sets of DTGDs . . . 17
2.2.1 Normal Form of DTGDs . . . 19
2.3 The Disjunctive Chase . . . 20
2.4 Complexity Theory . . . 21
2.5 Alternating Turing Machines . . . 22
2.6 Notational Conventions . . . 22

3 Decidability in the Deterministic World **23**
3.1 Guarded-based Classes of TGDs . . . 23
3.1.1 Inclusion Dependencies . . . 24
3.1.2 Linear TGDs . . . 25
3.1.3 Guarded TGDs . . . 27
3.1.4 Frontier-Guarded TGDs . . . 28
3.1.5 Weakly-(Frontier-)Guarded Sets of TGDs . . . 30
3.1.6 Summary . . . 32
3.2 Acyclicity-based Classes of TGDs . . . 33
3.2.1 Full TGDs . . . 34
3.2.2 Weakly-Acyclic Sets of TGDs . . . 34
3.2.3 Summary . . . 36
3.3 Sticky-based Classes of TGDs . . . 37
3.3.1 Sticky Sets of TGDs . . . 38
3.3.2 Weakly-Sticky Sets of TGDs . . . 40
3.3.3 Summary . . . 41
3.4 Further Discussion . . . 42

4 Guarded-based Classes of DTGDs **45**
4.1 Extending Guarded-based Sets of TGDs with Disjunction . . . 46
4.2 Query Answering under Guarded-based Sets of DTGDs . . . 47
4.3 Guarded Fragments of First-Order Logic . . . 50
4.3.1 The Guarded Fragment of First-Order Logic . . . 50
4.3.2 Guarded Negation First-Order Logic . . . 51
4.4 Decidability . . . 52
4.5 Answering Arbitrary Queries . . . 52
4.5.1 Overview . . . 53
4.5.2 Upper Bounds . . . 55
4.5.3 Lower Bounds . . . 58
4.6 Answering Bounded (Hyper-)Treewidth Queries . . . 71
4.7 Answering Acyclic Queries . . . 78
4.7.1 Overview . . . 78
4.7.2 Upper Bounds . . . 80
4.7.3 Lower Bounds . . . 82
4.8 Answering Atomic Queries . . . 91
4.8.1 Overview . . . 91
4.8.2 Upper Bounds . . . 93
4.8.3 Lower Bounds . . . 102
4.9 Summary . . . 108

5 Sticky Sets of DTGDs **111**
5.1 Extending Sticky Sets of TGDs with Disjunction . . . 111
5.2 Query Answering under Sticky Sets of DTGDs . . . 112
5.3 Undecidability of Query Answering . . . 115
5.4 Summary . . . 120

6 Weakly-Acyclic Sets of DTGDs **123**
6.1 Extending Weakly-Acyclic Sets of TGDs with Disjunction . . . 123
6.2 Query Answering under Weakly-Acyclic Sets of DTGDs . . . 125
6.3 Complexity Overview . . . 126
6.4 Upper Bounds . . . 128
6.5 Lower Bounds . . . 130
6.6 Summary . . . 135

7 Applications to Description Logics **137**
7.1 Additional Features . . . 137
7.1.1 Negative Constraints . . . 137
7.1.2 Equality-Generating Dependencies . . . 139
7.2 Querying Description Logic Knowledge Bases . . . 142
7.2.1 The Description Logic $\mathcal{ALCIH}$ . . . 143
7.2.2 Generic Complexity Results . . . 144
7.2.3 A Brief Case Study . . . 147

8 Conclusions **149**
8.1 Overview . . . 149
8.2 Brief Discussion of Results . . . 150
8.3 Directions for Future Research . . . 151

Bibliography **155**

Nomenclature **165**

List of Figures

2.1 (a) The hypergraph $\mathcal{H}(I)$; (b) the Gaifman graph $G_{\mathcal{H}(I)}$; and (c) a tree decomposition of $G_{\mathcal{H}(I)}$. 15

3.1 Expressivity comparison of guarded-based classes of TGDs. 33
3.2 Dependency graph for Example 3.14 35
3.3 Expressivity comparison of guarded- and acyclicity-based classes of TGDs. . . . 37
3.4 Dependency graph for Example 3.20 40
3.5 Expressivity comparison of decidable classes of TGDs. 42

4.1 A part of an encoded configuration tree. 60
4.2 Gadgets for the proof of Theorem 4.14. (a) A skeleton note which is the left child of its parent, with its two skeleton node children and its navigation gadget; (b) A cell gadget encoding configuration cell 2 (assuming that $k = 4$), where configuration cells 2 and 4 are not valid successors of the configuration cell 2. 60
4.3 Adapted gadgets for the proof of Proposition 4.15. 63
4.4 (a) A ternary ordered tree T; (b) The binary version T' of T in left child-right sibling representation; (c) The conversion of T' into a full binary tree by adding the "dummy" nodes Δ. 65
4.5 Representation of the computation tree in the proof of Theorem 4.20. 73
4.6 Computation tree. 84
4.7 The UCQ Q in the proof of Theorem 4.23 is acyclic. 87
4.8 Possible proof-trees from $r(a, b)$ w.r.t. Σ. 94
4.9 Possible trees of Example 4.31. 96
4.10 Rooted tree obtained from the disjoint union of $T_1, \ldots, T_n$ after merging the root nodes. 97
4.11 The alternating algorithm SearchPT. 99
4.12 Computation of the algorithm SearchPT. 100

List of Tables

3.1 The complexity of CQ-Answering under guarded-based classes of TGDs. 33
3.2 The complexity of CQ-Answering under acyclicity-based classes of TGDs. . . . 37
3.3 The complexity of CQ-Answering under sticky-based classes of TGDs. 42

4.1 The complexity of (U)CQ-Answering under guarded-based classes of DTGDs. 53
4.2 The complexity of (U)ACQ-Answering under guarded-based classes of DTGDs. 78
4.3 The complexity of (U)CQ_1-Answering under guarded-based classes of DTGDs. 91

6.1 The complexity of (U)CQ-Answering under weakly-acyclic sets of DTGDs. . . 126

7.1 Normal form and first-order representation of $\mathcal{ALCIH}$. 146

1 Introduction

In recent years, significant interest has arisen in extending traditional databases with ontological reasoning capabilities. These systems, often referred to as *ontological database management systems (ODBMS)*, are thus able to perform classical database operations, like query answering, but in addition take into account a set of rules expressing knowledge about the world[1]. Both the *description logic (DL)* [Baader et al., 2003] community and the database [Abiteboul et al., 1995] community have invested great effort into investigating such systems. Indeed, ODBMS are seen as a commercial opportunity, and major database software suppliers have begun to integrate ontological reasoning capabilities into their systems[2]. This is not surprising, as in the business world most raw data already resides in such enterprise-grade database management systems. To enable ontological reasoning engines to access such data creates space for innovation and thus opens up new business opportunities via, for example, the Semantic Web.

Also smaller, more targeted commercial ODBMS have been developed. For example, DataGrid, Inc.[3] develops an ODBMS called OWL DBMS for the Semantic Web that implements relevant ontological reasoning tasks. Semafora[4] (formerly Ontoprise) develops an inference engine for ontologies called OntoBroker that supports all relevant W3C Semantic Web recommendations and standards. In addition, many research-based systems for ontological reasoning tasks over databases have been developed. To name a few, QuOnto [Acciarri et al., 2005] and its successor MASTRO [Calvanese et al., 2011] are based on the DL-Lite family of description logics [Calvanese et al., 2007] and have interfaces to commercial database systems like Oracle's Database Server, IBM's DB2 and Microsoft's SQL Server; FaCT++ [Tsarkov and Horrocks, 2006] is a reasoner for the standardized web ontology language OWL; and DLV [Leone et al., 2006], a reasoner for answer set programming (see, e.g. Eiter et al. [2009] for a survey), was extended to allow for ontological reasoning in the system OntoDLV [Ricca et al., 2009], as well as DLV$^{\exists}$ [Leone et al., 2012]. For a more complete list of research-based reasoners, we refer the reader to a recent report on the OWL Reasoner Evaluation competition results; see, e.g. Gonçalves et al. [2013].

In ontological database management systems, a relational database D (also called an *assertional box*, or *ABox*, in the description logic field) containing extensional knowledge is com-

[1] Here, by "world" we mean the portion of the real world that the database tries to model.

[2] See, e.g. Oracle Semantic Technologies (`http://download.oracle.com/otndocs/tech/semantic_web/pdf/oradb_semantic_overview.pdf`).

[3] `http://dt123.com/DataGrid/DataGridWebsiteV1a/`

[4] `http://www.semafora-systems.com/en/products/ontobroker/`

bined with an ontological theory Σ (also called a *terminological box*, or *TBox*, in description logic parlance) that contains rules and constraints that describe how to derive new, intensional knowledge from the extensional data in the database. Thus, a query q is no longer answered just against the database D, but against the logical theory $D \cup \Sigma$ that describes all the extensional data in D plus any intensional knowledge that can be derived from D by means of the rules and constraints in Σ. More formally, instead of checking whether the query is entailed by the database alone (i.e. whether $D \models q$, like in the classical database setting), one now has to check whether the combined logical theory $D \cup \Sigma$ entails the query, that is, whether $D \cup \Sigma \models q$. Note that ontological query answering can be seen as the problem of answering queries over incomplete databases vis-a-vis a set of constraints; see, e.g. Calì et al. [2003a].

In the field of description logics, entailment ($\models$) adheres to the standard logical definition (i.e. first-order logic entailment), that is, entailment under arbitrary, possibly infinite, models. We will thus adopt the same definition of entailment in this thesis when we deal with ontological theories expressed by existential rules. Such rule-based languages have a prominent presence in the areas of artificial intelligence and databases. A noticeable formalism, originally intended for expressing complex queries over relational databases, which is at the basis of rule-based languages for the specification of ontological constraints over data and knowledge bases, is Datalog, that is, function-free first-order Horn logic. For a thorough overview and introduction, we refer the reader, for example, to [Abiteboul et al., 1995; Ceri et al., 1990].

Unfortunately, Datalog alone is not expressive enough to formulate meaningful ontological theories, mainly due to the inability of plain Datalog to infer the existence of new objects which are not already in the extensional database; a deficiency thoroughly discussed by Patel-Schneider and Horrocks [2007]. Thus, strong interest in enhancing Datalog with existential quantification in rule-heads emerged in recent years, as documented by a number of relevant publications; see, e.g. Baget et al. [2011a,b]; Krötzsch and Rudolph [2011]; Thomazo et al. [2012]; Calì et al. [2012]. The obtained rules are known under a variety of names such as *existential rules*, *tuple-generating dependencies (TGDs)*, and *Datalog$^{\pm}$ rules*, and are structurally similar to Datalog with *value invention*; see, e.g. Cabibbo [1998]; Mailharro [1998]; Faltings and Macho-Gonzalez [2005].

1.1 Research Challenges

Unfortunately, the addition of existential quantification as explained above can in general easily lead to undecidability of the main reasoning tasks, and in particular of conjunctive query answering; see, e.g. Beeri and Vardi [1981], and Baget et al. [2011a]; Calì et al. [2013] for very tight undecidable cases. This is due to the fact that the relevant models of the logical theory may be infinite and therefore not explicitly computable. The following example shows this fact:

Example 1.1. *Consider a database* $D = \{person(john)\}$ *which contains a single fact stating that John is a person. Let the ontology* Σ *contain the following rules:*

- $\forall X\, person(X) \rightarrow \exists Y\, father(Y, X)$, *and*
- $\forall X \forall Y\, father(X, Y) \rightarrow person(X)$,

stating that every person has a father and every father is a person. It is easy to see that there is a model M *(in fact the most "general" model) that is infinite:*

$$M = D \cup \{father(z_1, john), person(z_1), father(z_2, z_1), \ldots\},$$

where for each $i > 0$, z_i *is a (labelled) null value, or a placeholder.*

Clearly, the boolean conjunctive query $q : \exists X \exists Y \exists Z\, father(X, Y) \wedge father(Y, Z)$ *is true under* M*, and in fact under every model of* $D \cup \Sigma$*, and we thus have that* $D \cup \Sigma \models q$*; on the other hand,* $D \not\models q$*. Note that the two rules in* Σ *are actually inclusion dependencies (IDs); see e.g. Abiteboul et al. [1995].*

Algorithms to deal with this type of infinity when answering queries were first developed in a database context by Johnson and Klug [1984] for the restricted case of *inclusion dependencies*. In the literature, several other, more expressive, concrete languages which guarantee decidability have been proposed; see, e.g. Fagin et al. [2005]; Calì et al. [2013]; Baget et al. [2011a]; Krötzsch and Rudolph [2011]; Calì et al. [2011, 2012]; Leone et al. [2012]. The following list contains a short survey of three such restrictions on TGDs, forming the respective decidable languages:

Guarded(-based) TGDs [Calì et al., 2013, 2012] Based on the guarded fragment of first-order logic [Andréka et al., 1998], the idea of guarded TGDs is to restrict the bodies of TGDs (i.e. the left-hand side of the implication) to be *guarded*, that is, they need to have a guard atom that contains all universally quantified variables. Several further restrictions and relaxations of guardedness have been proposed, notably frontier-guardedness, introduced by Baget et al. [2011a], which only requires variables that are propagated to be guarded. Guardedness forces the existence of tree-like models, on which the common reasoning tasks become decidable. Guardedness is a well-accepted paradigm, giving rise to robust languages that capture important lightweight description logics such as members of the DL-Lite family [Artale et al., 2009] and $\mathcal{EL}$ [Baader, 2003].

Sticky TGDs [Calì et al., 2012b] This syntactic property requires that values that participate in a join are always propagated. Being a global syntactic property, the structure of a set of TGDs has to be analyzed in order to verify that this is indeed the case, and a value that has participated in a join can indeed never be lost again. As this property roughly speaking limits the number of joins that can happen, it guarantees decidability.

Weakly-Acyclic TGDs [Fagin et al., 2005] This syntactic property was introduced in the context of *data exchange*, where a given source database is restructured and translated into a target database such that all given constraints are satisfied. With the need to actually materialize the target database, a restriction of TGDs had to be found that would guarantee that such a target database would always be finite. Weak acyclicity guarantees just this property. In our case, this means that for the main reasoning tasks, we can restrict our attention to finite models, and hence we get decidability. A more general class of *super-weakly-acyclic TGDs* was proposed by Marnette [2009].

The above decidable languages will be discussed in-depth in Chapter 3.

1.2 Research Goals

Unfortunately, none of the above languages—not even TGDs in general—are powerful enough for nondeterministic reasoning. The following example demonstrates this point:

Example 1.2. *Consider the following set of rules:*

$$\begin{array}{lcl} \sigma_1 & : & parentOf(X,Y), father(Y) \rightarrow \exists Z\, gparentOf(X,Z) \\ \sigma_2 & : & gparentOf(X,Y) \rightarrow child(Y) \\ \sigma_3 & : & child(X) \rightarrow boy(X) \vee girl(X) \end{array}$$

which expresses a simple and natural statement: "a parent of a father is a grandparent of some child, which in turn is a boy or *a girl." Although σ_1 and σ_2 are TGDs, σ_3 is not, due to the presence of disjunction in the head.*

Obviously, to be able to represent such kind of disjunctive knowledge, we need to enrich the existing classes of TGDs with disjunction in the head, or, equivalently, to consider *disjunctive TGDs (DTGDs)*, which were originally introduced by Deutsch and Tannen [2003] as the more powerful concept of disjunctive embedded dependencies. An extension of plain Datalog without existential quantifiers (a.k.a. *full TGDs*) with disjunction, called *disjunctive Datalog*, has been studied by Eiter et al. [1997]. More recently, special cases of the problem of query answering under guarded-based DTGDs were investigated by Alviano et al. [2012].

It is the aim of this thesis to extend the concrete languages discussed in the previous section to allow for DTGDs and investigate what the impact of allowing disjunction is in terms of the decidability and computational complexity of query answering under the these formalisms. Reviewing the literature shows that the picture of the computational complexity of this problem is still quite foggy, and there are several challenging issues to be tackled.

In the present work, we thus concentrate on the following fundamental questions: What is the exact complexity of conjunctive query (CQ) answering under weakly-acyclic DTGDs [Fagin et al., 2005], sticky-based classes of DTGDs [Calì et al., 2011] and (weakly-)(frontier-)guarded DTGDs [Calì et al., 2013; Baget et al., 2011a], as well as their main subclasses, that is, linear DTGDs and disjunctive IDs (DIDs) [Calì et al., 2012]? How is it affected if we consider a signature

of bounded arity, or a fixed set of dependencies? Moreover, how is it affected if we pose the queries in a more expressive query language, in particular using unions of CQs (UCQs), or in less expressive query languages like bounded (hyper-)treewidth queries (B(H)TWQs), acyclic queries (ACQs) or even atomic queries (CQ_1)?

As we shall see, the addition of disjunction has a significant effect on the complexity of query answering. We establish a surprisingly strong lower bound for guarded DTGDs that helps fill in the complexity picture in this case, where reasoning remains decidable. For sticky sets of DTGDs, we demonstrate undecidability, which is interesting given the fact that the complexity of reasoning under sticky sets of TGDs is lower than that under guarded TGDs. For weakly-acyclic sets of DTGDs, we also obtain the relevant complexity results which nicely reflect the change from deterministic to non-deterministic reasoning.

1.3 Summary of Contributions

This thesis includes, and significantly extends, results published in the following papers:

[Gottlob et al., 2012; Bourhis et al., 2013a,b, 2014a,b].

The results in this work can be summarized as follows. Note that each of the following subsections correspond to a main chapter of the thesis, excluding the preliminaries in Chapter 2, the background survey in Chapter 3 and the conclusions in Chapter 8.

1.3.1 Query Answering under Guarded-based Classes of DTGDs

The guarded-based TGD classes, namely inclusion dependencies, linear, multi-linear, guarded, and frontier-guarded TGDs as well as weakly-guarded and weakly-frontier-guarded sets of TGDs, are extended with disjunction in a straightforward manner, and we give the appropriate definitions that define the analogous classes of DTGDs, respectively. Also, we state an equivalence result which shows that answering unions of conjunctive queries is in fact equivalent to answering a single conjunctive query. However, the reduction from unions of CQs to CQs comes at the cost of increasing the maximum predicate arity of the underlying schema by one.

After this preliminary investigation, the complexity of answering conjunctive queries is investigated. Firstly, the complexity of answering arbitrary (unions of) conjunctive queries is pinpointed. The following list gives an overview of the corresponding results:

- We show that CQ answering for (weakly-)(frontier-)guarded DTGDs is complete for the complexity class 2ExpTime in the combined complexity; this also holds for UCQs. Regarding the data complexity, we show that under frontier-guarded DTGDs it is co-NP-complete, while for weakly-(frontier-)guarded it is ExpTime-complete. The relevant upper bounds are easily obtained by exploiting results on more expressive languages such as guarded negation first-order logic [Bárány et al., 2011], while the lower bounds are inherited from existing results on TGDs.

- We show that CQ answering under a *fixed* set of DIDs is 2ExpTime-hard, even if restricted to predicates of arity at most three. In case of UCQs, the above result holds even if we just consider unary and binary predicates. These surprisingly strong lower bounds are established by a reduction from an appropriate variant of the validity problem of CQs w.r.t. a non-deterministic tree automaton, based on a result by Björklund et al. [2008]. Roughly, a query is *valid* w.r.t. a non-deterministic tree automaton if every tree accepted by the automaton—when viewed as a database—entails the query. Taking the 2ExpTime upper bound discussed above into account, this gives us the complete picture for the computational complexity of our problem.

After pinpointing the complexity of the query answering problem for arbitrary conjunctive queries, the problem of answering (unions of) bounded (hyper-)treewidth queries is investigated. It turns out that in fact the complexity of the query answering problem remains unchanged, even if only bounded (hyper-)treewidth queries are considered. In order to show this, a reduction from the non-acceptance problem of an alternating ExpSpace Turing machine is given for a fixed set of DIDs and a UCQ. Different from the above result for arbitrary CQs, we need an arity of five here. By virtue of this result, the upper bound for arbitrary conjunctive queries and the fact that, in the data complexity, all fixed queries trivially have bounded treewidth we establish the exact same complexity results for bounded (hyper-)treewidth queries that we do for arbitrary CQs.

The complexity of answering (unions of) acyclic queries is investigated next. In this case, we notice that the complexity of the query answering problem does indeed decrease—if only slightly—which indicates that restricting the query language to ACQs does indeed provide better worst-case performance guarantees. The following list contains a summary of the relevant results for acyclic queries:

- The first result is disappointing. We show that query answering under a set of DIDs remains 2ExpTime-complete in the general case, even for ACQs. While the upper bound can be obtained from the arbitrary query case, the lower bound is again established by simulating an alternating ExpSpace Turing machine. However, in this case, we need to exploit the fact that we can encode a number identifying a tape cell inside our predicates. This number, in binary format, is polynomial in size and we can thus address exponentially many tape cells.

- A more positive complexity bound is established for the case where the arity is fixed. Without the ability to encode polynomially many symbols inside a predicate, the complexity of query answering under all classes of DTGDs considered, except for the frontier-guarded classes, reduces to ExpTime-complete. The upper bound is established by giving a (straightforward) reduction to (un)satisfiability of guarded first-order logic sentences, which for theories of fixed arity is known to be in ExpTime; see Grädel [1999].

- For the case where the entire set of DTGDs is fixed, the complexity reduces to ExpTime-completeness for all classes of DTGDs considered. The upper bound again follows from a reduction to guarded first-order logic. The lower bound is established by simulating the behaviour of an alternating linear-space Turing machine, using a fixed set of DIDs.

When answering (unions of) acyclic queries, the data complexity remains unchanged compared to answering arbitrary queries.

Finally, we turn our attention to answering atomic queries under the guarded-based classes of DTGDs considered. In this case, the complexity of query answering reduces even further. The following list contains the relevant results:

- Answering atomic queries under linear DTGDs is shown to be feasible in ExpTime. This is done by exhibiting an alternating algorithm which uses at most polynomial space in each step. When a schema with predicates of bounded arity is considered, the algorithm only uses logarithmic space, yielding a PTime upper bound in this case. We show the corresponding PTime-completeness result by simulating an alternating LinSpace Turing machine using a set of DIDs of bounded arity.

- The problem of answering atomic queries under linear DTGDs is shown to fall into the highly parallelizable complexity class of AC_0—recall that this is the complexity class of recognizing words in languages defined by constant-depth Boolean circuits with AND and OR gates of unlimited fan-in; see, e.g. [Papadimitriou, 1994]. This is shown by establishing that the problem is first-order rewritable, that is, it can be reduced to the problem of answering a first-order query over a given database. For linear DTGDs this is possible because it follows from their definition that an atomic query can only be answered positively if there is a chain of DTGDs and head atoms connecting the query atom to a single database atom.

- By adapting our 2ExpTime lower bound proof for DIDs and acyclic queries to multi-linear DTGDs and atomic queries, we obtain that in the general case, answering atomic queries under multi-linear DTGDs is 2ExpTime-hard. Together with the 2ExpTime upper bound for arbitrary queries, we get that for all expressive classes of guarded-based sets of DTGDs, the complexity of query answering cannot be reduced even if we consider atomic queries, and remains 2ExpTime-complete.

- A further adaptation of the 2ExpTime lower bound from the previous bullet point, where we encode the tape cell number of the Turing machine in the predicate name instead of inside the predicate itself, thereby fixing the predicate arity, yields an ExpTime lower bound for the case of answering atomic queries under multi-linear DTGDs of fixed arity. This follows from the fact that the number of tape cells that can be encoded in this way reduces to a polynomial number instead of exponentially many, and we thus simulate an alternating PSpace Turing machine instead of an alternating ExpSpace one.

For atomic queries, the case where the set of DTGDs is fixed coincides with the case of data complexity. By using the results for arbitrary queries we establish that for the languages considered the data complexity remains the same as for arbitrary queries, except for linear DTGDs and DIDs where it is in AC_0.

In combination, the above results complete the complexity picture of answering (unions of) conjunctive queries under guarded-based classes of DTGDs, where the query can be either arbitrary, of bounded (hyper-)treewidth, acyclic or atomic, for the cases of data complexity, and where the set of DTGDs is unrestricted (combined complexity), of bounded arity, or fixed.

1.3.2 Query Answering under Sticky Sets of DTGDs

Firstly, the class of sticky sets of TGDs is extended with disjunction, and the relevant definitions are given to establish the class of sticky sets of DTGDs. As sticky sets of (D)TGDs allow for cross products in the theory, we can show that in fact the problem of answering (unions of) arbitrary conjunctive queries is equivalent to answering a single atomic query. This can be done by storing the cross product of the relevant relations appearing in the CQs, projecting out unused variables, and then asking for the resulting atom, including the relevant joins, in the (now atomic) query.

After establishing these preliminary results, the complexity of the query answering problem is investigated. It turns out that the combination of cross products and disjunction is a recipe for disaster (i.e. the problem becomes undecidable). In fact, we are able to show that even for schemas with a maximum arity of two, the query answering problem becomes undecidable. This is shown by simulating a deterministic Turing machine, and thereby providing a reduction from the halting problem. In fact, with sticky sets of DTGDs, it is possible to generate two infinite chains and then compute the cross product between them, yielding an infinite grid. The horizontal and vertical axis of this grid are used to represent the tape cells and computation steps of the Turing machine, respectively.

At the cost of increasing the maximum predicate arity to six, we then show that undecidability even holds in the case of data complexity, where both the set of DTGDs and the query is fixed. Together with the query equivalence stated above, we get that this holds even for atomic queries.

This is a rather surprising result, as query answering under sticky sets of TGDs is feasible in ExpTime, a lower complexity class than answering queries under guarded TGDs, and whereas for guarded TGDs, non-determinism does not seem to have any effect, at least in the combined complexity case, for sticky sets of TGDs non-determinism seems to be a strong cause of undecidability. In fact, the underlying reason for this is the combination of cross products (or joins) and disjunction.

1.3.3 Query Answering under Weakly-Acyclic Sets of DTGDs

The class of weakly-acyclic sets of TGDs is extended to DTGDs and relevant notions and definitions are given. The extension is straightforward inasmuch as a set of DTGDs is weakly-acyclic if and only if the set of TGDs obtained by replacing all the disjunctions by conjunctions is weakly-acyclic. The fact that weak acyclicity does not pose any direct conditions on the structure of the DTGDs allows for conjunctive queries to be encoded in rule bodies. This immediately implies that in general, the query answering problem for arbitrary queries is equivalent to answering atomic queries. However, this does not necessarily hold in the case where the set of DTGDs is fixed but the query is not, as the latter can then not be simply added to the former.

After establishing these preliminary results, the complexity of the query answering problem under the types of queries considered is investigated. The following list gives an overview of the relevant results:

- A co-N2ExpTime upper bound is established for the most general case of answering arbitrary UCQs under weakly-acyclic sets of DTGDs. To this end, a guess-and-check algorithm that runs in non-deterministic double-exponential time is proposed. A Π_2^{P} upper bound for answering unions of arbitrary CQs under weakly-acyclic sets of DTGDs is established by exhibiting a co-NP guess-and-check algorithm that makes use of an NP oracle to answer the query. Finally, a co-NP guess-and-check algorithm is proposed for the cases where either the query is of bounded (hyper-)treewidth, or fixed, yielding a co-NP upper bound for these cases.

- A co-N2ExpTime lower bound is established for the case of atomic query answering under a weakly-acyclic set of DTGDs of bounded arity. This is shown by a reduction from the problem of tiling a square of double-exponential size in a given number n. In fact, it is possible to generate double-exponentially many labelled null values in an order, using a weakly-acyclic set of TGDs. By exploiting this method, it is possible to build two such chains and compute their cross product, yielding a square grid of double-exponential size, representing the square that is to be tiled. Afterwards, with disjunctive rules, a tiling can be guessed and a query be constructed to verify that the tiling is indeed valid.

- A co-NP lower bound is established in the data complexity for atomic queries, by adapting a result by Calvanese et al. [2013], which holds for acyclic queries.

The above results, together with a Π_2^{P}-hardness result inherited from [Bárány et al., 2014], complete the complexity picture of query answering under weakly-acyclic sets of DTGDs for all types of queries considered in this work.

1.3.4 Applications to Description Logics

Additional features are introduced into the language of DTGDs, namely negative constraints (NCs) and equality-generating dependencies (EGDs). Negative constraints are TGDs where the head (or the right-hand side of the implication) contains only the logical constant *false* ($\bot$). We show that the addition of such negative constraints does not lead to an increase in complexity of the general query answering problem. EGDs are TGDs where the head consists of a single equality between two variables. The addition of these dependencies can easily lead to undecidability. However, an abstract condition called *separability* is proposed that ensures that query answering remains decidable.

After introducing these additional features, a reduction is given from the expressive description logic $\mathcal{ALCIH}$ to guarded DTGDs extended with negative constraints. This implies that query answering under $\mathcal{ALCIH}$, or any sub-formalism thereof, is feasible in 2ExpTime, and in ExpTime if the query is acyclic. Further, tight corresponding lower bounds are given in a generic way. By analyzing the construction of previous lower bounds obtained for guarded-based classes of DTGDs, we show that for any description logic that is a sub-formalism of $\mathcal{ALCIH}$ and powerful enough to express concept disjunction, limited existential quantification, role hierarchies and the role inverse, the query answering problem is complete for the complexity class 2ExpTime. Furthermore, for any description logic that is a sub-formalism of $\mathcal{ALCIH}$ and powerful enough to express concept disjunction, limited existential quantification, and the role inverse, answering unions of acyclic queries is ExpTime-complete, even if only a fixed set of inclusions is used (i.e. the TBox is fixed).

In a case study, tight complexity bounds for the query answering problem, based on the above generic complexity results, are then given for different, established description logics, including $\textit{DL-Lite}^{H}_{bool}$, $\textit{DL-Lite}^{F}_{bool}$, $\mathcal{ELU}$ and $\mathcal{ELUIH}$, as well as $\mathcal{ALCIH}$ itself.

1.4 Road Map

The remainder of the thesis is organized as follows. In Chapter 2, we present preliminary definitions and necessary background material. We recall some basics on trees, hypertrees and decompositions, as well as relational databases. We define (Boolean) conjunctive queries, (disjunctive) tuple-generating dependencies, the query answering problem under DTGDs, and the main technical tool that we make use of in the thesis: the *disjunctive chase*. Finally, we recall fundamental notions on alternation and alternating Turing machines. In Chapter 3, we review and survey various classes of TGDs that provide decidability guarantees of the query answering problem in the non-disjunctive case. The complexity picture of the query answering problem under these formalisms is summarized and an expressivity comparison is performed. In Chapter 4, guarded-based classes of TGDs are extended to DTGDs, and an extensive complexity analysis is performed, analyzing the cases of answering arbitrary, bounded (hyper-)treewidth, acyclic and atomic queries in the cases of the combined complexity, where

the maximum predicate arity is fixed, where the entire set of dependencies is fixed, and of the data complexity. The same is done for sticky sets of TGDs, which are extended to DTGDs and analyzed w.r.t. their complexity in Chapter 5. An undecidability result is given for this case, even for the data complexity and atomic queries. In Chapter 6, the class of weakly-acyclic sets of TGDs is extended to DTGDs, and an extensive complexity investigation is performed, in the same manner as for the guarded-based classes. Chapter 7 extends the language of DTGDs with some additional features, namely negative constraints and equality-generating dependencies, and looks at the consequences of the complexity results established in the previous chapters for the complexity of query answering under relevant description logics. Finally, we conclude in Chapter 8 with a discussion of the obtained results, as well as an overview of directions for future research and some open problems.

2 Preliminaries

In this chapter, we present background material necessary for this thesis. After giving some preliminary technical definitions in Section 2.1, we recall some basics on trees and tree decompositions, relational databases, (Boolean) conjunctive queries, as well as disjunctive tuple-generating dependencies (DTGDs), in order to define the central problem studied in this thesis: query answering under sets of DTGDs, which is discussed in Section 2.2. We also introduce the main technical tool that we are going to employ in our later proofs, namely the disjunctive chase procedure, which is discussed in Section 2.3, as well as some basics on alternating Turing machines in Section 2.5.

We assume that the reader is familiar with the fundamental concepts of deterministic and non-deterministic Turing machines. We will give a very brief survey the most basic notions of complexity theory, alternation, oracles, complexity classes, reductions and completeness. A more detailed exposition of all the complexity notions employed in this work can be found in standard textbooks; see, e.g. Papadimitriou [1994].

2.1 Technical Definitions

In this section, we will introduce the basic notions of trees and tree decompositions, relational databases, (Boolean) conjunctive queries and disjunctive tuple-generating dependencies. For further details on the second and third item, we refer the reader to [Abiteboul et al., 1995].

2.1.1 Trees, Hypertrees and Decompositions

In this work we will often use graphs to represent relevant structures. A *graph* is a pair $G = (V, E)$, where V is a set of vertices, and E is a binary relation of edges, with $E \subseteq V \times V$. A *hypergraph* is a pair $H = (V, E)$ with a set V of vertices and a set E of hyperedges. A hyperedge $e \in E$ is itself a set of vertices, with $e \subseteq V$.

Consider a hypergraph $\mathcal{H} = (V, E)$. The *GYO-reduct* of $\mathcal{H}$, denoted $GYO(\mathcal{H})$, is obtained by applying the following two steps exhaustively: (1) eliminate vertices that are contained in at most one hyperedge; and (2) eliminate hyperedges that are empty or contained in other hyperedges. We say that a hypergraph $\mathcal{H}$ is *acyclic* if $GYO(\mathcal{H}) = (\emptyset, \emptyset)$.

A *tree decomposition* [Robertson and Seymour, 1984] of a graph $G = (V, E)$ is a pair (T, χ), where T is a tree and χ maps each node t of T (we use $t \in T$ as a shorthand below) to a *bag* $\chi(t) \subseteq V$, such that

- for each $v \in V$, there is a $t \in T$, such that $v \in \chi(t)$;
- for each $(v, w) \in E$, there is a $t \in T$, such that $\{v, w\} \subseteq \chi(t)$; and
- for each $r, s, t \in T$, such that s lies on the path from r to t, $\chi(r) \cap \chi(t) \subseteq \chi(s)$.

The *width* of a tree decomposition is defined as the cardinality of its largest bag minus one. The *treewidth* of a graph is defined as the minimum width over all possible tree decompositions. It is a well-known notion that measures how similar a graph is to a tree. Note that a tree has a treewidth of 1.

A *(generalized) hypertree decomposition* [Gottlob et al., 2002, 2003] of a hypergraph $H = (V, E)$ is a triplet $HD = \langle T, \chi, \lambda \rangle$, where $T = (N, F)$ is a (rooted) tree and χ and λ are labelling functions such that for each node $n \in N$, $\chi(n) \subseteq V$ and $\lambda(n) \subseteq E$ and the following conditions hold:

- for every hyperedge $e \in E$ there exists a node $n \in N$ such that $e \subseteq \chi(n)$,
- for every vertex $v \in V$ the set $\{n \in N \mid v \in \chi(n)\}$ induces a connected subtree of T,
- for every node $n \in N$, $\chi(n) \subseteq \bigcup_{e \in \lambda(n)} e$

The *width* of a hypertree decomposition is the maximum λ-set size over all its nodes. For a graph G, the minimum width over all possible hypertree decompositions referred to as the *hypertree-width* of G. The *treewidth* for a hypergraph is defined as the maximum χ-set size of such a decomposition, minus one. The hypertree-width is a measure of how close to acyclic a hypergraph is, analogous to treewidth for graphs. It is well-known that a hypergraph is acyclic if and only if its hypertree-width is one. Moreover, the hypertree-width of a hypergraph is always less-or-equal than its treewidth.

Note that the treewidth of a hypergraph is equivalent to the treewidth of its Gaifman graph. The Gaifman graph $G = (V, E)$ of a hypergraph $H = (V', E')$ is obtained by letting $V = V'$, and for every hyperedge $e \in E'$, and every pair of nodes $(u, v) \in e \times e$, such that $u \neq v$, we add the edge (u, v) to E. Informally, we replace every hyperedge in H with a clique spanning its nodes.

2.1.2 Alphabets

Let **C**, **N** and **V** be pairwise disjoint, countably infinite sets. The elements of **C** are called *constants* and constitute the normal domain of a database, the elements of **N** are called *labeled nulls* and are used as placeholders for unknown values, and thus can be also seen as (globally) existentially quantified variables, and the elements of **V** are called (regular) variables, which are used in queries and dependencies. Different constants represent different values (*unique name assumption*), while different labelled nulls, or variables, may represent the same value. A fixed lexicographic order is assumed on $\mathbf{C} \cup \mathbf{N}$, such that every value in **N** follows all those in **C**. We denote by **X** sequences of not necessarily distinct variables $X_1, \ldots, X_k$, however, by slight

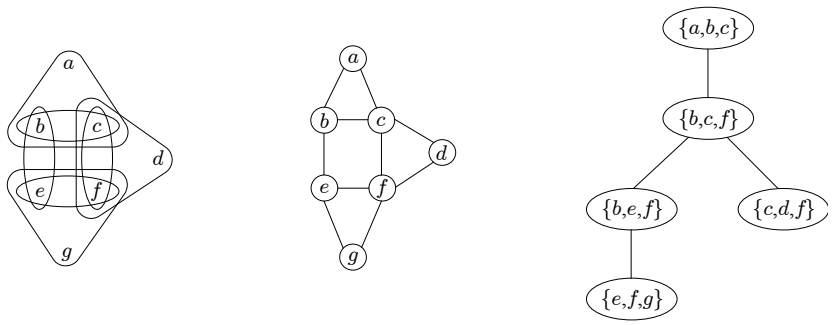


Figure 2.1: (a) The hypergraph $\mathcal{H}(I)$; (b) the Gaifman graph $G_{\mathcal{H}(I)}$; and (c) a tree decomposition of $G_{\mathcal{H}(I)}$.

abuse of notation, we will also sometimes treat $\mathbf{X}$ as a set. Throughout, let $[n] = \{1, \dots, n\}$, for every integer $n \geq 1$.

2.1.3 The Relational Model

A *relational schema* $\mathcal{R}$ (also simply called a *schema*) is a set of *relations* (also called *predicates*), each of which has an associated *arity*. The arity of a relation r is denoted $arity(r)$, and the maximum arity over all predicates of a schema $\mathcal{R}$ is denoted by $arity(\mathcal{R})$. A *position* in a schema $\mathcal{R}$ is a pair (r, i), where r is a relation in $\mathcal{R}$, and i a number such that $1 \leq i \leq arity(r)$. We will denote positions using the notation $r[i]$.

A *term* t is a constant ($t \in \mathbf{C}$), labeled null ($t \in \mathbf{N}$), or variable ($t \in \mathbf{V}$). An *atomic formula* (also simply called an *atom*) has the form $p(t_1, \dots, t_n)$, where p is an n-ary predicate, and $t_1, \dots, t_n$ are terms. For an atom $\underline{a}$, we denote as $dom(\underline{a})$ and $var(\underline{a})$ the set of terms and the set of variables occurring in $\underline{a}$, respectively. These notations naturally extend to sets of atoms. For convenience, we will treat conjunctions of atoms as sets of atoms. An *instance* I over a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $p(\mathbf{t})$, where $\mathbf{t}$ is a tuple of constants and labeled nulls, that is, $\mathbf{t} \subseteq (\mathbf{C} \cup \mathbf{N})^{|\mathbf{t}|}$. A *database* D is a finite instance that does not contain null values. Whenever an instance I is treated as a logical formula, in fact it is the formula $\exists \mathbf{X} \bigwedge_{\underline{a} \in I} \underline{a}$, where $\mathbf{X}$ contains a variable X_z for each labeled null z occurring in I.

Let us now define the treewidth of an instance. The hypergraph of an instance I, denote $\mathcal{H}(I)$, is a hypergraph (V, E), where $V = dom(I)$, and, for each $\underline{a} \in I$, there exists a hyperedge $e \in E$ such that $h = dom(\underline{a})$. The treewidth of an instance I, denoted $tw(I)$, is defined as the treewidth of its hypergraph $\mathcal{H}(I)$, that is, the treewidth of the Gaifman graph $G_{\mathcal{H}(I)}$ of $\mathcal{H}(I)$. Recall that the Gaifman graph of $\mathcal{H}(I)$ is the graph (V, E), where V is the node set of $\mathcal{H}(I)$, and $(v, u) \in E$ if and only if $\mathcal{H}(I)$ has a hyperedge h such that $\{v, u\} \subseteq h$.

Example 2.1 (Treewidth of an Instance)**.** *Consider the instance*

$$I = \{p(a,b,c), p(c,f,d), p(e,f,g), t(b,c), t(b,e), t(e,f), t(f,c)\},$$

where $\{a, b, c, d, e, f, g\} \subset \mathbf{C}$. *The hypergraph* $\mathcal{H}(I)$ *is depicted in Figure 2.1(a), its Gaifman graph* $G_{\mathcal{H}(I)}$ *in Figure 2.1(b), and a tree decomposition of* $G_{\mathcal{H}(I)}$ *in Figure 2.1(c). It is easy to verify that the given tree decomposition is optimal, and thus* $tw(I) = 2$.

2.1.4 Substitutions and Homomorphisms

A *substitution* from a set of symbols $S \subseteq \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ to a set of symbols $S' \subseteq \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ is a function $h : S \to S'$ defined as follows: $\emptyset$ is a substitution (the empty substitution), and if h is a substitution, then $h \cup \{s \mapsto s'\}$ is a substitution, where $s \in S$ and $s' \in S'$. If $s \mapsto s' \in h$, then we write $h(s) = s'$. The *restriction* of h to $T \subseteq S$, denoted $h_{|T}$, is the substitution $h' = \{t \mapsto h(t) \mid t \in T\}$. A *homomorphism* from a set of atoms A to a set of atoms A' is a substitution $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ such that: *(i)* if $t \in \mathbf{C}$, then $h(t) = t$; and *(ii)* if $p(t_1, \ldots, t_n) \in A$, then $h(p(t_1, \ldots, t_n)) = p(h(t_1), \ldots, h(t_n)) \in A'$. If there exists a homomorphism from A to A', and a homomorphism from A' to A, then we say that A and A' are *homomorphically equivalent*.

2.1.5 (Boolean) Conjunctive Queries

A *conjunctive query (CQ)* q is a positive existential first-order formula

$$\exists \mathbf{Y}\, \phi(\mathbf{X}, \mathbf{Y}),$$

where ϕ is a conjunction of atoms with variables from $\mathbf{X} \cup \mathbf{Y} \subset \mathbf{V}$, and possibly constants of $\mathbf{C}$. The arity of q is defined as the cardinality of $\mathbf{X}$, that is, the number of free variables occurring in $\exists \mathbf{Y}\, \phi(\mathbf{X}, \mathbf{Y})$. The tuple $\langle \mathbf{X} \rangle$ is called the *output tuple* of q. A 0-ary CQ is called *Boolean CQ (BCQ)*. The *answer* to a query $q = \exists \mathbf{Y}\, \phi(\mathbf{X}, \mathbf{Y})$ over an instance I, denoted $q(I)$ is defined as the following set: $q(I) = \{\langle \mathbf{t} \rangle \mid \mathbf{t} \subseteq \mathbf{C}^{|\mathbf{t}|} \wedge \exists$ homomorphism h, such that $h(\phi(\mathbf{X}, \mathbf{Y})) \subseteq I \wedge h(\mathbf{X}) = \mathbf{t}\}$. Note that if q is a BCQ, the empty tuple $\langle\rangle$ is the only possible answer. Formally, we say a BCQ q has a *positive answer* over an instance I, denoted $I \models q$, if and only if $\langle\rangle \in q(I)$, or, equivalently, $q(I) \neq \emptyset$. Otherwise, we say that q has a *negative answer* over I, denoted $I \not\models q$, or, equivalently, $q(I) = \emptyset$. An n-ary *union of conjunctive queries (UCQ)* is a disjunction of a finite number of n-ary CQs. By abuse of notation, we will, when it is convenient, consider a UCQ as set of CQs. The answer to an n-ary UCQ Q over an instance I, denoted $Q(I)$, is the set of n-tuples $\bigcup_{q \in Q} q(I)$. The answer to a union of BCQs over I is positive, written as $I \models Q$, if $Q(I) \neq \emptyset$.

Relevant Subclasses of CQs

Several subclasses of conjunctive queries have been considered in the literature, with the aim of reducing the complexity of the CQ evaluation problem. The simplest such fragment are *atomic queries*, denoted CQ_1, which are CQs that consist of a single atom only.

Another fragment is the class of *acyclic CQs (ACQs)*; see, e.g. [Chekuri and Rajaraman, 2000]. The acyclicity of a CQ is defined via the acyclicity of its hypergraph.

The hypergraph of a CQ q, denote $\mathcal{H}(q)$, is a hypergraph (V, E), where $V = dom(q)$, and, for each atom $\underline{a}$ in q, there exists a hyperedge $e \in E$ such that $e = dom(\underline{a})$. We say that q is *acyclic* if $\mathcal{H}(q)$ is acyclic.

Another important class are *CQs of bounded treewidth (BTWQs)*, introduced by Kolaitis and Vardi [2000]. The treewidth of a CQ q, denoted $tw(q)$, is defined as the treewidth of its hypergraph $\mathcal{H}(q)$.

It is easy to verify that acyclic CQs and CQs of bounded treewidth form incomparable query languages. For example, take a query whose hypergraph forms a cycle of n vertices. Clearly it is not acyclic, but, independent of n, its treewidth is always 2. Conversely, take, a query whose hypergraph has n vertices that form a clique, and a single edge containing all vertices. The treewidth of such a graph is $n - 1$, whereas it is acyclic. A class of conjunctive queries which is powerful enough to capture both acyclic queries and bounded treewidth queries are *CQs of bounded hypertree-width (BHTWQs)*; see, e.g. [Gottlob et al., 2002]. The hypertree-width of a query q is the hypertree-width of its associated hypergraph $\mathcal{H}(q)$.

2.1.6 Disjunctive Tuple-generating Dependencies

A *disjunctive tuple-generating dependency (DTGD)* σ is a first-order formula

$$\forall \mathbf{X} \left(\phi(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Y}_i \, \psi_i(\mathbf{X}, \mathbf{Y}_i) \right),$$

where $n \geq 1$, $\mathbf{X} \cup \mathbf{Y}_1 \cup \ldots \cup \mathbf{Y}_n \subset \mathbf{V}$, and $\phi, \psi_1, \ldots, \psi_n$ are conjunctions of atoms (possibly containing constants of $\mathbf{C}$). The formula ϕ is called the *body* of σ, denoted $body(\sigma)$, while $\bigvee_{i=1}^{n} \psi_i$ is the *head* of σ, denoted $head(\sigma)$. The set of variables $var(body(\sigma)) \cap var(head(\sigma)) \subseteq \mathbf{X}$, that is, the variables of $\mathbf{X}$ which appear both in the body and in the head of σ, is known as the *frontier* of σ, and is denoted as $fr(\sigma)$.

If $n = 1$, then σ is called *tuple-generating dependency (TGD)*. We will sometimes refer to sets of (D)TGDs as a *theory* or as a set of *(disjunctive) existential rules* or simply *rules*. The *schema* of a set Σ of DTGDs, denoted $sch(\Sigma)$, is the set of all predicates occurring in Σ. In the rest of the paper, for brevity, we will omit the universal quantifiers in front of DTGDs, and implicitly assume such a quantification. We will also use the comma (instead of $\wedge$) for conjoining atoms in the body and in the head of a DTGD. An instance I satisfies a DTGD σ, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(\phi(\mathbf{X})) \subseteq I$, then there exists $i \in [n]$ and $h' \supseteq h$ such that $h'(\psi_i(\mathbf{X}, \mathbf{Y}_i)) \subseteq I$; I satisfies a set Σ of DTGDs, denoted $I \models \Sigma$, if $I \models \sigma$, for each $\sigma \in \Sigma$. Whenever a set Σ of DTGDs is treated as a logical formula, in fact it is the formula $(\bigwedge_{\sigma \in \Sigma} \sigma)$.

2.2 Query Answering under Sets of DTGDs

Given a database D and a set Σ of DTGDs, the answers we consider are those that are true in *all* models of D with respect to Σ. Formally, the *models* of D with respect to Σ, denoted as

$models(D, \Sigma)$, is the set of all instances I such that $I \supseteq D$ and $I \models \Sigma$. The *answer* to an n-ary CQ q w.r.t. D and Σ is the set of n-tuples of constants

$$ans(q, D, \Sigma) = \bigcap_{I \in models(D,\Sigma)} q(I).$$

If q is Boolean, then its answer is *positive*, denoted $D \cup \Sigma \models q$, if $ans(q, D, \Sigma) \neq \emptyset$. The answer to a UCQ w.r.t. D and Σ is defined analogously. The standard version of the main decision problems tackled in this work are defined as follows:

TUPLE-CQ-ANSWERING

Instance: A database D, a set Σ of DTGDs, an n-ary CQ Q, and a tuple of terms $\mathbf{t} \in \mathbf{C}^n$.

Question: $\mathbf{t} \in ans(q, D, \Sigma)$?

The problem of TUPLE-UCQ-ANSWERING is defined analogously. It is well-known that CQ answering for arbitrary CQs can be easily reduced to CQ answering for BCQs by substituting the given tuple $\mathbf{t}$ into the CQ; thus, we can focus on BCQs. In the remainder of this thesis, by CQ and UCQ we refer to a BCQ and a union of BCQs, respectively. We can thus define the central decision problem that we study in this thesis, also referred to as the *query answering problem*, as follows:

CQ-ANSWERING

Instance: A database D, a set Σ of DTGDs and a CQ q.

Question: $D \cup \Sigma \models q$?

The UCQ-ANSWERING-problem is defined analogously. We also introduce versions of the query answering problem for each of the subclasses of CQs, which comprise the problem of (U)B(H)TWQ-ANSWERING for query answering under (unions of) bounded treewidth or bounded hypertree-width queries; the problem of (U)ACQ-ANSWERING for (unions of) acyclic queries; and finally the (U)CQ_1-ANSWERING problem for (unions of) atomic queries.

Following the taxonomy used by Vardi [1982], the *data complexity* of the above problems is calculated taking only the database as input. For the *combined complexity*, the query and set of DTGDs count as part of the input as well. We usually assume, without loss of generality, that both the database and the query use only predicates of $sch(\Sigma)$; let us give some more details. Consider a database D, a set Σ of DTGDs, and a CQ q. Let $D = D' \cup D''$, where $\{D', D''\}$ forms a partition of D and $D' = \{p(\mathbf{t}) \in D \mid p \in sch(\Sigma)\}$. Moreover, let $q' = \exists \mathbf{X}\, \phi(\mathbf{X}) \wedge \psi(\mathbf{X})$, where, for each predicate p occurring in $\phi(\mathbf{X})$ (resp. $\psi(\mathbf{X})$), $p \in sch(\Sigma)$ (resp. $p \notin sch(\Sigma)$). Clearly, if there exists a homomorphism h such that $h(\psi(\mathbf{X})) \subseteq D''$, then $D \cup \Sigma \models q$ if and only if $D' \cup \Sigma \models \exists \mathbf{X} h(\phi(\mathbf{X}))$; otherwise, $D \cup \Sigma \not\models q$. Notice that CQ answering is at least as hard as checking for the existence of h. The above argument can be straightforwardly extended to UCQs.

It is well known that query answering (even for atomic queries) is undecidable already when we only consider sets of (non-disjunctive) TGDs; this was shown by Beeri and Vardi [1981]. More recently, it has been shown that the query answering problem remains undecidable, even when the set of TGDs contains only a single TGD (see Baget et al. [2011a]), or when the set of TGDs is fixed (see Calì et al. [2013]). To guarantee decidability for the query answering problem under sets of TGDs, several restrictions exist in the literature. A thorough overview is provided in Chapter 3.

2.2.1 Normal Form of DTGDs

For query answering purposes, we can focus on DTGDs which are in normal form. A set Σ of DTGDs is in normal form if each $\sigma \in \Sigma$ is of the form

$$\phi(\mathbf{X}) \rightarrow p(\mathbf{X}) \qquad \text{or} \qquad \phi(\mathbf{X}) \rightarrow \exists Y\, p(\mathbf{X}, Y) \qquad \text{or} \qquad \phi(\mathbf{X}) \rightarrow p_1(\mathbf{X}) \vee p_2(\mathbf{X}).$$

The first one refers to existential-free single-atom-head TGDs, the second one to single-atom-head TGDs with exactly one occurrence of an existentially quantified variable, while the third one refers to existential-free DTGDs with a disjunction of exactly two atoms in the head. Every set Σ of DTGDs can be transformed in logarithmic space into a set $\mathsf{N}(\Sigma)$ in normal form such that, for every database D and UCQ Q, $D \cup \Sigma \models Q$ if and only if $D \cup \mathsf{N}(\Sigma) \models Q$. Let us explain how the normalization procedure works.

Consider a DTGD σ. First, we construct the set $\mathsf{N}_1(\sigma)$ by exhaustively applying the following two replacement rules, starting from σ, until each assertion is either a single-atom-head TGD, or a DTGD with a disjunction of two atoms in its head:

1. A DTGD of the form $\phi(\mathbf{X}) \rightarrow \vee_{i=1}^{n} \exists \mathbf{Y}_i\, \psi_i(\mathbf{X}, \mathbf{Y}_i)$ is replaced by

$$\begin{aligned} \phi(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_1\, p_1^\star(\mathbf{X}, \mathbf{Y}_1) \vee \exists \mathbf{Y}_{2..n}\, p_{2..n}^\star(\mathbf{X}, \mathbf{Y}_{2..n}) \\ p_1^\star(\mathbf{X}, \mathbf{Y}_1) &\rightarrow \psi_1(\mathbf{X}, \mathbf{Y}_1) \\ p_{2..n}^\star(\mathbf{X}, \mathbf{Y}_{2..n}) &\rightarrow \bigvee_{i=2}^{n} \psi_i(\mathbf{X}, \mathbf{Y}_i), \end{aligned}$$

   where $\mathbf{Y}_{2..n} = \cup_{i=2}^{n} \mathbf{Y}_i$, $p_1^\star$ is an $|\mathbf{X} \cup \mathbf{Y}_1|$-ary predicate and $p_{2..n}^\star$ is an $|\mathbf{X} \cup \mathbf{Y}_{2..n}|$-ary predicate; both $p_1^\star$ and $p_{2..n}^\star$ are auxiliary predicates not introduced so far.

2. A TGD of the form $\phi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, p_1(\mathbf{X}, \mathbf{Y}_1), \ldots, p_n(\mathbf{X}, \mathbf{Y}_n)$ is replaced by

$$\begin{aligned} \phi(\mathbf{X}) &\rightarrow \exists \mathbf{Y}\, p^\star(\mathbf{X}, \mathbf{Y}) \\ p^\star(\mathbf{X}, \mathbf{Y}) &\rightarrow p_1(\mathbf{X}, \mathbf{Y}_1) \\ &\vdots \\ p^\star(\mathbf{X}, \mathbf{Y}) &\rightarrow p_n(\mathbf{X}, \mathbf{Y}_n), \end{aligned}$$

   where $p^\star$ is an $|\mathbf{X} \cup \mathbf{Y}|$-ary auxiliary predicate not introduced so far.

Now, from $\mathsf{N}_1(\sigma)$, we obtain $\mathsf{N}(\sigma)$ by applying the following: for each $\sigma' \in \mathsf{N}_1(\sigma)$, assuming that $fr(\sigma') = \mathbf{X}$ and $\{Y_1, \ldots, Y_n\}$ are the existentially quantified variables of σ', σ' is replaced by

$$\begin{aligned}
body(\sigma') &\rightarrow \exists Y_1\, p_1^{\sigma'}(\mathbf{X}, Y_1) \\
p_1^{\sigma'}(\mathbf{X}, Y_1) &\rightarrow \exists Y_2\, p_2^{\sigma'}(\mathbf{X}, Y_1, Y_2) \\
p_2^{\sigma'}(\mathbf{X}, Y_1, Y_2) &\rightarrow \exists Y_3\, p_3^{\sigma'}(\mathbf{X}, Y_1, Y_2, Y_3) \\
&\vdots \\
p_{n-1}^{\sigma'}(\mathbf{X}, Y_1, \ldots, Y_{n-1}) &\rightarrow \exists Y_n\, p_n^{\sigma'}(\mathbf{X}, Y_1, \ldots, Y_n) \\
p_n^{\sigma'}(\mathbf{X}, Y_1, \ldots, Y_n) &\rightarrow head(\sigma),
\end{aligned}$$

where, for each $i \in [n]$, $p_i^{\sigma'}$ is an $(|\mathbf{X}| + i)$-ary auxiliary predicate not introduced so far. Finally, given a set Σ of DTGDs, $\mathsf{N}(\Sigma)$ is defined as $\bigcup_{\sigma \in \Sigma} \mathsf{N}(\sigma)$. The next lemma follows by construction:

Lemma 2.2. *Consider a set Σ of DTGDs. Then,*

1. $\mathsf{N}(\Sigma)$ *is in normal form;*
2. $\mathsf{N}(\Sigma)$ *is constructible in logarithmic space; and*
3. *for every database D and UCQ Q, $D \cup \Sigma \models Q$ if and only if $D \cup \mathsf{N}(\Sigma) \models Q$.*

In the sequel, in order to simplify our technical proofs, sometimes we will focus on normalized sets of DTGDs. It will be explicitly stated when the DTGDs under consideration are assumed to be in normal form.

2.3 The Disjunctive Chase

In our investigation of (U)CQ answering, we employ the *disjunctive chase*, originally introduced by Deutsch and Tannen [2005] for disjunctive embedded dependencies. The disjunctive chase extends the well-known chase procedure in such a way that it can treat disjunctive dependencies. This is done as follows. Each disjunctive chase step "branches" out to several instances, each satisfying one of the disjuncts of the DTGD that is applied, and thus the result of the disjunctive chase is, in general, a set of instances (and not a single instance as of the classical chase). The disjunctive chase works on an instance through the so-called DTGD chase rule:

Definition 2.3 (DTGD Chase Rule)**.** *Consider an instance I, and a DTGD σ of the form $\phi(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Y}\, \psi_i(\mathbf{X}, \mathbf{Y})$. σ is* applicable *to I if there exists a homomorphism h such that $h(\phi(\mathbf{X})) \subseteq I$, and the result of applying σ to I with h is the set $\{I_1, \ldots, I_n\}$, where $I_i = I \cup h'(\psi_i(\mathbf{X}, \mathbf{Y}))$, for each $i \in [n]$, and $h' \supseteq h$ is such that $h'(Y)$ is a "fresh" null not occurring in I, and following lexicographically all those in I, for each $Y \in \mathbf{Y}$. For such an application, which defines a single* chase step*, we write $I\langle \sigma, h \rangle \{I_1, \ldots, I_n\}$.*

A *disjunctive chase tree* of a database D and a set Σ of DTGDs is a (possibly infinite) tree such that the root is D, and for every node I, assuming that $\{I_1, \ldots, I_n\}$ are the children of I, there exists $\sigma \in \Sigma$ and a homomorphism h such that $I\langle\sigma, h\rangle\{I_1, \ldots, I_n\}$. The disjunctive chase algorithm for D and Σ consists of an exhaustive application of DTGD chase steps in a *fair fashion*, that is, every DTGD that can be applied is applied in each path of the chase tree, which leads to a disjunctive chase tree T of D and Σ; let $chase(D, \Sigma)$ be the set $\{I \mid I \text{ is a leaf of } T\}$. Notice that each leaf of T is well-defined as the least fixpoint of a monotonic operator. By construction, each instance in $chase(D, \Sigma)$ is a model of D and Σ. Interestingly, $chase(D, \Sigma)$ is a *universal model set* of D and Σ, that is, for each $I \in models(D, \Sigma)$, there exists $J \in chase(D, \Sigma)$ and a homomorphism h_J such that $h_J(J) \subseteq I$; cf. Deutsch et al. [2008]. This property implies the following useful result, which actually shows that the disjunctive chase is a formal algorithmic tool for query answering purposes.

Theorem 2.4. *Consider a database D, a set Σ of DTGDs, and a UCQ Q. It holds that $D \cup \Sigma \models Q$ iff $I \models Q$, for each $I \in chase(D, \Sigma)$.*

To describe the above result, we say that the chase is *universal*.

2.4 Complexity Theory

A complexity class is a set of languages (also referred to as *problems*), where a language is a set of words over an alphabet. We are interested in the decision problem: given a word ω and a language $\mathcal{L}$, decide whether $\omega \in \mathcal{L}$. Complexity classes are characterized by the well-known notion of *Turing machines*. A Turing machine is said to *decide* a language L if it solves the corresponding decision problem. Turing machines can be *deterministic (DTMs)* or *non-deterministic (NTMs)*. For a function $t : \mathbb{N} \to \mathbb{N}$, the complexity class $\textsc{Time}(t(n))$ (resp. $\textsc{NTime}(t(n))$) is defined as the set $\{\mathcal{L} \mid \exists$ DTM (resp. NTM) M deciding $\mathcal{L}$ in time $O(t(n))\}$. $\textsc{Space}(t(n))$ and $\textsc{NSpace}(t(n))$ are defined analogously, where M decides in space $O(t(n))$. The complexity class of *polynomial time (PTIME)* is defined as follows: $\textsc{PTime} = \bigcup_{k=0}^{\infty} \textsc{Time}(n^k)$. The class of *non-deterministic polynomial time (NP)* is defined as $\textsc{NP} = \bigcup_{k=0}^{\infty} \textsc{NTime}(n^k)$. Analogously we define PSPACE, NPSPACE, LOGSPACE, NLOGSPACE, EXPTIME, NEXPTIME, 2EXPTIME, N2EXPTIME, etc., depending on whether the function $t(n)$ is a polynomial, logarithmic, exponential, double-exponential, etc. If there exists no Turing machine solving a problem in finite time, we say the problem is *undecidable*.

If the decision problem for a language $\mathcal{L}$ is in a complexity class $\mathcal{A}$ (i.e. $\mathcal{L} \in \mathcal{A}$), then the converse of the decision problem (i.e. for a word ω, decide whether $\omega \notin \mathcal{L}$) is in co-$\mathcal{A}$. It is known that for any function $t(n)$, $\textsc{Time}(t(n)) = \text{co-}\textsc{Time}(t(n))$. Further, by Savitch's Theorem [Savitch, 1970], for any function $t(n) > \log n$, $\textsc{Space}(t(n)) = \textsc{NSpace}((t(n))^2)$, and thus $\textsc{PSpace} = \textsc{NPSpace}$, $\textsc{ExpSpace} = \textsc{NExpSpace}$, etc. It is also known that non-deterministic

space is closed under complementation, that is, $\text{NSPACE}(t(n)) = \text{co-NSPACE}(t(n))$; cf. [Immerman, 1988; Szelepcsényi, 1988]. A Turing machine with an *oracle for problem B* can, at each step of the computation, query the oracle for a solution to B. For a complexity class $\mathcal{A}$, $\mathcal{A}^B$ is the complexity class of problems solvable by a Turing machine for $\mathcal{A}$ with access to an oracle for problem B. The notation is usually extended to sets of languages $\mathcal{B}$, such that $\mathcal{A}^{\mathcal{B}} = \bigcup_{B \in \mathcal{B}} \mathcal{A}^B$. For each $i \geq 0$, the complexity class $\Sigma^{\text{P}}_{\text{i}+1}$ is defined as $\text{NP}^{\Sigma^{\text{P}}_{\text{i}}}$, with $\Sigma^{\text{P}}_0 = \text{PTIME}$. Furthermore, we denote the complement as follows: $\Pi^{\text{P}}_{\text{i}} = \text{co-}\Sigma^{\text{P}}_{\text{i}}$.

2.5 Alternating Turing Machines

The concept of alternation was introduced by Chandra et al. [1981]. An *alternating Turing machine* is a tuple $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states, respectively partitioned into universal states, existential states, an accepting state and a rejecting state, Λ is the tape alphabet, $\delta \subseteq (S \times \Lambda) \times (S \times \Lambda \times \{-1, 0, +1\})$ is the transition relation, and $s_0 \in S$ is the initial state. We assume that Λ contains a special blank symbol $\sqcup$. The symbols -1, 0 and $+1$ denote the cursor directions left, stay and right.

A *computation tree* for M is a tree labelled by configurations (tape content, cursor position, and internal state) of M such that (1) if node v is labelled by an existential configuration, then v has one child, labelled by one of the possible successor configurations; (2) if v is labelled by a universal configuration, then v has one child for each possible successor configuration; (3) the root is labelled by the initial configuration; and (4) all leaves are labelled by accepting or rejecting configurations. A computation tree is *accepting* if it is finite and all leaves are labelled by accepting configurations.

The acceptance problem, that is, the problem of deciding whether, given an alternating Turing machine M, there exists an accepting configuration tree for M, is known to be EXPTIME-hard (resp. 2EXPTIME-hard) for alternating Turing machines using only polynomial (resp. exponential) space.

2.6 Notational Conventions

In the remainder of this thesis we will, in general, follow certain notational conventions that allow for the easy identification of the meaning of symbols used in mathematical formulas and expressions. In particular, we shall denote constants as lower-case letters at the beginning of the alphabet (e.g. a, b, c); relations as lower-case letters around the letter "r" (e.g. s, r, p), null values as the lower-case letter z, usually with a subscript; variables as upper-case letters around the letter "X" (e.g. V, W, X, Y, Z); vectors or sets of constants (resp. variables) as a corresponding bold letter (e.g. $\mathbf{a}$, $\mathbf{b}$, resp. $\mathbf{X}$, $\mathbf{Y}$); CQs as q and UCQs as Q; databases as D; dependencies as σ and sets of dependencies as Σ; schemas as calligraphic upper-case letters near "R" (e.g. $\mathcal{S}, \mathcal{R}$); and atoms as an underlined letter (e.g. $\underline{a}, \underline{r}$).

3 Decidability in the Deterministic World

In this chapter we discuss the different kinds of restrictions known to ensure decidability of query answering under sets of (non-disjunctive) TGDs. This will form the basis for our investigation into the impact of adding disjunction to such deterministic classes of TGDs. Several relevant complexity results are surveyed.

Generally the restrictions guaranteeing decidability of query answering can be classified as either *abstract* or *concrete* properties. Abstract classes are based on the behaviour of the underlying reasoning algorithms. Therefore it is generally not decidable whether a given set of TGDs Σ has the desired property or not. Although there are a number of useful abstract classes in the deterministic world (see, e.g. Baget et al. [2011a]; Baget and Mugnier [2002]; Calvanese et al. [2007]), these properties have predominantly been studied for deterministic theories and thus cannot provide adequate decidability guarantees vis-a-vis non-determinism. Concrete classes on the other hand are defined by syntactic properties on sets of TGDs that are computable in finite time using appropriate algorithms. The focus of the subsequent chapters of this work will be on extending these syntactic restrictions to DTGDs.

Each syntactic restriction discussed below is either a *local* or a *global* condition. Local conditions place restrictions on the form and structure of each individual TGD, and are thus usually easy to verify. Global conditions, on the other hand, place restrictions on a whole set of TGDs. This makes this form of restriction much more powerful, but it can be a difficult problem to determine whether a given set of TGDs actually adheres to the restrictions.

The following sections deal with well-established families of syntactic restrictions. Section 3.1 provides an overview of classes of TGDs based on a principle called guardedness; Section 3.3 deals with classes of TGDs that possess the sticky property; and Section 3.2 deals with classes of TGDs that are acyclic in the sense that they guarantee chase termination, thus guaranteeing that all relevant models of the theory are finite.

3.1 Guarded-based Classes of TGDs

This section deals with classes of TGDs that are based on a principle called *Guardedness*. This syntactic restriction was first introduced in the context of the Guarded Fragment of First-Order Logic by Andréka et al. [1998] and ensures that the chase has a tree-like structure. For more details on the Guarded Fragment, see Chapter 4.

There are multiple guarded-based classes of TGDs for which query answering becomes decidable. The following sections will deal with them in turn, in ascending order with respect to their expressivity.

3.1.1 Inclusion Dependencies

The first class discussed here are *Inclusion Dependencies (IDs)*, a very well known and established formalism that is used predominantly in the database world to express the fact that values in certain columns of a relation must also appear in certain columns of another relation. The following example shows how to state that every manager is an employee, where mgr and emp are predicates with the obvious meaning.

Example 3.1. *The following is an example of an ID, using different notations:*

- *Traditional ID notation:* $mgr[1] \subseteq emp[1]$
- *SQL notation:*

```
CREATE TABLE mgr
(
    …
    FOREIGN KEY (id) REFERENCES emp(id)
)
```

- *TGD notation:* $mgr(X, Y) \rightarrow \exists Z_1 \exists Z_2\, emp(X, Z_1, Z_2)$

As this work mainly deals with TGDs, we will stick to the TGD notation. IDs are very simple TGDs, with at most one atom in the body and the head respectively, and no repeated variables (i.e. self-joins). The formal definition is as follows:

Definition 3.2. *A TGD σ is called an* Inclusion Dependency *if it is of the form*

$$r(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, s(\mathbf{X}, \mathbf{Y}),$$

where r and s are relations, and $\mathbf{X}$ *and* $\mathbf{Y}$ *are disjoint sequences of variables where each variable only appears once.*

It is easy to show that IDs are already powerful enough to give rise to an infinite chase expansion. Given a database $D = \{r(a, b)\}$ and the ID $\sigma : r(X, Y) \rightarrow \exists Z\, r(Y, Z)$, then chasing D with σ results in the infinite instance $\{r(a, b), r(b, z_1), r(z_1, z_2), \ldots\}$.

However, answering a query q under sets of IDs Σ is decidable. This follows from the fact that the chase expansion of a database D and Σ is a tree (or forest) and thus, also the logical theory $D \cup \Sigma \cup \neg q$ must have a tree-like model, if the query is false. Using the famous, graph-theoretic

result by Courcelle [1990], which roughly states that for fragments of monadic second-order logic that exhibit the *finite treewidth model property*, we get decidability of the satisfiability problem. This argument was already established by Calì et al. [2013] for the more expressive class of weakly-guarded TGDs. We shall see in Section 3.1.5 that IDs are trivially weakly-guarded TGDs.

Calì et al. [2003b] introduce a backward-chaining algorithm that rewrites a given BCQ q and a set Σ of IDs into a union of BCQs q_Σ, such that for every database D, $D \cup \Sigma \models q$ if and only if $D \models q_\Sigma$. This immediately implies first-order rewritability. An alternative way to obtain this result is that IDs possess the so-called *bounded derivation-depth property (BDDP)*, introduced by Calì et al. [2012], which roughly says that for query answering purposes it is enough to consider a finite, initial part of the chase, where the size of this initial part only depends on the set of TGDs and the query, but not on the database. Using this initial part, a first-order query q_Σ can be obtained, such that, again, it holds that for every database D, $D \cup \Sigma \models q$ if and only if $D \models q_\Sigma$. As shown by Vardi [1995], answering first-order queries over a database is known to be in the complexity class AC_0 when only the database is considered as input, and we thus get that CQ-Answering is in AC_0 in the data complexity.

In the classical work by Johnson and Klug [1984], the complexity of CQ-Containment under inclusion dependencies was studied. The complexity results obtained there also hold for our CQ-Answering problem, as the two problems are LogSpace-equivalent (see also Chapter 2). In a rather involved proof, they establish that it is enough to compute a finite, initial portion of the chase for query answering purposes and based on this fact, they design a PSpace algorithm to decide the CQ-Containment problem. In case where either the predicate arity is bounded by a constant, or the set of TGDs Σ is fixed, this algorithm works in NP.

The corresponding PSpace lower bound is shown by giving a reduction to the implication problem under IDs; see Casanova et al. [1984]. The NP lower bound in case of fixed sets of IDs follows directly from the NP-hardness of query answering without constraints shown in [Chandra and Merlin, 1977].

3.1.2 Linear TGDs

Calì et al. [2012] extend Inclusion Dependencies to *linear TGDs*. This class of TGDs, in comparison to IDs, allows variable repetition, as well as a conjunction of multiple atoms in the head of each rule. However, only one atom is allowed in the body. This allows, for example, self-joins. The syntactic restriction is a local one, imposed on every single TGD.

Example 3.3. *The following is a set of linear TGDs:*

$$\begin{aligned} s(X,Y,X) &\rightarrow \exists Z\, r(X,Z) \\ r(X,Y) &\rightarrow \exists Z\, s(Z,X,Z), s(Y,X,Y) \end{aligned}$$

Note that repeated variables can occur in the body as well as the head of the TGDs, and a conjunction of multiple head atoms is allowed. Note further that none of the two linear TGDs above are IDs.

Linear TGDs allow us, for example, to test for reflexive relations. Imagine the following TGD: $relation(X, X) \rightarrow reflexive(X)$. It "extracts" all the individuals in a relation that are related to themselves. With IDs it is not possible to express such a statement. The formal definition of linear TGDs is as follows:

Definition 3.4. *A TGD σ is called* linear *if it is of the form*

$$r(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \phi(\mathbf{X}, \mathbf{Y}),$$

where r is a relation, ϕ is a conjunction of atoms, and $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sequences of variables.

As every ID is a linear TGD, by the simple example given in Section 3.1.1 we can immediately see that also linear TGDs can give rise to infinite chase expansions. However, by the same argument as for IDs, we get decidability: again, the chase expansion of a database D and Σ of linear TGDs is a tree (or forest). This was originally shown by Calì et al. [2013] for the more expressive class of weakly-guarded TGDs, of which linear TGDs trivially are a subclass.

First-order rewritability of sets of linear TGDs was shown by Calì et al. [2012], who show that we can again rely on the BDDP. However, the backward-chaining algorithm for IDs, given in [Calì et al., 2003b], cannot be employed for linear TGDs, as it cannot deal with variable repetitions in the rule heads. However, an extended version of this algorithm, that can also treat linear rules is presented in [Calì et al., 2012b]. Unfortunately, it is well known that approaches like the ones in [Calì et al., 2003b; Calì et al., 2012b] are not well suited for real-world databases, because they produce a perfect rewriting, which in general is very large in size, and thus many queries would have to be evaluated. Alternative techniques that try to optimize the rewriting can for example be found in [Orsi and Pieris, 2011; Gottlob et al., 2011].

From the earlier argument, we get that clearly, CQ-Answering under linear TGDs is in AC_0 in the data complexity. Regarding the other complexity cases, the same techniques as for IDs can be employed. Thus, we get that CQ-Answering under linear TGDs is PSpace-complete in the combined complexity, and NP-complete in cases where the arity is bounded by a constant or the set of dependencies is fixed.

Multi-Linear TGDs. For completeness reasons will also consider the class of *multi-linear* DTGDs. Multi-linearity is a slight relaxation of the notion of linearity, and appears, for example, in [Calì et al., 2012]. The body of a linear DTGD consists of a single atom, containing all the universally quantified variables. Multi-linear relaxes this restriction slightly, whereby the body of the rule is allowed to be a conjunction of atoms, all of which contain all the universally quantified variables.

3.1.3 Guarded TGDs

An extension of linear TGDs, proposed by Calì et al. [2013] are *guarded TGDs*. Guardedness in this context is again a local condition: informally, a TGD is guarded if one body atom contains all the body variables. This allows "filtering" of a relation by joining to a subset of that relation. Also, it allows a very limited form of transitivity. The following example demonstrate the expressive power of guarded TGDs:

Example 3.5. *The following set of (non-linear) TGDs express that an employee that supervises another employee is a manager (σ_1), and that the unary relation $supervised$ shall contain all the employees that are supervised, directly or indirectly, by the CEO (σ_2 and σ_3).*

$$\sigma_1 : \quad emp(X), supervises(X, Y), emp(Y) \rightarrow mgr(X)$$
$$\sigma_2 : \quad ceo(X), supervises(X, Y) \rightarrow supervised(Y)$$
$$\sigma_3 : \quad supervised(X), supervises(X, Y) \rightarrow supervised(Y)$$

Note that the above rules are all non-linear, but there is an atom in the body of each, that "guards" the rule, that is, it contains all the variables occurring in the respective rule body. Also note that σ_2 and σ_3 only encode a limited form of transitivity: guardedness only allows us to express reachability, that is, identify those nodes reachable from a given set of origin nodes.

As seen in the example above, guarded TGDs cannot encode full transitivity, as the transitive closure rule $r(X, Y), r(Y, Z) \rightarrow r(X, Z)$ is not a guarded TGD and violates the following definition which sets out the formal conditions for guarded TGDs:

Definition 3.6. *A TGD σ is called* guarded *if it is of the form*

$$\alpha(\mathbf{X}), \phi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y}),$$

where α is a relation of arity $|\mathbf{X}|$, ϕ and ψ are conjunctions of atoms, $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sequences of variables. The atom $\alpha(\mathbf{X})$ is called the guard.

Linear TGDs and inclusion dependencies are trivially guarded, as they only have exactly one body atom, therefore sets of guarded TGDs can give rise to infinite chase expansions. However, as opposed to linear TGDs, sets of guarded TGDs are also not first-order rewritable sets. This is shown by creating a database, query and a set of guarded TGDs in such a way that answering the query requires the computation of reachability over a relation in the database. It is well known that this property cannot be expressed in a finite first-order query. Query answering under guarded TGDs remains decidable however. Because of the guards that serve as a "guideline" during the chase expansion, the chase still retains a tree-like structure (i.e. represented as a graph, the chase always has finite treewidth, even though the chase expansion might be infinite). Because of this, the argument for IDs and linear TGDs is still valid, as

the famous theorem by Courcelle [1990] does not only hold for trees but also for tree-like (i.e. finite treewidth) structures.

The complexity of query answering under guarded TGDs was investigated in-depth by Calì et al. [2012, 2013]. First, it is established that guarded TGDs possess the so-called *bounded guard-depth property (BGDP)*, which is related to the BDDP in that inasmuch as it only pertains to the guard atoms and their immediate children in the chase. The *chase graph* of a database D and a set Σ of TGDs is a directed graph whose nodes are the atoms in $chase(D, \Sigma)$ and having an edge between atoms $\underline{a}$ and $\underline{b}$ if, during the application of the TGD σ in a chase step, $\underline{a}$ is used to satisfy the body of σ and $\underline{b}$ is an atom in the head of σ introduced by that chase step. An atom $\underline{a}$ is marked as a guard atom, if during the application of a chase step, $\underline{a}$ is mapped to the guard of a rule σ. The *guarded chase forest* for D and Σ is the restriction of the respective chase graph to all guard atoms and their immediate children. The BGDP now states that whenever the chase entails a query, it does so in a finite, initial portion C of the guarded chase forest, where its size depends only on the query and the set of TGDs, but not the database. In case the former two are fixed, this initial portion can be constructed, and the (fixed) query evaluated in polynomial time. The lower bound is obtained by a reduction from the atom entailment problem under propositional logic programs with at most two body atoms; see, e.g. Dantsin et al. [2001]. We thus have that CQ-Answering under sets of guarded TGDs is PTime-complete in the data complexity.

Regarding the other complexity cases, Calì et al. [2013] exhibit a rather involved algorithm, deciding the CQ-Answering problem under sets of guarded TGDs in $\text{NP}^{\mathcal{C}}$, where $\mathcal{C}$ is an oracle for complexity class $\mathcal{C}$. In case of the combined complexity, $\mathcal{C}$ is the complexity class 2ExpTime. In case where the arity is bounded by a constant, $\mathcal{C}$ equals ExpTime. In case of a fixed set of TGDs, $\mathcal{C}$ equals PTime. This gives us 2ExpTime, ExpTime and NP membership, respectively for these three cases. The lower bounds are obtained by simulating an alternating Turing machine using, respectively, exponential space, polynomial space and logarithmic space, in case of the combined complexity, bounded arity and fixed set of TGDs. We thus have 2ExpTime-, ExpTime- and NP-completeness for these cases, respectively. Note that the NP-hardness also follows from the fact that the CQ-Containment problem is NP-hard, even for empty sets of TGDs; cf. Chandra and Merlin [1977].

3.1.4 Frontier-Guarded TGDs

Frontier-guarded TGDs, proposed by Baget et al. [2011a], is a local condition placed on TGDs that relaxes the guardedness condition for TGDs. It does so by exploiting the notion of the frontier of a TGD. The *frontier* of a TGD σ, denoted $fr(\sigma)$, is the set of variables occurring both in the body and in the head of σ. The idea of frontier-guardedness is that it suffices to guard those variables that are propagated, that is, those in the frontier. This allows more powerful filtering rules—in fact, as long as only guarded variables are propagated, a full conjunctive query can be specified in the body of a rule. The following example illustrates this point:

Example 3.7. *The following is a set of frontier-guarded TGDs:*

$$\begin{aligned} r(X,Y), r(Y,Z), r(Z,X) &\rightarrow incycle(X) \\ incycle(X), r(X,Y), r(Y,Z) &\rightarrow incycleandchain(X) \end{aligned}$$

The rules above state that an individual is in a circle if it participates in a circle of size three. They further state that the individual is also in a chain, if it is the root of a chain of length two. Note that none of the above TGDs are guarded. However, the variables in the frontier always appear together in at least one body atom.

Frontier-guarded rules allow us to check arbitrary conditions about individuals. However, only propositional information about the validity of these conditions can be propagated, but no information on "how" the individuals fulfil the conditions, or which other individuals are involved int fulfilling the conditions. In the following, we give the formal definition of frontier-guarded TGDs:

Definition 3.8. *A TGD σ is called* frontier-guarded *if it is of the form*

$$\alpha(\mathbf{Y}), \phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{Y}, \mathbf{Z}),$$

where α is a relation of arity $|\mathbf{Y}|$, ϕ and ψ are conjunctions of atoms, $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Y}$ are pairwise disjoint sequences of variables, where the variables in $\mathbf{X}$ appear only in the body, those in $\mathbf{Y}$ appear both in the body and the head, and those in $\mathbf{Z}$ appear only in the head of the rule. The atom $\alpha(\mathbf{Y})$ is called the frontier-guard*.*

Like guarded TGDs, frontier-guarded TGDs can give rise to an infinite chase expansion. However, decidability follows, as it did for guarded TGDs, from the fact that sets of frontier-guarded TGDs exhibit the finite treewidth model property, that is, if there is a model, then there also is a model of finite treewidth. By Courcelle's Theorem [Courcelle, 1990], we again derive decidability.

The complexity of the CQ-Answering problem was investigated in Baget et al. [2011b]. Regarding the data complexity, PTime-completeness is established by exploiting the lower bound from guarded TGDs, as well as an upper bound by reduction to the fact inference problem of Datalog programs, which is known to be in PTime in the data complexity; see, e.g. the extensive survey by Dantsin et al. [2001].

Regarding the combined complexity, a 2ExpTime upper bound is established by exploiting a result from [Bárány et al., 2014], where it is shown that CQ-Answering under guarded first-order theories is in 2ExpTime. In particular, given an instance $\langle D, \Sigma, q \rangle$ of the CQ-Answering problem under sets of frontier-guarded TGDs, Baget et al. construct, in polynomial time, a guarded first-order formula φ and a union of CQs q', such that $D \cup \Sigma \models q$ if and only if $\varphi \models q'$. The corresponding lower bound follows directly from guarded TGDs.

3.1.5 Weakly-(Frontier-)Guarded Sets of TGDs

Weakly-guarded sets of TGDs were introduced by Calì et al. [2013] and generalize the class of guarded TGDs. The idea is that in fact, not every universally quantified variable must be guarded, but it suffices to guard the variables occurring at *affected positions*, that is, informally, those positions where a null value can appear during the construction of the chase. All other positions will hold only constants from $dom(D)$, which is a finite set, which means that these constants alone cannot cause an infinite chase expansion, even if unguarded. The following example demonstrates the expressiveness of weakly-guarded sets of TGDs:

Example 3.9. *The following is a weakly-guarded set of TGDs:*

$$\begin{aligned} r(X,Y), r(Y,Z) &\rightarrow r(X,Z) \\ r(X,Y) &\rightarrow \exists Z\, s(X,Y,Z) \\ s(X,Y,Z), r(X,X) &\rightarrow \exists W\, s(X,Z,W) \end{aligned}$$

Observe that, in the second and third TGD, an existentially quantified variable appears in position $s[3]$*, and therefore this position is affected. However, in the third TGD, the variable at position* $s[3]$ *is propagated to* $s[2]$*. We must therefore conclude that also the latter position is affected.*

Note that, as seen in the example above, weakly-guarded sets of TGDs are a global property. We cannot check TGD after TGD, but have to take the entire set of TGDs into account, to determine the set of affected positions.

Weakly-guarded sets of TGDs capture Datalog inasmuch as they allow for any set of Datalog rules, as long as they only operate on non-affected positions. In the example above, we can, for example, encode the transitive closure of a binary relation, as long as we know that we will only compute the transitive closure over the finite set of constants of $dom(D)$. However, once null values come into play, we are restricted to guarded rules, at least where the affected positions are concerned, in order to preserve the tree-like structure of the chase. We may thus have an arbitrarily cyclic, but finite structure inside the chase that operates only on constants from $dom(D)$, and a possible infinite, but guarded and thus tree-like, structure in addition to that. Overall, this thus preserves the finite treewidth model property. In order to formally define weakly-guarded sets of TGDs, we first need to formally define the notion of an affected position:

Definition 3.10. *Given a set of TGDs* Σ *over a relational schema* $\mathcal{R}$*, the set of* affected positions*, denoted* $affected_{\Sigma}(\mathcal{R})$*, is defined inductively as the least fixed point of the following:*

- *Let* $r[i]$ *be a position in the head of a TGD* $\sigma \in \Sigma$*. If an existentially quantified variable appears at position* $r[i]$ *in* σ*, the* $r[i]$ *is affected.*
- *Let* $r[i]$ *be a position in the head of a TGD* $\sigma \in \Sigma$*. If a universally quantified variable appears at position* $r[i]$*, that variable only appears at affected positions in the body of* σ*, then* $r[i]$ *is affected.*

With this definition in place, we can now define weakly-guarded sets of TGDs:

Definition 3.11. *A set* Σ *of TGDs is* weakly-guarded *if for each TGD* $\sigma \in \Sigma$ *it holds that all universally quantified variables that appear only at affected positions in the body of* σ *appear together in a body atom. This atom is called the* weak-guard*.*

While weakly-guarded sets of TGDs can obviously lead to an infinite chase expansion, they do preserve the finite treewidth model property, that is, the fact that the chase expansion has finite treewidth. This is established in [Calì et al., 2013] using the notion of $[S]$-*acyclicity* of the chase, where S is a set of symbols. $[S]$-acyclicity generalizes classical hypergraph acyclicity in that it, informally speaking, restricts the cyclic part to S, and we thus have that an instance over a schema $\mathcal{R}$ is acyclic if and only if it is $[\emptyset]$-acyclic and its treewidth is less than or equal to $|S| + arity(\mathcal{R})$. In particular, for weakly-guarded sets of TGDs Σ and a database D over a schema $\mathcal{R}$, the chase is shown to be $[dom(D)]$-acyclic, and therefore to have finite treewidth, as both D and $\mathcal{R}$ are finite. From Courcelle's Theorem [Courcelle, 1990], by the argument presented earlier, we again get decidability, and by the fact that weakly-frontier-guarded sets of DTGDs capture all previously introduced classes, we get decidability for all the guarded-based classes.

Calì et al. [2013] then proceed to study the complexity of CQ-Answering under weakly-guarded sets of TGDs, by employing an extended version of the algorithm presented in Section 3.1.3. Recall that the algorithm runs in $\text{NP}^{\mathcal{C}}$, where $\mathcal{C}$ is a complexity class. In case of the combined complexity, $\mathcal{C}$ is shown to be 2ExpTime. In case where the arity is bounded by a constant, $\mathcal{C}$ reduces to ExpTime and remains so even when the set of TGDs and the query are both fixed. Consequently, we have that CQ-Answering under weakly-guarded sets of TGDs is in 2ExpTime in the combined complexity, and in ExpTime in case of bounded arity, fixed set of TGDs and even in the data complexity.

For each of the discussed upper bounds, a corresponding lower bound holds. In the combined complexity, the 2ExpTime-hardness follows immediately from the fact that the problem is already 2ExpTime-hard under guarded TGDs. In the data complexity, ExpTime-hardness is shown by simulating an alternating PSpace Turing machine. The latter result is rather surprising, as CQ-Answering under guarded TGDs is possible in PTime.

Weakly-Frontier-Guarded Sets of TGDs. Analogously to the generalization of sets of guarded TGDs to weakly-guarded sets of TGDs discussed above, Baget et al. [2011a] generalized their notion of frontier-guarded TGDs to weakly-frontier-guarded sets of TGDs. The formal definition can be found below:

Definition 3.12. *A set* Σ *of TGDs is* weakly-frontier-guarded *if for each TGD* $\sigma \in \Sigma$ *it holds that all the variables of* $fr(\sigma)$ *that appear only at affected positions in the body of* σ *appear together in a body atom. This atom is called the* weak-frontier-guard*.*

For example, the TGD

$$\sigma : supervises(X, Z), mgr(Y, Z) \rightarrow \exists W\, mgr(Y, W), supervises(W, X)$$

is weakly-frontier-guarded, with $supervises(X, Z)$ as its weak-frontier-guard. Clearly the affected positions are $mgr[1]$, $supervises[1]$ and $supervises[2]$. Given the frontier $fr(\sigma) = \{X, Y\}$, we can see that the single variable occurring only at affected positions in $body(\sigma)$ is X which is guarded by the weak-frontier-guard. It is easy to verify that σ is neither frontier-guarded, nor weakly-guarded.

Regarding decidability, obviously the chase expansion under weakly-frontier-guarded sets of TGDs can be infinite. However, Baget et al. [2011a] establish the finite treewidth model property of the chase expansion and thus decidability follows.

A complexity investigation of the class of weakly-frontier-guarded sets of TGDs was performed by Baget et al. [2011b]. CQ-Answering is shown to be ExpTime-complete in the data complexity, which is shown by establishing that weakly-frontier-guarded sets of TGDs fall into an abstract, decidable class of TGDs called *greedy bounded treewidth sets*. This abstract class, introduced in [Baget et al., 2011b], guarantees membership of the CQ-Answering problem in ExpTime[1]. The lower bound immediately follows from the ExpTime-hardness of CQ-Answering under weakly-guarded sets of TGDs, as discussed earlier in this section.

Regarding the combined complexity, Baget et al. [2011b] show that, given a database D, a weakly-frontier-guarded set Σ of TGDs and a query Q, it is possible to construct a set Σ' of frontier-guarded TGDs, such that $D \cup \Sigma \models q$ if and only if $D \cup \Sigma' \models q$. This reduction works by cleverly "grounding" the variables at non-affected positions. In this way, the 2ExpTime upper bound of CQ-Answering under frontier-guarded TGDs transfers over to weakly-frontier-guarded sets of TGDs. The corresponding lower bound follows immediately from the fact that CQ-Answering under guarded TGDs is already 2ExpTime-hard; see Section 3.1.3.

3.1.6 Summary

The well-established guarded-based classes of TGDs discussed above are collectively based on the guarded fragment of first-order logic, introduced by Andréka et al. [1998] further and analyzed w.r.t. the complexity by Grädel [1999]. When applied to TGDs, the guardedness restriction naturally yields the class of guarded TGDs, introduced by Calì et al. [2013]. To lower the complexity of query answering, this class was further restricted to linear TGDs (see Calì et al. [2013, 2012]), as well as to IDs, a well-known formalism long used in the area of databases. In order to increase the expressive power, generalizations of the class of guarded TGDs were proposed, namely, weakly-guarded sets of TGDs (see Calì et al. [2013]), frontier-guarded TGDs and weakly-frontier-guarded sets of TGDs (see Baget et al. [2011a,b] for details). A comparison of expressivity of these formalisms can be seen in Figure 3.1. Here, IDs are inclusion

[1] In fact, it guarantees that CQ-Answering is in 3ExpTime in the combined complexity, in 2ExpTime in case of bounded arity, and in ExpTime in the data complexity.

dependencies, LTGDs are sets of linear TGDs, GTGDs are sets of guarded TGDs, FGTGDs are sets of frontier-guarded TGDs, WGTGDs are weakly-guarded sets of TGDs, and WFGTGDs are weakly-frontier-guarded sets of TGDs.

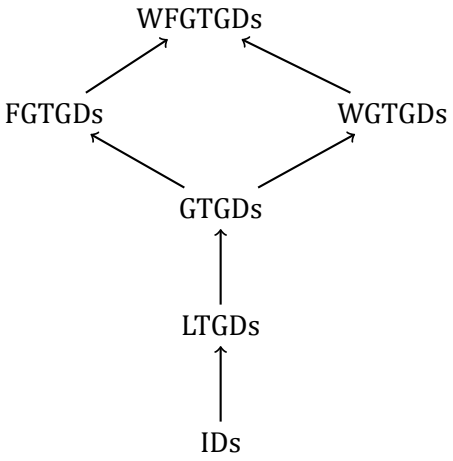


Figure 3.1: Expressivity comparison of guarded-based classes of TGDs.

All of the classes discussed above produce a chase expansion that is of finite treewidth, and thereby it follows that the CQ-Answering problem is decidable. The complexity ranges from AC_0 to ExpTime in the data complexity, and from PSpace to 2ExpTime in the combined complexity. Table 3.1 lists the different complexity results.

Class	Data Complexity	Combined Complexity
IDs	in AC_0	PSpace-complete
LTGDs	in AC_0	PSpace-complete
GTGDs	PTime-complete	2ExpTime-complete
FGTGDs	PTime-complete	2ExpTime-complete
WGTGDs	ExpTime-complete	2ExpTime-complete
WFGTGDs	ExpTime-complete	2ExpTime-complete

Table 3.1: The complexity of CQ-Answering under guarded-based classes of TGDs.

3.2 Acyclicity-based Classes of TGDs

This section deals with classes of TGDs that ensure decidability of query answering by forcing the chase to terminate. From the fact that the chase produces a universal model that is sufficient for query answering, we can immediately conclude that a finite-time algorithm exists that builds the chase expansion and then answers the query over this finite model.

Multiple variants of such acyclicity-based classes of TGDs exist, with the simplest one being sets of acyclic TGDs. This class simply forbids recursion and the chase trivially terminates. However, such a simple class did not prove to be very interesting or useful in practice, and thus we will not focus on this class here. The following two sections will, respectively, deal with

the class of full or total TGDs that are in essence pure Datalog rules, and the class of weakly-acyclic sets of TGDs, that limits the use of existential quantification in order to guarantee chase termination.

3.2.1 Full TGDs

A TGD is called *full* (or *total*, cf. Beeri and Vardi [1981]), if there are no existentially quantified variables in its head. Therefore, the class of full (single-atom head) TGDs coincides with the language of Datalog; see, e.g. Abiteboul et al. [1995]. It is easy to see that sets of full TGDs cannot be first-order rewritable, as they can express transitivity[2]. However, it is easy to see that the chase expansion under a set of full TGDs is finite. As there are no existentially quantified variables, the chase of a database D with a set Σ of full TGDs over a schema $\mathcal{R}$ can only make use of the constants that already appear in $dom(D)$, and as D is finite, it must terminate.

CQ-Answering under full TGDs is PTime-complete in the data complexity and ExpTime-complete in the combined complexity. The upper bound follows from the observation, that the maximum number of atoms that can appear in the chase is the number of relations times the number of permutations of constants from $dom(D)$ in a sequence of the length of the maximum arity of the schema. In formal terms, the maximum number of atoms $k = |\mathcal{R}| \cdot |dom(D)|^{arity(\mathcal{R})}$. In case of the data complexity, where the set of TGDs, and thus the schema $\mathcal{R}$ is fixed, at most polynomially many atoms can appear in the chase, yielding the desired upper bounds. The corresponding lower bounds follow from classical results regarding the fact inference problem for Datalog programs; see, e.g. Dantsin et al. [2001].

3.2.2 Weakly-Acyclic Sets of TGDs

Weakly-acyclic sets of TGDs, introduced in the context of data exchange in the landmark paper by Fagin et al. [2005], are a careful combination and extension of both acyclic TGDs and full TGDs. The main idea is to prevent recursions through existential quantifications. Thus, the only symbols that can appear in atoms in the chase of a database D with a weakly-acyclic set of TGDs are the constants in $dom(D)$, and a finite number of labelled nulls. Thus, after finitely many steps, the chase must terminate.

The formal definition of weak acyclicity is based on the notion of the dependency graph, which is defined as follows:

Definition 3.13. *The* dependency graph $G = \langle V, E = E_N \uplus E_S \rangle$ *of a set* Σ *of TGDs over a schema* $\mathcal{R}$ *is a multigraph, where the multiset of edges is partitioned into normal edges* E_N *and special edges* E_S*. G is constructed as follows:*

- *Let* V *be the set of positions* $r[i]$ *occurring in* $\mathcal{R}$*.*

[2] For example, the rule $r(X,Y), r(Y,Z) \rightarrow r(X,Z)$ computes the transitive closure of a relation.

- *Add a normal edge* $(r[i], s[j])$ *to* E_N*, if there exists a TGD* $\sigma \in \Sigma$*, such that relation* r *occurs in the body of* σ *and relation* s *occurs in the head of* σ*, and the positions* $r[i]$ *and* $s[j]$ *are occupied by the same (universally quantified) variable in* σ*.*
- *Add a special edge* $(r[i], s[j])$ *to* E_S*, if there exists a TGD* $\sigma \in \Sigma$*, such that relation* r *occurs in the body of* σ *and relation* s *occurs in the head of* σ*, the (universally quantified) variable at position* $r[i]$ *in* σ *also appears in the head, and an existentially quantified variable appears at position* $s[j]$ *in the head of* σ*.*

The following example illustrates this notion.

Example 3.14. *Let schema* $\mathcal{R}$ *contain the relations* r, s, p *and* t*. Consider the following set* Σ *of TGDs over* $\mathcal{R}$*, consisting of three TGDs:*

$$\begin{aligned} r(V, W, X, Y) &\rightarrow \exists Z\, s(W, Z), p(W, Y) \\ p(W, X), s(W, Y) &\rightarrow t(Y, X) \\ t(X, Y) &\rightarrow p(X, Y) \end{aligned}$$

By looking at the structure of the TGDs, we can see that there the second and third TGD are recursive. However, while in this simple example it might be obvious to the trained eye, in general it is not straightforward to determine whether such a recursion might lead to an infinite chase expanstion. The dependency graph G *of* Σ *can help in this regard. It is depicted in Figure 3.2 below:*

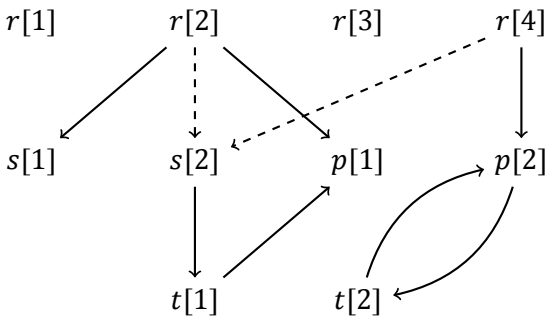


Figure 3.2: Dependency graph for Example 3.14

On inspection, the dependency graph is cyclic, which confirms our previous observation that the set of TGDs is recursive. However, it is immediately apparent that no cycle passes through a special edge, that is, there is no recursion through an existential variable. Therefore, the dependency graph allows us to determine that the set Σ *of TGDs will never lead to an infinite chase expansion.*

As we have seen, a sufficient condition that guarantees chase termination for a set of TGDs is that the dependency graph does not reveal any recursions through existential variables, which are represented by special edges. We can now give the formal definition of weakly-acyclic sets of TGDs.

Definition 3.15. *A set* Σ *of TGDs is called* weakly-acyclic *if and only if its dependency graph does not contain a cycle through a special edge.*

Given a set of TGDs, it is possible in PTime to decide whether it is weakly-acyclic or not: the dependency graph can be constructed in polynomial time, and checking for cycles through at least one special edge is also known to be feasible in PTime.

Fagin et al. [2005] show that weak acyclicity is a sufficient condition for chase termination. As the chase expansion is finite, we immediately get decidability of the CQ-Answering problem. Intuitively, the normal edges in the dependency graph keep track of the possible propagation of a value from one position to another, while the special edges keep track of the fact that whenever such a value is propagated, the chase also introduces a new, labelled null value into all those positions where an existentially quantified variable occurs. If a cycle goes through a special edge, then the occurrence of a null value in a certain position may cause the appearance of a new null value in the same position later on in the chase. This cycle may continue forever. Cycles through normal edges are however allowed. Weakly-acyclic sets of TGDs capture full TGDs, because full TGDs lack existential quantifiers and thus no special edges ever occur in the dependency graph.

CQ-Answering under weakly-acyclic sets of TGDs is PTime-complete in the data complexity, and 2ExpTime-complete in the combined complexity. These upper bounds are already implicit in [Fagin et al., 2005]. Roughly, this follows from the fact that the (finite) chase expansion of a database under a weakly-acyclic set of TGDs can be built in polynomial time, if the set of TGDs is fixed, and in double-exponential time in general. The PTime lower bound with respect to the data complexity follows immediately from the same hardness result for full TGDs, which are, as discussed in the previous paragraph, also weakly-acyclic. The 2ExpTime lower bound was established by Calì et al. [2011], where it is shown that it is possible to arrange double-exponentially many labelled nulls in an order, and thus simulate a 2ExpTime Turing machine, using a weakly-acyclic set of TGDs.

3.2.3 Summary

One attempt to obtain decidability of query answering is to try and limit their power to express recursions in such a way that every recursion is bound to stop after finitely many steps. The set of TGDs is, in a sense, acyclic. As recursion is an important tool to express knowledge, banning it altogether seems too strict. However, there are other approaches, surveyed in the previous sections, that yield reasonably expressive languages while still retaining the core feature that recursions, and thus ultimately the chase expansion, remains finite. We have surveyed the language of full TGDs or, equivalently, Datalog (see, e.g. Abiteboul et al. [1995]; Beeri and Vardi [1981]), as well as the language of weakly-acyclic TGDs, introduced in the landmark paper by Fagin et al. [2005]. A comparison of expressivity, including the previously discussed guarded-based classes of TGDs, can be found below in Figure 3.3. Here, FTGDs are full TGDs, WATGDs

are weakly-acyclic sets of TGDs, and as before, IDs are inclusion dependencies, LTGDs are sets of linear TGDs, GTGDs are sets of guarded TGDs, FGTGDs are sets of frontier-guarded TGDs, WGTGDs are weakly-guarded sets of TGDs, and WFGTGDs are weakly-frontier-guarded sets of TGDs.

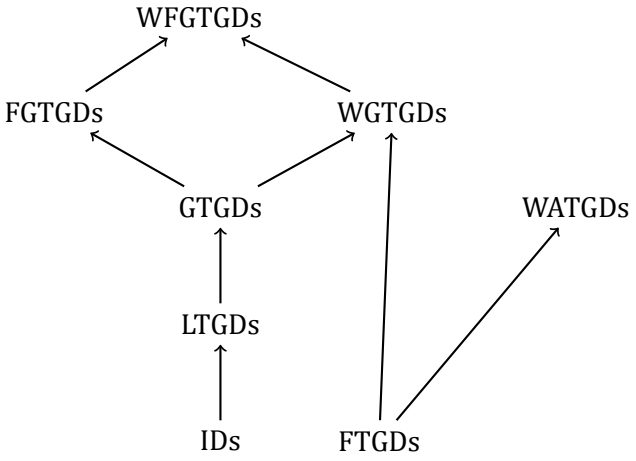


Figure 3.3: Expressivity comparison of guarded- and acyclicity-based classes of TGDs.

Since the classes of full TGDs and weakly-acyclic sets of TGDs both guarantee chase termination, we have that in those cases query answering becomes immediately decidable. It is sufficient to construct the (finite) chase expansion, and then, in finite time, evaluate the query over it. The complexity of the CQ-Answering problem is PTime-complete in the data complexity for both classes, and, in the combined complexity, ranges from ExpTime-completeness for full TGDs to 2ExpTime-completeness for weakly-acyclic sets of TGDs. Table 3.2 provides an overview of complexity results.

Class	Data Complexity	Combined Complexity
FTGDs	PTime-complete	ExpTime-complete
WATGDs	PTime-complete	2ExpTime-complete

Table 3.2: The complexity of CQ-Answering under acyclicity-based classes of TGDs.

3.3 Sticky-based Classes of TGDs

In this section, we will survey classes of TGDs that are based on a principle called *Stickiness*. This principle is an abstract property, that gives rise to a family of sticky-based languages. It was introduced by Calì et al. [2012b] and thoroughly disseminated in the doctoral thesis by Pieris [2011]. Sticky-based languages possess the property whenever the application of some rule with repeated variables leads to a join on some symbol s during the chase, from thereon out all of the descendant atoms of the newly derived atom must always contain that null value, that is, every rule that is applied subsequently to a descendant atom must propagate

the null value. This abstract property is called the *sticky property* of the chase, which is formally defined as follows:

Definition 3.16. *Given a database D and a set Σ of TGDs over schema $\mathcal{R}$, suppose that during the construction of the chase, the rule $\sigma \in \Sigma$, containing a repeated variable X in the body, is applied via a homomorphism h, and the atoms $\underline{a}_1, \ldots, \underline{a}_k$ are thereby generated, where $k = |head(\sigma)|$. Then, $chase(D, \Sigma)$ has the* sticky property, *if for each $\underline{a} \in \{\underline{a}_1, \ldots, \underline{a}_k\}$ the symbol $h(X)$ occurs in $\underline{a}$ and all its descendants.*

This central, abstract property is subsequently relied upon to derive decidability of query answering. A sufficient syntactic condition that defines the class of *sticky sets of TGDs* is presented in [Calì et al., 2012b]. In [Calì et al., 2011], both the sticky property as well as the corresponding syntactic condition are relaxed in order to find more expressive classes that capture linear TGDs, and weakly-acyclic TGDs. The relevant classes of TGDs are discussed in turn in the following subsections.

3.3.1 Sticky Sets of TGDs

Sticky sets of TGDs are defined via a global, syntactic restriction placed on sets of TGDs. The aim of this restriction is to guarantee that the respective sets of TGDs possess the sticky property discussed earlier. Essentially, given a set Σ of TGDs, the idea is to require that, whenever a position may contain a symbol that was propagated via a repeated variable in the body of some rule in Σ (i.e. by a join), this position must always be propagated by all TGDs in Σ. While this may be overly restrictive, it is clearly a sufficient condition to guarantee the sticky property. The following example illustrates this restriction:

Example 3.17. *The following set of TGDs is sticky, and thus possesses the sticky property:*

$$\begin{aligned} emp(W, X, Y, Z) &\rightarrow \exists V\ dept(X, V), runs(X, Z), inarea(Z, Y) \\ runs(X, Y), inarea(Y, Z) &\rightarrow \exists W\ external(W, Z, Y) \\ external(X, Y, Z) &\rightarrow \exists W location(Z, W) \end{aligned}$$

Upon inspection, it turns out that the second TGD contains a join. We now have to verify that the join variable is propagated, and that any position coming to hold the join symbol must always be propagated in all TGDs.

It is easy to verify that the join variable X is indeed propagated in the second TGD. This means that position $external[3]$ holds a join symbol. It can now be verified that, whenever the relation $external$ occurs in a TGD, position $external[3]$ is always propagated. We thus have verified that the set of TGDs given above is indeed sticky, as stated.

In more involved examples, verifying the stickiness condition can be a tedious task. Calì et al. [2012b] propose a marking procedure that verifies stickiness and is in fact used to define the syntactic property itself. This procedure, called SMarking, is defined below:

Definition 3.18. SMarking *takes a set* Σ *of TGDs as input and returns a set of marked TGDs where for each marked TGD, every body variable is either marked, or not. The procedure works in two steps, where the second step is exhaustively applied until a fixpoint is reached.*

Initial Marking Step *For each TGD* $\sigma \in \Sigma$ *and each variable* V *in* $body(\sigma)$*, if there exists an atom* $\underline{a}$ *in* $head(\sigma)$ *that does not contain* V*, mark* V *in* σ*.*

Propagation Step *For each TGD* $\sigma \in \Sigma$ *and each variable* V *in* $body(\sigma)$*, if there exists a* $\sigma' \in \Sigma$ *such that* $head(\sigma)$ *and* $body(\sigma')$ *both contain a relation* r *and the positions where* V *appears in* r *in* $head(\sigma)$ *coincide exclusively with positions of marked variables in* $body(\sigma')$*, then mark* V *in* σ*.*

When SMarking is applied to the TGDs in Example 3.17 above, the variable W would be marked in the first TGD, the variable X in the second, and the variables X and Y in the third. With such a marking, it is now easy to verify whether a given set of TGDs is sticky.

Definition 3.19. *A set* Σ *of TGDs is* sticky *if there is no TGD* $\sigma \in$ SMarking(Σ) *such that a marked variable occurs in* $body(\sigma)$ *more than once.*

The procedure SMarking runs in polynomial time in the worst case, and it is thus possible to decide in PTime whether a set of TGDs is sticky or not. Note that IDs, by definition, cannot have repeated variables in the body and are thus trivially sticky sets of TGDs.

Regarding the complexity of the CQ-Answering problem under sticky sets of TGDs, in [Calì et al., 2012b] it is established that sticky sets of TGDs are first-order rewritable. Thus the data complexity is in AC_0. Regarding the other complexity cases, an alternating algorithm, called StickyQAns, deciding the query answering problem is devised in Pieris [2011]. It runs in time $NP^{\mathcal{C}}$, where $\mathcal{C}$ is APSpace in the combined complexity, and ALogSpace in case where the set of TGDs is fixed. As APSpace (resp. ALogSpace) coincide with ExpTime (resp. PTime), we have that CQ-Answering is in ExpTime in the combined complexity, and in NP in case the set of TGDs is fixed.

As for the corresponding lower bounds, [Calì et al., 2011] establishes that the fact inference problem under lossless Datalog (i.e. Datalog where all body variables also occur in the head) can be reduced to CQ-Answering under sticky sets of TGDs. This problem is shown to be ExpTime-complete in Pieris [2011]. The NP-hardness result follows directly from the CQ-Containment problem without constraints; see Johnson and Klug [1984].

Sticky-Join Sets of TGDs Obviously, sticky sets of TGDs are not expressive enough to capture linear rules (e.g. the TGD $\sigma : r(X, Y, X) \rightarrow \exists Z\, s(Y, Z)$ is not sticky). In [Calì et al., 2011], sticky sets of TGDs are extended to *sticky-join* sets of TGDs, that allow linear TGDs (i.e. self-joins inside the same, single atom of the body, where the join variable is not propagated). Also the abstract sticky property is extended to the *sticky-join property*, that differs from the sticky

property insofar as that it does not require repeated variables, but repeated variables in a join across two different atoms.

Calì et al. [2011] then show that the complexity of the CQ-Answering problem is preserved by this extension, and thus remains in AC_0 in the data complexity, NP-complete in case of fixed sets of TGDs, and ExpTime-complete in the combined complexity.

However, as the syntactic condition imposed on sticky-join sets of TGDs is much more involved than the original sticky condition, deciding whether a set of TGDs is in fact sticky-join becomes a PSpace-complete problem.

3.3.2 Weakly-Sticky Sets of TGDs

In [Calì et al., 2011], the authors proceed to extend the class of sticky sets of TGDs to be able to express weakly-acyclic rules. The resulting class is called weakly-sticky. The idea is to slightly change the definition of the sticky condition in such a way that for a set Σ of TGDs, losing a join symbol s is allowed if the position holding s is in a part of the dependency graph of Σ that is not reachable from a cycle through a special edge. This ensures that, at the "lossy" position, only a finite number of symbols can occur, thus preserving decidability of query answering. The following example illustrates this point:

Example 3.20. *The following set Σ of TGDs is weakly-sticky:*

$$\begin{aligned} r(X,Y), r(Y,Z) &\rightarrow r(X,Z) \\ r(X,Y) &\rightarrow s(X,Y) \\ s(X,Y) &\rightarrow \exists Z\, s(Y,Z) \end{aligned}$$

Note that in the above set of TGDs, we are allowed to lose join symbols, as is the case in the first TGD, and thus the set Σ is not sticky. However, we can be sure that, at positions $r[1]$ and $r[2]$, only a finite number of symbols can occur, which can be easily verified using the dependency graph. The graph for Σ can be seen below in Figure 3.4.

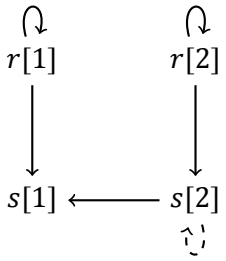


Figure 3.4: Dependency graph for Example 3.20

From the dependency graph above, it is clear that the set of TGDs is not weakly-acyclic. We can however see that only positions $s[1]$ and $s[2]$ can contain infinitely many symbols. This means that the part of the chase that does not exhibit the sticky property (namely the part built by those TGDs of Σ involving relation r) is finite. Decidability is thus preserved.

The formal definition of the syntactic class of weakly-stickiness is the same as for stickiness (see Definition 3.19) with a minor adaptation:

Definition 3.21. *A set* Σ *of TGDs is* weakly-sticky *if there is no TGD* $\sigma \in \mathsf{SMarking}(\Sigma)$ *such that a marked variable* V *occurs in* $body(\sigma)$ *more than once and a position where* V *occurs is reachable from a cycle through a special edge in the dependency graph of* Σ*.*

Regarding the complexity of the CQ-Answering problem under weakly-sticky sets of TGDs, in [Pieris, 2011] it is shown that weakly-sticky sets of TGDs are not first-order rewritable. In fact, the data complexity is in shown to be in PTime. This is shown using an ALogSpace=PTime algorithm, called WeaklyStickyQAns, deciding the query answering problem. In case where the query is not fixed, it runs in time $\text{NP}^{\mathcal{C}}$, where $\mathcal{C}$ is AExpSpace in the combined complexity, and ALogSpace in case where the set of TGDs is fixed. As AExpSpace (resp. ALogSpace) coincide with 2ExpTime (resp. PTime), we have that CQ-Answering is in 2ExpTime in the combined complexity, and in NP in case the set of TGDs is fixed.

As for the corresponding lower bounds, the PTime-hardness in the data complexity comes from a reduction to the fact inference problem under fixed Datalog programs; see, e.g. Dantsin et al. [2001]. The 2ExpTime-hardness in the combined complexity can be directly inherited from weakly-acyclic sets of TGDs (see Section 3.2.2), and in case where the set of TGDs is fixed, as with sticky sets of TGDs, we again get the NP-hardness result from the CQ-Containment problem without constraints; see Johnson and Klug [1984].

Weakly-Sticky-Join Sets of TGDs In the same manner as the stickiness condition can be extended to sticky-join sets of TGDs to allow for linear TGDs, weakly-stickiness can be extended to the class of so-called *weakly-sticky-join* sets of TGDs. Also in this case, the complexity of the CQ-Answering problem is preserved, and thus the same as for weakly-sticky sets of TGDs. However, once more the problem of deciding whether a set of TGDs fulfills the weakly-sticky-join condition becomes PSpace-hard.

3.3.3 Summary

The last idea to obtain decidability of query answering surveyed in detail in this chapter is to impose limits on which symbols can be "lost" by the application of TGDs during the construction of the chase. The sticky property (and the extended sticky-join property) provide a useful but abstract condition. The idea of stickiness is to prohibit symbols that have participated in a join to be lost again later on in the chase. As shown in [Calì et al., 2012b; Calì et al., 2011; Pieris, 2011], where stickiness is introduced, this condition is sufficient to provide decidable query answering. Also in these works, the complexity of the query answering problem is studied and algorithms for evaluating queries are proposed.

A comparison of expressivity, including the previously discussed guarded-based, as well as the acyclicity-based classes of TGDs, can be found below in Figure 3.5. Here, STGDs are

sticky sets of TGDs, SJTGDs are sticky-join sets of TGDs, WSTGDs are weakly-sticky sets of TGDs and WSJTGDs are weakly-sticky-join sets of TGDs. Furthermore, as before, FTGDs are full TGDs, WATGDs are weakly-acyclic sets of TGDs, IDs are inclusion dependencies, LTGDs are sets of linear TGDs, GTGDs are sets of guarded TGDs, FGTGDs are sets of frontier-guarded TGDs, WGTGDs are weakly-guarded sets of TGDs, and WFGTGDs are weakly-frontier-guarded sets of TGDs.

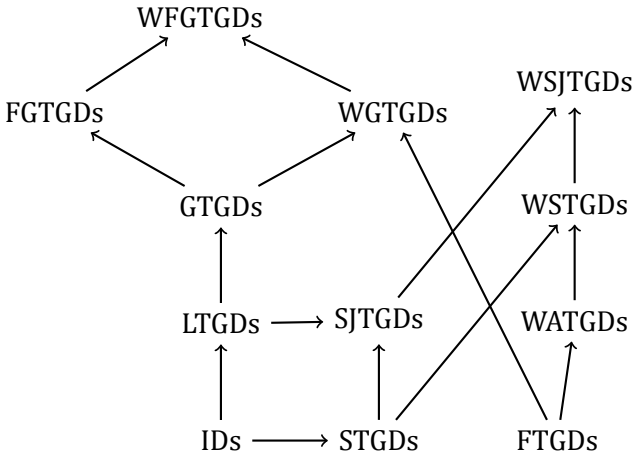


Figure 3.5: Expressivity comparison of decidable classes of TGDs.

To conclude this section, we give an overview of the complexity of the CQ-Answering problem, which for the data complexity is either in AC_0 for the non-weakly classes, or PTime-complete otherwise. In case of a fixed set of TGDs, query answering is NP-complete, whereas in the combined complexity the problem difficulty ranges from ExpTime-completeness to completeness for the complexity class 2ExpTime. Table 3.3 provides an overview of complexity results.

Class	Data Complexity	Combined Complexity
STGDs	in AC_0	ExpTime-complete
SJTGDs	in AC_0	ExpTime-complete
WSTGDs	PTime-complete	2ExpTime-complete
WSJTGDs	PTime-complete	2ExpTime-complete

Table 3.3: The complexity of CQ-Answering under sticky-based classes of TGDs.

3.4 Further Discussion

In this chapter, we have so far discussed three well-established paradigms for decidability that will form the basis for the complexity investigations in the main chapters of this thesis. In this section, we will quickly outline other paradigms and classes that have been discussed in the literature, but that are out of scope for this thesis.

Abstract Properties. One abstract property that guarantees decidability of query answering is the class of *finite expansion sets*. It was first proposed in the context of *conceptual graphs* by Baget and Mugnier [2002]. This can however be reformulated to apply to the chase, and coincides with those cases where the chase procedure terminates and thus the chase expansion is finite. Another such abstract property is the class of *finite treewidth sets*, that roughly guarantee that the chase expansion has finite treewidth. The observation that this leads to decidability of query answering is already implicit in [Calì et al., 2013], and is explicitly stated in [Baget et al., 2011a]. A third notion is the class of *first-order rewritable sets* of TGDs. Originally introduced in the area of Description Logics (see, e.g. Calvanese et al. [2007]), for a class that is first-order rewritable, the CQ-ANSWERING problem immediately falls into AC_0 in the data complexity, as established by Vardi [1995]. First-order rewritable sets are closely related to *finite unification sets* [Baget et al., 2009], which require that for a set of TGDs the perfect rewriting using backward-chaining unification must be finite.

Syntactic Properties. Several syntactic criteria ensuring decidability, apart from those discussed earlier in this chapter, have been proposed in the literature. Most notably, Leone et al. [2012] introduce the class of *shy sets of TGDs* that narrow down and use this adapted notion of affected positions, surveyed in Section 3.1.5, to guarantee decidability and practical feasibility of query answering. Also, recently the class of *tame sets of TGDs* was introduced by Gottlob et al. [2013b], where guardedness and stickiness paradigms are combined. The possibility of combining the notions of guardedness and weak acyclicity to obtain more expressive classes of TGDs was studied by Krötzsch and Rudolph [2011]. In [Calì et al., 2012], guarded TGDs are enriched with *stratified negation*, a simple non-monotonic form of negation in rules, classically used in the context of Datalog. The class of *domain-restricted TGDs*, which guarantees first-order rewritability, is investigated in [Baget et al., 2009].

Apart from the ones discussed in Section 3.2, other syntactic restrictions for acyclicity have been proposed that guarantee chase termination. Apart from the already discussed paradigm of weak acyclicity [Fagin et al., 2005], the notion of a *stratified witness* [Deutsch and Tannen, 2003], and, more recently, the notion of *super weak acyclicity* [Marnette, 2009] were introduced. Intuitively, each of these restrictions guarantees that there is no cycle between creation and propagation of null values in the chase, and thus the chase terminates. The notion of weak acyclicity has been used as a building block for larger, more expressive classes; for example a class based on *stratification* [Deutsch et al., 2008], one based on *inductive restriction* [Meier et al., 2009], and one based on *rewriting* [Spezzano and Greco, 2010].

Eiter and Simkus [2010] present the class of *FDNC logic programs* that allow for function symbols (F), disjunction (D), negation under stable model semantics (N) and constraints (C), while decidability of the standard reasoning tasks is preserved. They also show that consistency checking and brave reasoning under this language are both EXPTIME-complete, and they investigate restricted versions of the language with lower complexity.

4 Guarded-based Classes of DTGDs

With this chapter, we will begin in earnest our investigation of the impact of disjunction on reasoning under existential rules. To this end, we study the complexity of the query answering problem, and from the comparison of the complexity results with and without disjunction hope to provide an insight into how much it "costs" in terms of complexity to allow disjunction when reasoning under existential rules.

As we have discussed in the previous chapters, query answering under TGDs is in general undecidable. In this chapter, we will thus focus on adding disjunction to the main guarded-based classes of TGDs that were introduced in Section 3.1. As discussed in that chapter, guarded-based classes of TGDs exhibit the finite treewidth model property that guarantees decidability. We will see that this in fact holds also for guarded-based classes of DTGDs, and thus decidability is ensured.

We will also see that, as opposed to sticky-based and acyclicity-based classes of DTGDs that we will investigate in later chapters, for the guarded-based classes of DTGDs it is necessary to investigate the different query languages considered in this thesis separately. Indeed, after some general observations in the first two sections, the present chapter is split into a section for each query language studied, which outlines the complexity investigation for that particular type of query.

The main results of this chapter can be briefly summarized as follows: for the most expressive, guarded-based classes of DTGDs, we find that the impact of allowing disjunction is usually negligible in terms of the complexity of the query answering problem. However, there is a large complexity jump for the weaker classes like disjunctive IDs (DIDs) and linear DTGDs. For arbitrary queries, we are able to show that the complexity of query answering under fixed sets of rules jumps from co-NP-complete for sets of IDs to 2ExpTime-complete for sets of DIDs. Indeed, 2ExpTime is in fact the highest complexity we encounter, also in the combined complexity. We then show that reducing the expressive power of the query language also reduces the complexity in a reasonable way, as long as we forbid cyclicity in the query. In fact, even bounded (hyper-)treewidth queries still exhibit the same complexity as arbitrary queries.

In order to start our complexity investigation, below, in the first section of this chapter we will discuss how to extend guarded-based sets of TGDs with disjunction.

4.1 Extending Guarded-based Sets of TGDs with Disjunction

In order to extend the guarded-based classes of TGDs, discussed in Section 3.1, with disjunction, we need to adapt the introduced deterministic notions of rules to allow for disjunction in the rule heads. We start of by extending IDs to disjunctive IDs, or DIDs:

Definition 4.1. *A DTGD σ is called a* Disjunctive Inclusion Dependency (DID) *if it is of the form*

$$r(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists Y_i \, s_i(\mathbf{X}, \mathbf{Y}),$$

where $n \geq 1$, r and s_i are relations, and $\mathbf{X}$ and $\mathbf{Y}_i$ are pairwise disjoint sequences of variables, where each variable only appears once.

As we can see, the extension of IDs to DIDs is straightforward. It turns out that for linear, guarded and frontier-guarded TGDs, the extension to DTGDs is also trivial:

Definition 4.2. *A DTGD σ is called* linear *(resp.* guarded*; resp.* frontier-guarded*), if it is of the form*

$$\phi(\mathbf{X}, \mathbf{Y}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Z}_i \, \psi_i(\mathbf{Y}, \mathbf{Z}_i),$$

where $n \geq 1$, ϕ is a single atom $\alpha(\mathbf{X}, \mathbf{Y})$ (resp. a conjunction of atoms containing a guard atom $\alpha(\mathbf{X}, \mathbf{Y})$; resp. a conjunction of atoms containing a guard atom $\alpha(\mathbf{Y})$), and ψ_i are conjunctions of atoms. Atom $\underline{\alpha}$ is called the guard (resp. frontier-guard).

As we can see from the definition above, the notion of the guard, resp. frontier-guard translates directly to DTGDs. For completeness reasons, we will also treat multi-linear DTGDs in this chapter. As described in Chapter 3, the only difference w.r.t. linear rules is that there may be multiple body atoms, all of which have to be guards.

It remains to define the notion of weakly-(frontier-)guarded sets of DTGDs. Recall that these classes are defined via the notion of an *affected position*. In order to extend the classes to DTGDs, we thus first need to extend the definition of an affected position. The intuitive meaning of an affected position is that a null value can appear at such a position during the (deterministic) chase. For the disjunctive chase, we will require that a null value can appear at an affected position if it may appear in any one model of the disjunctive chase. The formal definition of this extension is as follows:

Definition 4.3. *Given a set of DTGDs Σ over a relational schema $\mathcal{R}$, the set of* affected positions*, denoted $affected_{\Sigma}(\mathcal{R})$, is defined inductively as the least fixed point of the following:*

- *Let $r[i]$ be a position in the head of a DTGD $\sigma \in \Sigma$. If an existentially quantified variable appears at position $r[i]$ in σ, the $r[i]$ is affected.*

- *Let $r[i]$ be a position in the head of a DTGD $\sigma \in \Sigma$. If a universally quantified variable appears at position $r[i]$, that variable only appears at affected positions in the body of σ, then $r[i]$ is affected.*

Having extended this definition to also capture DTGDs, we can now define the classes of weakly-(frontier-)guarded sets of DTGDs. The formal definition is as follows. Recall that the frontier $fr(\sigma)$ of a DTGD σ is the set of universally quantified variables that appear both in the head and the body of σ.

Definition 4.4. *A set Σ of TGDs is* weakly-guarded *(resp. weakly-frontier-guarded) if for each TGD $\sigma \in \Sigma$ it holds that all universally quantified variables (resp. all variables of $fr(\sigma)$) that appear only at affected positions in the body of σ appear together in a body atom. This atom is called the* weak-guard *(resp.* weak-frontier-guard*).*

We now have the relevant definitions in place to study the query answering problem under guarded-based classes of DTGDs. In fact, the extension of the guarded-based classes with disjunction is fairly straightforward. In the next section, we will look at the query answering problem, and how we can simplify our analysis.

4.2 Query Answering under Guarded-based Sets of DTGDs

In order to clarify the complexity picture of the query answering problem, we generally want to study four problems: the CQ-Answering problem, the B(H)TWQ-Answering problem, the ACQ-Answering problem, and the CQ_1-Answering problem, as well as their respective versions with unions of queries. It is thus beneficial to study, for a given class of DTGDs, whether any of these problems coincide (i.e. where we can give a polynomial time, or even logarithmic space reduction from the more expressive to the less expressive query language). It is the aim of this section to shed light on this issue.

Unfortunately, it turns out that all of the above problems are distinct, except for the case of (weakly-)frontier-guarded sets of DTGDs. In the following lemma, we will show that in fact for this class, all of the above problems coincide, both in the combined and in the data complexity.

Lemma 4.5. *The following problems, under (weakly-)frontier-guarded sets of TGDs (and thus also DTGDs), are LogSpace-equivalent in both the combined and the data complexity:*

1. (U)CQ-Answering
2. (U)B(H)TWQ-Answering
3. (U)ACQ-Answering
4. (U)CQ_1-Answering

Proof. As atomic queries are acyclic which are in turn of bounded (hyper-)treewidth and thus a subclass of arbitrary CQs, the steps (4) ⇒ (3) ⇒ (2) ⇒ (1) are trivial. It thus only remains to show (1) ⇒ (4), that is that, under (weakly)-frontier-guarded sets of TGDs, UCQ-Answering can be reduced to CQ_1-Answering.

In order to show this, let $\langle D, \Sigma, Q\rangle$ be an instance of the UCQ-Answering problem under (weakly-)frontier-guarded sets of TGDs over schema $\mathcal{R}$. We will construct another instance $\langle D, \Sigma', q'\rangle$ of the CQ_1-Answering problem, such that $D \cup \Sigma \models q$ if and only if $D \cup \Sigma' \models q'$. To this end, it is relevant to note that (weakly-)frontier-guarded sets of TGDs allow for arbitrary conjunctive queries to be embedded into rule bodies, as long as none of the relevant variables are propagated to the head—recall that only the frontier of a rule needs to be guarded.

In fact, the reduction is straightforward. Let $Q = q_1 \vee \ldots q_n$, where q_i is a disjunct in Q. Let $\mathcal{R}' = \mathcal{R} \cup \{p\}$, where p is a propositional relation. Let $\Sigma' = \Sigma \cup \{q_i \rightarrow p \mid 1 \leq i \leq n\}$, that is, the original set Σ and a set of *query-rules* that represent the query Q. It is indeed easy to verify that, as desired, $D \cup \Sigma \models Q$ if and only if $D \cup \Sigma' \models q$.

It remains to show that Σ' is in fact (weakly-)frontier-guarded. However, this is straightforward to see: in every rule $\sigma \in \Sigma'$, the frontier-variables must be guarded. By assumption, the original theory Σ was (weakly-)frontier-guarded. But for all the query-rules that we added to Σ, the frontier is empty (i.e. no variable is propagated from body to head). The addition of such rules can therefore never cause the theory to become non-(weakly-)frontier-guarded.

It is easy to see that this reduction can be completed in LogSpace, and that, if the set Σ and the query Q are fixed, it yields a fixed set Σ' and a fixed atomic query q. It thus works both in the combined and in the data complexity as desired, which completes the proof. □

As we have seen, for weakly-frontier-guarded sets of DTGDs, as well as sets of frontier-guarded DTGDs, we have that the different versions of the query answering problem are indeed LogSpace-equivalent, both in the combined and in the data complexity. Looking at the above reduction, we even see that it works in case where the arity is bounded by a constant, as the arity of the schema is never modified, and only a propositional (arity zero) relation is introduced.

Unfortunately, for the other guarded-based classes of DTGDs, we cannot show such a result. However, it turns out that we can at least show that the problem of answering CQs and UCQs of a particular type are equivalent. Before we show the formal result, let us give an intuitive explanation of how the transformation from UCQs to CQs works:

Each predicate p of the underlying schema is replaced with a new predicate p' with arity $arity(p) + 1$. This extra position holds a marker, either t (for *true*) or f (for *false*). Every database atom gets the value t at this new position, which implies that is a valid atom. Moreover, each DTGD is extended so that it simply propagates this position unaltered to the head.

A copy of each CQ in the given UCQ is added to the database (variables are replaced by new distinct constants) with f at the new position. Clearly, every CQ in the given UCQ trivially maps

to the database, with f at the new position. However, all the valid atoms, that is, the atoms that can be derived by the chase of the original database w.r.t. the original set of DTGDs, have t at the new position.

We can now replace disjunctions in the UCQ by conjunctions (and thus convert the UCQ into a CQ), and just check that for at least one query its new position maps to t. This can simply be done by adding to the database atoms of the form $or(\cdot,\cdot,\cdot)$ encoding the logical *or*, connecting all the subqueries of the original UCQ via such or predicates, and stating that the end result must be t. The formal result and its proof follow:

Lemma 4.6. *Consider a database D, a set Σ of DTGDs, and a UCQ Q over a schema $\mathcal{R}$. A database D', a set Σ' of DTGDs and a CQ q over a schema $\mathcal{R}'$, where $arity(\mathcal{R}') = arity(\mathcal{R}) + 1$, can be constructed in polynomial time such that $D \cup \Sigma \models Q$ if and only if $D' \cup \Sigma' \models q$.*

Proof. We assume, without loss of generality, that the CQs occurring as disjuncts in Q do not have any variables in common. In the rest of the proof, let t and f be constants of $\mathbf{C}$ not occurring in D. Given a tuple of terms $\mathbf{t}$, we denote by $\mathbf{t}_{\downarrow}$ the tuple of constant obtained after "freezing" $\mathbf{t}$, that is, after replacing each variable V in $\mathbf{t}$ with a new constant $c_V \in \mathbf{C}$ not occurring in $dom(D) \cup \{t, f\}$. We are now ready to give the formal reduction.

The Database D'. The new database D' is defined as follows:

$$\begin{aligned} & \{p'(\mathbf{t}, t) \mid p(\mathbf{t}) \in D\} \\ \cup\ & \{true(t), false(f), or(t,t,t), or(t,f,t), or(f,t,t), or(f,f,f)\} \\ \cup\ & \{p'(\mathbf{t}_{\downarrow}, f) \mid q \in Q \wedge p(\mathbf{t}) \in body(q)\}. \end{aligned}$$

The Set Σ'. The new set Σ' of DTGDs is obtained from Σ by replacing each atom of the form $p(\mathbf{X})$ occurring in a DTGD with the atom $p'(\mathbf{X}, T)$, where T is a new variable not occurring in Σ. For example, the DTGD $p(X,Y), s(Y,Z) \rightarrow \exists W\, p(X,W), r(W)$ will be replaced by the DTGD $p'(X,Y,T), s'(Y,Z,T) \rightarrow \exists W\, p'(X,W,T), r'(W,T)$.

The CQ q. Assume that $Q = q_1 \vee \ldots \vee q_n$. For a CQ $q_i \in Q$, we define $q_i[X_i]$ as the CQ obtained from q_i by replacing each atom of the form $p(\mathbf{t})$ with the atom $p'(\mathbf{t}, X_i)$, where X_i is a new variable not occurring in Q. The CQ q is defined as the query:

$$\exists X_1 \ldots \exists X_n \exists Y_1 \ldots \exists Y_{n+1} \left(false(Y_1) \wedge \bigwedge_{q_i \in Q} (q_i[X_i] \wedge or(Y_i, X_i, Y_{i+1})) \wedge true(Y_{n+1}) \right).$$

By construction, for each $q_i[X_i]$, there exists a homomorphism that maps $body(q_i[X_i])$ to the database D'. However, this fact does not imply that the query q is trivially entailed by $chase(D', \Sigma')$ since, in order to satisfy the atom $true(Y_{n+1})$, at least one X_i must be mapped

to the constant t. Thus, by construction, the only way to entail q is to map at least one sub-query $q_i[X_i]$ to $chase(D', \Sigma')$ via a homomorphism h, where also $h(X_i) = t$. Notice that the only atoms in $chase(D', \Sigma')$ containing t at the last position are the ones derived from the original copy of D. Thus, a sub-query $q_i \in Q$ is entailed by an instance $I \in chase(D, \Sigma)$ if and only if the corresponding instance I' of $chase(D', \Sigma')$ entails q. Since the above construction is feasible in polynomial time, the claim follows. □

With the above result, we have shown that at the cost of increasing the maximum arity of the schema by one, we can reduce the problem of answering unions of CQs to the problem of answering a single CQ. It is not difficult to see from the construction, that this holds in the combined complexity, if the arity, or the entire set of rules Σ is fixed, and also in the data complexity. As the newly constructed query q arranges the sub-queries $q_i \in Q$ into a tree, it is also not difficult to verify that if all the sub-queries q_i are acyclic, or of bounded (hyper-)treewidth, respectively, then the newly constructed query q also retains this property. Therefore, the above lemma holds, even if the query is either acyclic or of bounded (hyper-)treewidth.

This concludes our general investigation of the problem of query answering under guarded-based classes of DTGDs. In the next section, we will provide a brief overview over guarded-based fragments of first-order logic.

4.3 Guarded Fragments of First-Order Logic

In our later complexity analysis, we will exploit complexity results established for expressive guarded-based fragments of first-order logic. This section provides an overview of the relevant fragments we consider.

4.3.1 The Guarded Fragment of First-Order Logic

The *guarded fragment of first-order logic (GFO)*, introduced by Andréka et al. [1998], is a collection of first-order formulas with some syntactic restrictions in the quantification pattern, which is analogous to the relativized nature of modal logic. It is defined as follows:

Definition 4.7. *The* guarded fragment of first-order logic (GFO) *is the smallest set of formulas over a schema $\mathcal{R}$*

1. *containing all atomic $\mathcal{R}$-formulas and equalities;*
2. *closed under the logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$; and*
3. *if $\underline{a}$ is an $\mathcal{R}$-atom or an equality atom containing all the variables of $\mathbf{X} \cup \mathbf{Y}$, and ϕ is a GFO formula with free variables contained in $(\mathbf{X} \cup \mathbf{Y})$, then*

$$\forall \mathbf{X}\,(\underline{a} \rightarrow \phi) \qquad \textit{and} \qquad \exists \mathbf{X}\,(\underline{a} \wedge \phi)$$

are GFO formulas as well.

The complexity of the satisfiability problem or GFO formulas was investigated by Grädel [1999], who shows that the problem is 2ExpTime-complete in general and ExpTime-complete in case where the maximum predicate arity is fixed. Bárány et al. [2014] investigated the query answering problem under GFO sentences.

The *loosely-guarded fragment of first-order logic (LGFO)* is a generalization of GFO where the quantifiers are guarded by conjunctions of atomic formulas such that the quantifier-rule (3) is relaxed as follows:

(3) if $\underline{a}_1 \wedge \ldots \wedge \underline{a}_n$ is a conjunction of $\mathcal{R}$-atoms containing all the variables of $\mathbf{X} \cup \mathbf{Y}$, and for every pair of distinct variables $(V, W) \in \mathbf{X} \times (\mathbf{X} \cup \mathbf{Y})$ there exists at least one atom in $\{\underline{a}_1, \ldots, \underline{a}_n\}$ that contains both V and W, and ϕ is an LGFO formula with free variables contained in $(\mathbf{X} \cup \mathbf{Y})$, then

$$\forall \mathbf{X}\,((\underline{a}_1 \wedge \ldots \wedge \underline{a}_n) \rightarrow \phi) \qquad \text{and} \qquad \exists \mathbf{X}\,((\underline{a}_1 \wedge \ldots \wedge \underline{a}_n) \wedge \phi)$$

are LGFO formulas as well.

The same complexity results as for GFO are also shown to be valid for LGFO; see Grädel [1999]; Bárány et al. [2014].

4.3.2 Guarded Negation First-Order Logic

Guarded negation first-order logic (GNFO), introduced by Bárány et al. [2011], restricts first-order logic by requiring that all occurrences of negation are of the form $\underline{a} \wedge \neg\phi$, where $\underline{a}$ is an atom containing all the free variables of ϕ. Formally, the formulas of GNFO are defined as follows:

Definition 4.8. *Formulas of first-order logic belong to* Guarded Negation First-Order Logic (GNFO)*, if they are generated by the recursive definition*

$$\phi ::= p(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \exists X\,\phi \mid \underline{a} \wedge \neg\phi,$$

where each $t_i \in \mathbf{C} \cup \mathbf{V}$, and in the last clause, $\underline{a}$ is an atomic formula containing all free variables of ϕ.

It can be verified that GNFO is strictly more expressive than GFO. The complexity of the satisfiability problem was pinpointed as being 2ExpTime-complete in [Bárány et al., 2011], and query answering under GNFO sentences was investigated and shown to have the same complexity in [Bárány et al., 2012]. This concludes this section, introducing the guarded-based expressive logics we will use in our subsequent complexity investigation, mostly to obtain relevant upper bounds. The next four sections will, in turn and ordered by decreasing expressivity, deal with the different query languages we investigate, and contain a full complexity analysis of the query answering problem.

4.4 Decidability

The fragments of first-order logic, as well as the extended guarded-based classes of existential rules both exhibit the *finite treewidth model property*. A fragment $\mathcal{L}$ of first-order logic enjoys the *finite treewidth model property* if the following holds: for each formula $\phi \in \mathcal{L}$, whenever ϕ is satisfiable, then there exists a model of ϕ (which can be seen as a relational instance) of finite treewidth. By Courcelle's theorem [Courcelle, 1990], which states that any property definable in monadic second-order logic—and thus also first-order logic—over graphs of finite treewidth is decidable, we get decidability of the query answering problem.

Let us now give the following decidability result that establishes that all the classes of DTGDs we treat in this chapter are decidable.

Theorem 4.9. UCQ-ANSWERING *under weakly-frontier-guarded DTGDs is decidable.*

Proof. Let $\mathcal{L}_{WFG}$ be the fragment of first-order logic which can express only formulas of the form $\Phi_{D,\Sigma,Q} = (D \wedge \Sigma \wedge \neg Q)$, where D is a database, Σ is a weakly-frontier-guarded set of DTGDs, and Q is a UCQ. Since $D \cup \Sigma \models Q$ if and only if $\Phi_{D,\Sigma,Q}$ is unsatisfiable, it suffices to show that $\mathcal{L}_{WFG}$ enjoys the finite treewidth model property.

Consider an arbitrary formula $\Phi_{D,\Sigma,Q} \in \mathcal{L}_{WFG}$. By the universality property of the disjunctive chase, $D \cup \Sigma \models Q$ iff $I \models Q$, for each $I \in chase(D, \Sigma)$. Hence, $\Phi_{D,\Sigma,Q}$ is satisfiable iff there exists $I \in chase(D, \Sigma)$ such that $(I \wedge \neg Q)$ is satisfiable. Clearly, if $\Phi_{D,\Sigma,Q}$ is satisfiable, then there exists $I \in chase(D, \Sigma)$ which is a model of $\Phi_{D,\Sigma,Q}$. By definition of the disjunctive chase, the instance I can be seen as the result of the (non-disjunctive) chase under D and a set of weakly-frontier-guarded (non-disjunctive) TGDs. It is implicit in [Baget et al., 2011a], where weakly-frontier-guarded (non-disjunctive) TGDs are investigated, that I has finite treewidth, and the claim follows. □

The above theorem establishes decidability of (U)CQ answering under the guarded-based classes of DTGDs that we consider in this thesis. However, it tells nothing about the computational complexity of our problems. Understanding the complexity of the problems under consideration will be the subject of the next sections.

4.5 Answering Arbitrary Queries

In this section, we focus on answering (unions of) conjunctive queries under our respective classes of DTGDs. The overview section below contains a summary of our results, and discusses results inherited from existing work. Full proofs for our novel upper and lower complexity bounds are given in Section 4.5.2 and 4.5.3, respectively.

	Combined Complexity	Bounded Arity	Fixed Theory	Data Complexity
DID	2ExpTime	2ExpTime	2ExpTime LB: Thm. 4.18	co-NP LB: [Calvanese et al., 2013, Thm. 4.5]
L	2ExpTime	2ExpTime	2ExpTime	co-NP
ML	2ExpTime	2ExpTime	2ExpTime	co-NP
G	2ExpTime	2ExpTime	2ExpTime	co-NP
FG	2ExpTime	2ExpTime	2ExpTime	co-NP UB: Thm. 4.12
WG	2ExpTime	2ExpTime	2ExpTime	ExpTime LB: [Calì et al., 2013, Thm. 4.1]
WFG	2ExpTime UB: Thm. 4.10	2ExpTime	2ExpTime	ExpTime UB: Thm. 4.13

Table 4.1: The complexity of (U)CQ-Answering under guarded-based classes of DTGDs.

4.5.1 Overview

Table 4.1 summarizes the complexity results for answering (U)CQs under the various DTGD formalisms considered in this paper. Each row corresponds to a class of DTGDs (which is decoded by substituting L for linear, ML for multi-linear, G for guarded, F for frontier, and W for weakly), while each column corresponds to a different setting of the problem.

In each cell of the table, we have indicated where to find the corresponding results (UB and LB stands for upper and lower bound, respectively). Note that the missing references for the upper (resp. lower) bounds are immediately inherited from the first lower-left (resp. upper-right) cell in which a reference is given. The results of this section, together with a brief description of the employed techniques, follow:

Upper Bounds:

1. UCQ-Answering under weakly-frontier-guarded DTGDs is feasible in 2ExpTime in combined complexity (Theorem 4.10) — this is established by a reduction to the unsatisfiability problem of GNFO sentences which is 2ExpTime-complete, see [Bárány et al., 2011, Theorem 2];

2. UCQ-Answering under frontier-guarded DTGDs is in co-NP in data complexity (Theorem 4.12) — this is proved by a reduction to UCQ answering under GFO which is co-NP-complete in data complexity, see [Bárány et al., 2014, Theorem 5.1]; and

3. UCQ-Answering under weakly-frontier-guarded DTGDs is in ExpTime in data complexity (Theorem 4.13) — this is shown by a reduction to UCQ answering under GFO sentences with bounded arity, and then exploit the fact this problem is feasible in exponential time in the size of the sentence, see [Bárány et al., 2014, Theorem 5.1].

Lower Bounds:

1. CQ-Answering under DIDs is 2ExpTime-hard in combined complexity, even if we focus on predicates of arity at most two (Theorem 4.16) — this is established by a reduction from the validity problem of a CQ w.r.t. a non-deterministic tree automaton which we show to be 2ExpTime-hard (Proposition 4.15) by a reduction from the non-acceptance problem of an alternating exponential space Turing machine;

2. UCQ-Answering under a fixed set of DIDs is 2ExpTime-hard, even if we consider predicates of arity at most two (Theorem 4.17) — this is shown by a non-trivial adaptation of the construction in the proof of Theorem 4.16; and

3. CQ-Answering under a fixed set of DIDs is 2ExpTime-hard, even if we consider predicates of arity at most three (Theorem 4.18) — this is shown by exploiting Lemma 4.6, which states that UCQ-Answering under (arbitrary) DTGDs is reducible in polynomial time to CQ-Answering at the price of increasing the arity of the underlying schema by one, and then we exploit Theorem 4.17.

Clearly, the 2ExpTime upper bound for weakly-frontier-guarded DTGDs (the most expressive formalism considered here), and the 2ExpTime lower bound in the case of a fixed set of DIDs (the weakest formalism studied in this paper), close the picture of the computational complexity of our problem w.r.t. the combined complexity, the case of bounded arity, and the case of a fixed set of DTGDs.

Existing results provide us with two optimal lower bounds for the data complexity of our problem, which complete the entire picture of the complexity of (U)CQ answering.

Inherited Results:

1. CQ-Answering under DIDs is co-NP-hard in data complexity [Calvanese et al., 2013, Theorem 4.5] — in fact this result shows that CQ-Answering under a Description Logic TBox with a single axiom of the form $A_1 \sqsubseteq A_2 \sqcup A_3$, where each A_i is an atomic concept, which in turn is logically equivalent to the DID $A_1(X) \rightarrow A_2(X) \vee A_3(X)$, is co-NP-hard in data complexity; and

2. CQ-Answering under weakly-guarded DTGDs is ExpTime-hard in the data complexity [Calì et al., 2013, Theorem 4.1] — actually this result considers weakly-guarded TGDs (without disjunctive heads).

From the results of this section, we observe that for the expressive formalisms under consideration, namely weakly-(frontier-)guarded DTGDs, the addition of disjunction does not have a significant effect on the complexity of (U)CQ answering. However, for the less expressive formalisms, the impact can be enormous; for example, we observe a jump from NP-completeness to 2ExpTime-completeness in the case of a fixed set of DIDs.

Let us now proceed with the formal proofs of our results.

4.5.2 Upper Bounds

We start this section by showing that UCQ-Answering under weakly-frontier-guarded sets of DTGDs is in 2ExpTime in combined complexity. This is shown by a reduction to the problem of deciding whether a GNFO sentence is unsatisfiable.

Theorem 4.10. UCQ-Answering *under weakly-frontier-guarded sets of DTGDs is in 2ExpTime in the combined complexity.*

Proof. We provide a polynomial-time reduction to the problem of deciding whether a GNFO sentence is unsatisfiable, which in turn is in 2ExpTime; see Bárány et al. [2011]. Our reduction consists of two steps: (1) we reduce our problem to the problem of UCQ answering under frontier-guarded DTGDs; and (2) we show how the constructed set of frontier-guarded DTGDs can be equivalently rewritten as a GNFO sentence. Consider a database D, a set Σ of weakly-frontier-guarded DTGDs, and a UCQ Q. Our reduction now works as follows:

Step 1. The key idea is to exploit the fact that whenever the (disjunctive) chase applies a DTGD, by definition, in every model the *non-affected variables* (i.e. variables occurring at the non-affected positions) are mapped to constants of $\mathbf{C}$. It is therefore possible to partially ground the DTGDs in such a way that the non-affected variables are replaced by all possible combinations of constants, yielding an equivalent set of frontier-guarded DTGDs. However, such a transformation would be exponential in the worst case. To overcome this difficulty we adapt a technique proposed by Baget et al. [2011b] which allows us to simulate partial groundings with only polynomial blow-up.

Roughly speaking, instead of explicitly replacing the non-affected variables by constants, we extend each predicate of the underlying schema in such a way that it contains all m constants of the database in its first m positions, and a grounding of its n non-affected variables in the next n positions. Using a set of $|sch(\Sigma)| \cdot n \cdot m$ inclusion dependencies, each propagating one of the constants to one of the non-affected variables, we can simulate the partial grounding as desired. Using this procedure we can convert a weakly-frontier-guarded set of DTGDs into a set of frontier-guarded DTGDs in polynomial time. Let us now formalize the above informal description.

Given a DTGD $\sigma \in \Sigma$, $nav(\sigma)$ denotes the set of variables occurring in at least one position of $nonaffected_{\Sigma}(\Sigma)sch(\Sigma)$. Fix also a bijection $\#_{\sigma} : nav(\sigma) \rightarrow [|nav(\sigma)|]$. Let $C_1, \ldots, C_m$ and $N_1, \ldots, N_n$ be variables of $\mathbf{V}$ not occurring in Σ, where m is the number of constants $c_1, \ldots, c_m$

appearing in D, and $n = \max_{\sigma \in \Sigma}\{|nav(\sigma)|\}$. Given a term t, let $\tau_\sigma(t) = N_{\#_\sigma(t)}$ if $t \in nav(\sigma)$; otherwise, let $\tau_\sigma(t) = t$. The mapping τ_σ is extended to atoms as follows:

$$\tau_\sigma(p(t_1, \ldots, t_k)) \;=\; p(C_1, \ldots, C_m, N_1, \ldots, N_n, \tau_\sigma(t_1), \ldots, \tau_\sigma(t_k)).$$

Notice that τ_σ can naturally be defined for sets of atoms. We now define the function τ, mapping DTGDs of Σ to frontier-guarded DTGDs, as follows:

$$\tau(\sigma) \;=\; \tau_\sigma(body(\sigma)) \rightarrow \tau_\sigma(head(\sigma)).$$

Intuitively, the variables $C_1, \ldots, C_m$ will permanently hold the constants of $dom(D)$, while the variables $N_1, \ldots, N_n$ will be assigned a combination of those constants.

We define the set Σ' of frontier-guarded DTGDs as the set $\{\tau(\sigma) \mid \sigma \in \Sigma\} \cup \Sigma_C$, where Σ_C consists of the following IDs: $\bigcup_{p \in sch(\Sigma), i \in [m], j \in [n]} \{\sigma^p_{i,j}\}$, with

$$\sigma^p_{i,j} \;=\; p(C_1, \ldots, C_m, N_1, \ldots, N_n, X_1, \ldots, X_k) \rightarrow p(C_1, \ldots, C_m, N^\star_1, \ldots, N^\star_n, X_1, \ldots, X_k),$$

where $k = arity(p)$, $N^\star_j = C_i$, and $N^\star_k = N_k$ for each $k \neq j$. Roughly, Σ_C is responsible for generating all the possible combinations of the constants occurring in D.

Finally, let $D' = \{\tau'(\underline{a}) \mid \underline{a} \in D\}$ and $Q' = \{\tau'(\underline{a}) \mid \underline{a} \in Q\}$, where

$$\tau'(p(t_1, \ldots, t_k)) \;=\; p(c_1, \ldots, c_m, \underbrace{c_1, \ldots, c_1}_{n}, t_1, \ldots, t_k).$$

Let us clarify that the selection of the constant c_1 at positions $m+1, \ldots, m+n$ is an arbitrary one since, due to the IDs of Σ_C, all the combinations of constants of $dom(D)$ will eventually appear at those positions during the construction of the chase. By construction, $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models Q'$; this completes the first step of our reduction.

Step 2. Observe that a frontier-guarded DTGD σ of the form $\forall \mathbf{X}(\phi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y}))$ can be equivalently rewritten as the sentence $\Phi_\sigma = \neg(\exists \mathbf{X}(\phi(\mathbf{X}) \wedge \neg \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})))$, which falls in GNFO since all the free variables of $\exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})$, that is, the variables of $\mathbf{X}$, appear in the frontier-guard of $\phi(\mathbf{X})$. Moreover, given a CQ q, $\neg q$ trivially falls in GNFO since in q there are no free variables. From the above discussion, we conclude that the sentence $\Psi_{D', \Sigma', Q'} = (D' \wedge \bigwedge_{\sigma \in \Sigma'} \Phi_\sigma \wedge \bigwedge_{q \in Q'} \neg q)$ falls in GNFO. The claim follows since $D' \cup \Sigma' \models Q'$ iff $\Psi_{D', \Sigma', Q'}$ is unsatisfiable. □

Notice that an alternative way to obtain the above result is to reduce our problem to UCQ answering under GFO sentences, which is also in 2ExpTime; cf. Bárány et al. [2014]. However, the translation from frontier-guarded DTGDs to GNFO is straightforward, whereas the translation to GFO is more involved, as GFO imposes tighter syntactic restrictions on formulas.

We now focus on the data complexity of UCQ answering under (weakly-)frontier-guarded DTGDs. It is known that CQ answering under frontier-guarded TGDs can be reduced in linear time to UCQ answering under GFO sentences; cf. Baget et al. [2011b]. Interestingly, the same reduction (with minor adaptations) can be employed even if we consider frontier-guarded DTGDs:

Lemma 4.11. UCQ-Answering *under frontier-guarded DTGDs can be reduced in linear time to* UCQ-Answering *under GFO sentences.*

Proof. Consider a database D, a set Σ of frontier-guarded DTGDs, and a UCQ Q. For each $\sigma \in \Sigma$ of the form $\forall \mathbf{X}(\phi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}', \mathbf{Y}))$, where $\mathbf{X}' \subseteq \mathbf{X}$, we define

$$\tau_1(\sigma) \;=\; \forall \mathbf{X}(\phi(\mathbf{X}) \rightarrow p_\sigma(\mathbf{X}')) \qquad \text{and} \qquad \tau_2(\sigma) \;=\; \forall \mathbf{X}'(p_\sigma(\mathbf{X}') \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}', \mathbf{Y})),$$

where p_σ is an auxiliary $|\mathbf{X}'|$-ary predicate not occurring in $sch(\Sigma)$. Assuming that $g_\sigma(\mathbf{X}')$ is the frontier-guard of σ, $\tau_1(\sigma)$ can be equivalently rewritten as follows; in the sequel, p'_σ is an auxiliary $|\mathbf{X}'|$-ary predicate:

$$\begin{aligned}
\forall \mathbf{X}(\phi(\mathbf{X}) \rightarrow p_\sigma(\mathbf{X}')) \;&\equiv\; \neg \exists \mathbf{X}(\phi(\mathbf{X}) \wedge \neg p_\sigma(\mathbf{X}')) \\
&\equiv\; (\neg \exists \mathbf{X}(\phi(\mathbf{X}) \wedge p'_\sigma(\mathbf{X}'))) \wedge (\forall \mathbf{X}'(g_\sigma(\mathbf{X}') \wedge \neg p_\sigma(\mathbf{X}') \rightarrow p'_\sigma(\mathbf{X}'))) \\
&\equiv\; (\neg \underbrace{\exists \mathbf{X}(\phi(\mathbf{X}) \wedge p'_\sigma(\mathbf{X}'))}_{\Phi^1_\sigma}) \wedge (\underbrace{\forall \mathbf{X}'(g_\sigma(\mathbf{X}') \rightarrow p_\sigma(\mathbf{X}') \vee p'_\sigma(\mathbf{X}'))}_{\Phi^2_\sigma}).
\end{aligned}$$

We define the formulas:

$$\Psi^1_\Sigma \;=\; \left(\neg \bigvee_{\sigma \in \Sigma} \Phi^1_\sigma \right) \qquad \text{and} \qquad \Psi^2_\Sigma \;=\; \left(\bigwedge_{\sigma \in \Sigma} (\tau_2(\sigma) \wedge \Phi^2_\Sigma) \right).$$

It is not difficult to see that

$$D \cup \Sigma \models Q \qquad \text{iff} \qquad \left(D \wedge \Psi^1_\Sigma \wedge \Psi^2_\Sigma\right) \models Q \qquad \text{iff} \qquad \left(D \wedge \Psi^2_\Sigma\right) \models \left(Q \vee \Psi^1_\Sigma\right).$$

Ψ^2_Σ is a set of guarded DTGDs, and thus is equivalent to a GFO sentence Γ^2_Σ; hence, $(D \wedge \Gamma^2_\Sigma)$ is a GFO sentence. Since $(Q \vee \Psi^1_\Sigma)$ is a UCQ, the claim follows. □

By exploiting the above auxiliary result, we are now ready to establish the desired upper bounds for the data complexity of our problem.

Theorem 4.12. UCQ-Answering *under frontier-guarded DTGDs is in co-NP in the data complexity.*

Proof. The result follows from Lemma 4.11, and the fact that UCQ-Answering under GFO sentences is in co-NP in the data complexity, as shown by Bárány et al. [2014]. □

We now focus on weakly-frontier-guarded sets of DTGDs:

Theorem 4.13. UCQ-Answering *under weakly-frontier-guarded set of DTGDs is in ExpTime in the data complexity.*

Proof. Consider a database D, a set Σ of weakly-frontier-guarded DTGDs, and a UCQ Q. We first reduce our problem to UCQ answering under frontier-guarded DTGDs by replacing the non-affected variables in the DTGDs of Σ with all possible constants occurring in D. In other words,

we partially ground the set Σ, and we obtain a set Σ' of frontier-guarded DTGDs. Clearly, Σ' is of exponential size in the number of non-affected variables, but of polynomial size in $|dom(D)|$. By Lemma 4.11, there exists a linear translation τ such that $D \cup \Sigma' \models Q$ iff $D \cup \tau(\Sigma') \models \tau(Q)$, where $\tau(\Sigma')$ is a GFO sentence and $\tau(Q)$ a UCQ. It is important to say that, although $|\tau(Q)|$ depends on D, the size of each CQ in $\tau(Q)$ does not depend on D. As shown by Bárány et al. [2014], UCQ answering under GFO sentences depends doubly-exponentially on the size of each CQ of the given UCQ and the maximum arity of the schema, and exponentially on the size of the given GFO sentence. Since the size of each query of $\tau(Q)$ and the maximum arity of the schema are constant, while the size of $\tau(\Sigma')$ is polynomial in D, we get an ExpTime upper bound w.r.t. D, and the claim follows. □

Recall that CQ-Answering under DIDs is co-NP-hard in the data complexity [Calvanese et al., 2013], while for weakly-guarded sets of DTGDs it is ExpTime-hard [Calì et al., 2013]. Thus, the picture of the data complexity of CQ-Answering under our guarded-based classes of DTGDs is now complete.

4.5.3 Lower Bounds

We now present a series of strong 2ExpTime lower bounds for answering arbitrary CQs under DIDs. This will be done by a reduction from the validity problem of a CQ w.r.t. a nondeterministic tree automaton, where the $child$ and $parentorchild$ predicates can be used in the query to access the tree. The first step is thus to show that the latter problem itself is 2ExpTime-hard. We will accomplish this by adapting an existing result, that is, the validity problem of a CQ w.r.t. a nondeterministic tree automaton, using $child$ and $descendant$ predicates to access the tree, is 2ExpTime-hard [Björklund et al., 2008, Theorem 6].

The reason why, in the above result, we need to replace the predicate $descendant$ with the predicate $parentorchild$ is because the descendant relation on nodes is transitive. This means that, in order to be able to simulate it using DTGDs, we need a class of DTGDs which allows us to express the assertion $descendant(X, Y), descendant(Y, Z) \rightarrow descendant(X, Z)$. However, the formalism under consideration, namely DIDs, is not expressive enough to encode transitivity, and thus there is no way to simulate the descendant relation. Before giving our auxiliary technical result, for the sake of completeness, we recall the proof of the result given by Björklund et al. [2008]. Let us first introduce some additional terminology.

Trees, Queries and Tree Automata

Fix a finite alphabet Λ. We consider rooted, ordered, finite, labeled, unranked trees, which are directed from the root downwards. In other words, we focus on finite trees in which nodes can have arbitrarily many children ordered from left to right. The set of nodes of a tree T is denoted $nodes(T)$, while its root is denoted $root(T)$.

A CQ q over a tree T is a CQ that can use unary predicates from $\{a\}_{a\in\Lambda}$, and also binary predicates from a (finite) set $\mathcal{R}$ that allow the query to access the tree. We refer to this fragment of CQs as CQ($\mathcal{R}$). Possible predicates of $\mathcal{R}$ are $child$ and $descendant$. Clearly, $a(v)$ means that v is labeled by $a \in \Lambda$, while $child(v,u)$ (resp., $descendant(v,u)$) states that u is a child (resp., descendant) of v. A *valuation* of q on T is a total function $\theta : var(q) \to nodes(T)$. Such θ satisfies the query if every atom of q is satisfied by the assignment. We say that T *models* q, written $T \models q$, if there exists a valuation of q on T that satisfies q. The language $L(q)$ of q is the set of all trees that model q.

A *nondeterministic (unranked) tree automaton (NTA)* over the alphabet Λ is a tuple $A = (S, \Lambda, \delta, F)$, where S is a finite set of states, δ is a set of transition rules of the form $(s,a) \to L$, where $(s,a) \in S \times \Lambda$ and L is a regular string language over S, and $F \subseteq S$ is the set of accepting states. A *run* of A on a tree T is a labeling $\lambda : nodes(T) \to S$ such that, for every $v \in nodes(T)$ with label a and children $v_1, \dots, v_n$ from left to right, there exists $(s,a) \to L \in \delta$ such that $\lambda(v) = s$ and $\lambda(v_1) \dots \lambda(v_n) \in L$. Note that if v is a leaf, then we need that $(s,a) \to \epsilon \in \delta$, where ϵ is the empty string. A run λ of A on T is *accepting* if $\lambda(root(T)) \in F$. A tree T is *accepted* by A if there exists an accepting run of A on T. The set of all accepted trees by A is denoted $L(A)$.

Given a CQ q and a tree automaton A, we say that q is *valid* w.r.t. A if $L(A) \subseteq L(q)$. A relevant decision problem is the *validity* of CQs w.r.t. nondeterministic tree automata, that is, given a CQ q and an NTA A, decide whether q is valid w.r.t. A.

The Validity Problem for CQ($child, descendant$)

As said, to obtain the desired result we are going to adapt an existing one, that is, the validity problem of a CQ w.r.t. an NTA, using $child$ and $descendant$ predicates to access the tree, is 2ExpTime-hard [Björklund et al., 2008, Theorem 6]. Let us now recall the proof of this result.

Theorem 4.14 (Björklund et al. [2008])**.** *Validity of CQ($child, descendant$) w.r.t. an NTA is 2ExpTime-hard.*

Proof (Sketch). The proof is by reduction from the non-acceptance problem of an alternating exponential space Turing machine M with an empty tape, which is already 2ExpTime-hard. Recall that we can assume, without loss of generality, that M never uses more than 2^n tape cells.

The general idea of the proof is as follows. An NTA A_{CT} is constructed that accepts trees which encode possible accepting computation trees of M. However, A_{CT} does not guarantee that the accepted trees encode consistent computation trees of M w.r.t. the transition relation of M; this is achieved via a query q_{CT}. More precisely, given a tree $T \in L(A_{CT})$, q_{CT} is satisfied by T iff the transition relation of M is *not* respected by T. This implies that q_{CT} is valid w.r.t. A_{CT} iff there is no consistent and accepting computation tree for M, which in turn is equivalent

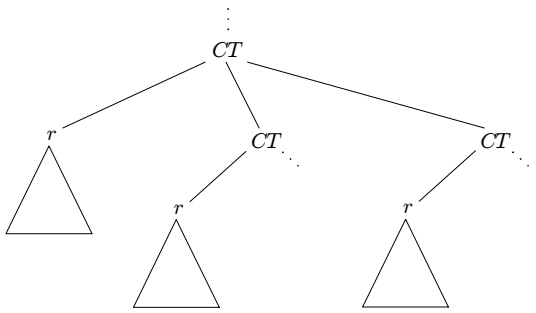


Figure 4.1: A part of an encoded configuration tree.

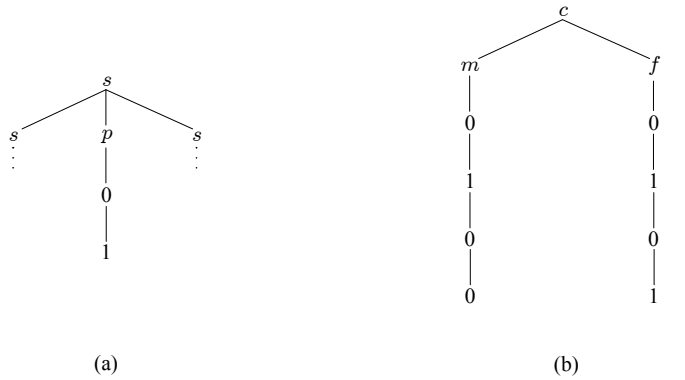


Figure 4.2: Gadgets for the proof of Theorem 4.14. (a) A skeleton note which is the left child of its parent, with its two skeleton node children and its navigation gadget; (b) A cell gadget encoding configuration cell 2 (assuming that $k = 4$), where configuration cells 2 and 4 are not valid successors of the configuration cell 2.

to say that M rejects. But of course 2ExpTime is closed under complementation, and hence the validity of CQs w.r.t. NTAs is 2ExpTime-hard.

To implement the above idea, suitable encodings for the computation trees and the configurations of M are needed.

Encoding Computation Trees. The encoding of a computation tree T of M, denoted $enc(T)$, is depicted in Figure 4.1, and is obtained from T by replacing each node v of T with a tree T_v, where the root of T_v is labeled by CT; the leftmost child of the root of T_v is labeled by r, and is actually the root of the tree which encodes the actual configuration which labels v (this encoding is defined below); and for each child v_i of v, the root of T_v has a subtree $enc(T[v_i])$, where $T[v_i]$ is the subtree of T rooted at v_i.

Encoding Configurations as Trees. Before encoding a configuration of M as a tree, first is represented as a sequence of 2^n configuration cells. Intuitively, a configuration cell contains the content of a tape cell of M plus some additional information such as, for example, the cells that are not currently visited by the cursor, and also were not visited in the last configuration. Notice that there are only polynomially many configuration cells for M, let say k, and a fixed integer $i \in [k]$ is associated to each one of them. Given two sequences C_1 and C_2 of 2^n con-

figuration cells, a set of constraints, denoted $\mathcal{C}(M)$, can be identified such that the following holds: C_1 is a valid encoding of a configuration implies C_2 is a valid encoding of a successor configuration of C_1 iff the constraints of $\mathcal{C}(M)$ are satisfied. In fact, $\mathcal{C}(M)$ is a set of pairs (i, j) expressing that the configuration cell j is a valid successor for the configuration cell i. For more details on configuration cells and the constraints of $\mathcal{C}(M)$, we refer the reader to the extended version of [Björklund et al., 2008].

Let us now describe how the configurations of M can be encoded as trees. Each sequence of 2^n configuration cells is represented as a full binary tree of height n, in which a leaf represents a configuration cell, and is defined as follows. The root is labeled by r, while the other nodes are labeled by s. The s-labeled nodes are called *skeleton nodes*. To each skeleton node v a *navigation gadget* is attached, indicating whether v is a left or a right child. More precisely, a path of length three labeled with $p, 0, 1$ (resp., $p, 1, 0$) is attached to the left (resp., right) child; see Figure 4.2(a).

Furthermore, to each leaf skeleton node, that is, a skeleton node that has no skeleton node children, a *cell gadget* (which is a binary tree) is attached, providing the relevant information about the tape cell it represents. Let us describe the gadget for the i-th configuration cell. The root of the gadget has label c (for *cell*) and has two children labeled by m (for *me*) and by f (for *forbidden*), respectively. Under the m-labeled node a path of length k is attached, where all nodes are labeled by 0, except the i-th node from the top which is labeled by 1. Under the f-labeled node a path of length k is also attached, where the j-th node from the top is labeled by 0 if $(i, j) \in \mathcal{C}(M)$; otherwise, is labeled 1. In other words, the nodes labeled by 1 represent configuration cells which are not allowed on top of the configuration cell i; see Figure 4.2(b).

Detecting Violations of the Transition Relation. The crucial step of the reduction is to use the query q_{CT} to detect a violation of the transition relation of M, or, equivalently, to check the consistency between configuration trees w.r.t. the transition relation of M. The way that configurations are encoded, allows the query to navigate from a node representing configuration cell i in one configuration tree to the node representing cell i in a successor configuration, and this is exactly what is needed in order to perform the above consistency check.

A query $sameCell(X, Y)$ can be constructed which evaluates to true for two leaf skeleton nodes X and Y iff they belong to successive configuration trees and represent the same configuration cells. To this aim, several subqueries are successively defined. The first one asserts that two nodes R_X and R_Y are roots of two successive configuration trees:

$$succ(R_X, R_Y) \equiv \exists X \exists Y\, (CT(X) \wedge CT(Y) \wedge child(X, Y) \wedge r(R_X) \wedge r(R_Y) \wedge child(X, R_X) \wedge child(Y, R_Y))\,.$$

By using the above query, $\Phi_i(X, Y)$ can be defined which asserts that the nodes X and Y belong to successive configuration trees, and are both at level $i > 0$. For brevity, $child^i(X, Y)$ is used

as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1}\,(child(X, Z_1) \wedge \ldots \wedge child(Z_{i-1}, Y))$.

$$\Phi_i(X,Y) \equiv \exists R_X \exists R_Y \Big(succ(R_X, R_Y) \wedge child^i(R_X, X) \wedge child^i(R_Y, Y) \wedge s(X) \wedge s(Y)\Big).$$

Now, a query $\Psi_i(X,Y)$ can be defined which expresses that $\Phi_i(X,Y)$ holds, and, in addition, the nodes X and Y are either both left children or both right children of their parents. Notice that left and right children can be distinguished by the distance of their 1-labeled gadget node from their p-labeled gadget node. In fact, a skeleton node v at level i of a configuration tree and a skeleton node u at level i of a successor configuration tree are both left or both right children, if the nodes v_1 and u_1 with label 1 in their respective gadgets have a common ancestor which has distance $i+4$ from v_1 and $i+5$ from u_1. The definition of $\Psi_i(X,Y)$ follows:

$$\begin{aligned}\Psi_i(X,Y) \equiv \exists P_l \exists P_r \exists T_l \exists T_r \exists Z\,(&\Phi_i(X,Y) \wedge p(P_l) \wedge p(P_r) \wedge 1(T_l) \wedge 1(T_r) \wedge child(X,P_l) \wedge \\ &child(Y,P_r) \wedge descendant(P_l,T_l) \wedge descendant(P_r,T_r) \wedge \\ &child^{i+4}(Z,T_l) \wedge child^{i+5}(Z,T_r)\Big).\end{aligned}$$

By exploiting the above auxiliary predicates, the query $sameCell(X,Y)$ can be defined as follows:

$$\begin{aligned}&sameCell(S_X,S_Y)\\ \equiv\ &\exists R_X \exists R_Y \exists X_1 \ldots \exists X_{n-2} \exists Y_1 \ldots \exists Y_{n-2}\\ &\Bigg(child(R_X,X_1) \wedge \bigwedge_{1 \leqslant i < n-2} child(X_i,X_{i+1}) \wedge child(X_{n-2},S_X) \wedge \\ &\qquad child(R_Y,Y_1) \wedge \bigwedge_{1 \leqslant i < n-2} child(Y_i,Y_{i+1}) \wedge child(Y_{n-2},S_Y) \wedge \\ &\qquad\qquad \bigwedge_{1 \leqslant i \leqslant n-1} \Psi_i(X_i,Y_i) \wedge \Psi_n(S_X,S_Y)\Bigg).\end{aligned}$$

With the query $sameCell(X,Y)$ in place, the consistency of two subsequent configuration trees w.r.t. the transition relation of M can now be checked by exploiting the cell gadgets:

$$\begin{aligned}&q_{CT}\\ \equiv\ &\exists S_l \exists S_r \exists T_l \exists T_r \exists F_l \exists M_r \exists P_l \exists P_r \exists Z\,(sameCell(S_l,S_r) \wedge child(S_l,T_l) \wedge child(S_r,T_r) \wedge \\ &f(F_l) \wedge m(M_r) \wedge 1(P_l) \wedge 1(P_r) \wedge child(T_l,F_l) \wedge child(T_r,M_r) \wedge descendant(F_l,P_l) \wedge \\ &descendant(M_l,P_l) \wedge child^{n+k+3}(Z,P_l) \wedge child^{n+k+4}(Z,P_r)\Big).\end{aligned}$$

This completes the construction. □

The Validity Problem for CQ($child, parentorchild$)

By adapting the proof of the above result, we can now show an analogous result where the predicates $child$ and $parentorchild$ (with the obvious semantics) are used to access the tree:

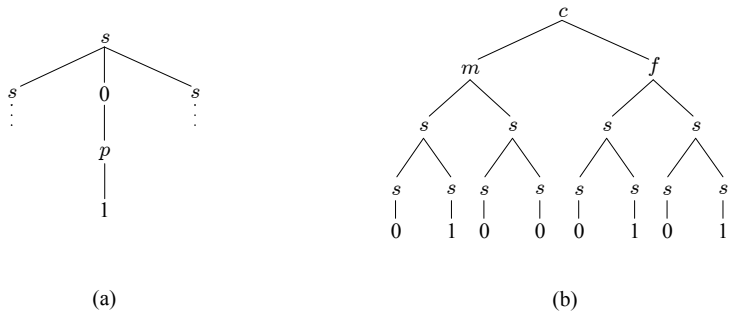


Figure 4.3: Adapted gadgets for the proof of Proposition 4.15.

Proposition 4.15. *Validity of CQ($child, parentorchild$) w.r.t. an NTA is 2ExpTime-hard.*

Proof. The proof is again by reduction from the non-acceptance problem of an alternating exponential space Turing machine M. We assume, without loss of generality, that M has an empty input, uses at most 2^n tape cells, has exactly one accepting state s_a and one rejecting state s_r. Every configuration of M has exactly two different successor configurations, and, whenever M halts, the head is on the left-most cell of the input tape, and the tape is empty. Furthermore, M always goes from a universal to an existential state and vice-versa, and every universal step only affects the state, but not the tape content or head position. We also assume that M always halts.

The construction is an adaptation of the one employed for proving Theorem 4.14, with the aim of constructing an NTA A_{CT} and a CQ q_{CT} such that q_{CT} is valid w.r.t. A_{CT} iff M rejects. Since we can no longer use the $descendant$ predicate in the query to "skip" tree nodes, the navigation and cell gadgets must be suitably adapted for use with our $parentorchild$ predicate.

- *Navigation Gadget.* The navigation gadget is adapted by switching the first and second label, resulting, for example, in the structure depicted in Figure 4.3(a) obtained after adapting the one in Figure 4.2(a).
- *Cell Gadget.* The paths of length k attached to the m- and f-labeled nodes (recall that k is the number of all the possible configuration cells which is polynomial) are now transformed into full binary trees of the same structure as the trees encoding configurations. Such binary trees have height logarithmic in k, and the leaves are labeled by 0 or 1, equivalent to the paths they represent. For example, when adapting the cell gadget in Figure 4.2(b), we obtain the cell gadget depicted in Figure 4.3(b); for clarity, the navigation gadgets attached to s-labeled nodes are omitted.

Having the adapted gadgets in place, we are now ready to conclude the construction. The NTA A_{CT} can be constructed in exactly the same way as the one in the proof of Theorem 4.14; the details are omitted, and we refer the reader to the extended version of [Björklund et al., 2008]. Let us now construct the query q_{CT}. In the rest of the proof, we use $child^i(X, Y)$ as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1}\,(child(X, Z_1) \wedge \ldots \wedge child(Z_{i-1}, Y))$.

As in the proof of Theorem 4.14, we define a query $sameCell(X,Y)$ which evaluates to true for two leaf skeleton nodes iff they belong to successive configuration trees and represent the same configuration cells. However, $sameCell(X,Y)$ must be defined using the predicate $parentorchild$ instead of the predicate $descendant$. In fact, $sameCell(X,Y)$ is defined in the same way as above, keeping the subqueries $succ(R_X,R_Y)$ and $\Phi_i(X,Y)$ unchanged, but modifying $\Psi_i(X,Y)$ as follows:

$$\begin{aligned}\Psi_i(X,Y) \equiv\ & \exists P_l \exists P_r \exists T_l \exists T_r \exists Z \big(\Phi_i(X,Y) \wedge p(P_l) \wedge p(P_r) \wedge 1(T_l) \wedge 1(T_r) \wedge child^2(X,P_l) \wedge \\ & child^2(Y,P_r) \wedge parentorchild(P_l,T_l) \wedge parentorchild(P_r,T_r) \wedge \\ & child^{i+4}(Z,T_l) \wedge child^{i+5}(Z,T_r)\big).\end{aligned}$$

Furthermore, once the same configuration cell in two subsequent configuration trees is identified, we need to compare the m- and f-labeled paths to validate the transition. Since our cell gadgets are now trees of the same structure as the trees encoding configurations, we can rely on a mechanism similar to the one given above. More precisely, a query denoted $sameLeaf(X,Y,X',Y')$ can be defined which evaluates to true for two 0/1-labeled leaf nodes X' and Y', which belong to a tree rooted at an m-labeled node X and an f-labeled node Y, respectively, if and only if they belong to successive trees and represent the same leaf node. To this aim, the subqueries $\Phi_i(X,Y)$ and $\Psi_i(X,Y)$ are modified as follows:

$$\widehat{\Phi}_i(R_X,R_Y,X,Y) \equiv child^i(R_X,X) \wedge child^i(R_Y,Y) \wedge s(X) \wedge s(Y),$$

that is, we drop the atom $succ(R_X,R_Y)$, and

$$\widehat{\Psi}_i(R_X,R_Y,X,Y) \equiv \Psi_i[\Phi_i(X,Y) \setminus \widehat{\Phi}_i(R_X,R_Y,X,Y)],$$

where $\Psi_i[\Phi_i(X,Y) \setminus \widehat{\Phi}_i(R_X,R_Y,X,Y)]$ is the query obtained from $\Phi_i(X,Y)$ be replacing the conjunct $\Phi_i(X,Y)$ with $\widehat{\Phi}_i(R_X,R_Y,X,Y)$. The query $sameLeaf(X,Y,X',Y')$ is defined as follows:

$$\begin{aligned}& sameLeaf(R_X,R_Y,S_X,S_Y) \\ \equiv\ & \exists X_1 \ldots \exists X_{n-2} \exists Y_1 \ldots \exists Y_{n-2} \\ & \Bigg(child(R_X,X_1) \wedge \bigwedge_{1 \leqslant i < \lceil \log k \rceil - 2} child(X_i,X_{i+1}) \wedge child(X_{n-2},S_X) \wedge \\ & \quad child(R_Y,Y_1) \wedge \bigwedge_{1 \leqslant i < \lceil \log k \rceil - 2} child(Y_i,Y_{i+1}) \wedge child(Y_{n-2},S_Y) \wedge \\ & \quad \bigwedge_{1 \leqslant i \leqslant n-1} \widehat{\Psi}_i(R_X,R_Y,X_i,Y_i) \wedge \widehat{\Psi}_{\lceil \log k \rceil}(R_X,R_Y,S_X,S_Y) \Bigg).\end{aligned}$$

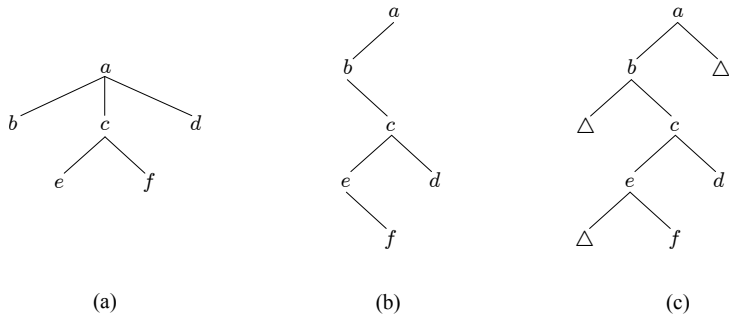


Figure 4.4: (a) A ternary ordered tree T; (b) The binary version T' of T in left child-right sibling representation; (c) The conversion of T' into a full binary tree by adding the "dummy" nodes $\triangle$.

With the queries $sameCell(X, Y)$ and $sameLeaf(X, Y, X', Y')$ in place, the consistency of two subsequent configuration trees w.r.t. the transition relation of M can now be checked by exploiting the cell gadgets as follows:

$$
\begin{aligned}
& q_{CT} \\
\equiv\ & \exists S_l \exists S_r \exists T_l \exists T_r \exists F_l \exists M_r \exists P_l \exists P_r \exists L_l \exists L_r \exists Z\ (sameCell(S_l, S_r) \wedge child(S_l, T_l) \wedge \\
& \quad child(S_r, T_r) \wedge f(F_l) \wedge m(M_r) \wedge 1(P_l) \wedge 1(P_r) \wedge child(T_l, F_l) \wedge child(T_r, M_r) \wedge \\
& \quad\quad sameLeaf(F_l, M_r, L_l, L_r) \wedge child(L_l, P_l) \wedge child(L_r, P_r))\,.
\end{aligned}
$$

This completes the construction. □

In the proof of Proposition 4.15, we consider ternary ordered trees. It is well-known that a ternary ordered tree can be transformed into a binary ordered tree by representing it in left child-right sibling way, which in turn can be transformed into a full binary tree by adding some "dummy" nodes; an example of such a transformation is shown in Figure 4.4. This implies that the proof of Proposition 4.15 can be accordingly adapted in order to show that the 2ExpTime-hardness holds even for NTAs over full binary trees.

CQ Answering under DIDs

Having the above result in place, we are now ready to show that CQ answering under DIDs is 2ExpTime-hard in combined complexity, by simulating the behavior of an NTA over full binary trees using a set of DIDs.

Theorem 4.16. CQ-Answering *under DIDs is 2ExpTime-hard in combined complexity, even for predicates of arity at most two.*

Proof. Consider a CQ q which falls in the fragment CQ($child, parentorchild$), and an NTA $A = (S, \Lambda, \delta, F)$ over full binary trees, where S is a finite set of states, Λ is a finite alphabet, δ is a set of transition rules of the form $(s, a) \to L$, where $L \in S^2 \cup \{\epsilon\}$, and F is the set of accepting states. Our goal is to construct a database D and a set Σ of DIDs such that $D \cup \Sigma \models q$ iff q is valid w.r.t. A. Before giving the construction, let us describe the schema of Σ.

The Schema of Σ. The schema $sch(\Sigma)$, apart from the binary predicates describing the tree axis, namely $child$ and $parentorchild$, contains also the following predicates:

- $root(\cdot)$; $root(v)$ says that v is the root of the tree;
- $s(\cdot)$, for each $s \in S$; $s(v)$ states that the node v is in state s;
- $a(\cdot)$, for each $a \in \Lambda$; $a(v)$ expresses that the node v is labeled by a;
- $p_{s,a,L}(\cdot)$, for each $(s, a, L) \in S \times \Lambda \times S^2 \cup \{\epsilon\}$; $p_{s,a,L}(v)$ says that v is in state s, is labeled by a, and its children (from left to right) are labeled by L; and
- $child^i[(s, a), s_1, s_2](\cdot, \cdot)$, where $i \in \{1, 2\}$, for each transition rule $(s, a) \rightarrow (s_1, s_2)$ of δ; $child^i[(s, a), s_1, s_2](v, u)$ states that u is the i-th child of v, where v is in state s and labeled by a, while u is in state s_i.

Observe that the above schema depends on the particular NTA that we encode, and thus it is not fixed; however, it can be constructed in polynomial time. Notice that in $sch(\Sigma)$ we only have unary and binary predicates.

The Set Σ. We are now ready to define the set Σ of DIDs, which will simulate the behavior of the NTA A. Let us first give the intuition underlying Σ. First, we guess in which accepting state the root of the tree is. Also, for each node v in state $s \in S$, we guess a valid label for it and for its children. Then, for such a node v, we introduce, via a (non-disjunctive) TGD, two children according to the transition function δ using the $child$ predicate; we also generate appropriate $parentorchild$ atoms. These children are then labeled with the corresponding subsequent state according to δ. Repeating this procedure via the disjunctive chase, we obtain a set of models, where each of them represents exactly one of the trees that A would accept. Formally speaking, Σ consists of the following DTGDs:

- We guess the state of the root by the DID

$$root(X) \rightarrow \bigvee_{s \in F} s(X).$$

- For each $s \in S$,

$$s(X) \rightarrow \bigvee_{a \in \Lambda,\ (s,a) \rightarrow L \in \delta} p_{s,a,L}(X).$$

- For each $(s, a, L) \in S \times \Lambda \times S^2 \cup \{\epsilon\}$,

$$p_{s,a,L}(X) \rightarrow a(X).$$

- For each $i \in \{1,2\}$ and $(s,a) \to L \in \delta$ such that $L = (s_1, s_2) \neq \epsilon$,

$$\begin{aligned} p_{s,a,L}(X) &\to \exists Y child^i[(s,a), s_1, s_2](X,Y) \\ child^i[(s,a), s_1, s_2](X,Y) &\to s_i(Y) \\ child^i[(s,a), s_1, s_2](X,Y) &\to child(X,Y) \end{aligned}$$

- Finally, we have the IDs

$$\begin{aligned} child(X,Y) &\to parentorchild(X,Y) \\ child(X,Y) &\to parentorchild(Y,X). \end{aligned}$$

Correctness. The database D consists of the atom $root(c)$, where $c \in \mathbf{C}$ is a constant which represents the root of the tree. Each instance $I \in chase(D,\Sigma)$, if restricted to the $child$- and a-predicates, where $a \in \Lambda$, by construction, is a binary tree accepted by A. Furthermore, again by construction, there is an instance $I \in chaseD\Sigma$ for each such tree accepted by A. Moreover, the $parentorchild$-predicate in I is semantically equivalent to the $parentorchild$-relation of A.

Therefore, if $D \cup \Sigma \models q$ or, equivalently, each $I \in chase(D,\Sigma)$ entails q, then each tree accepted by A is also entailed by q, that is, $L(A) \subseteq L(q)$, which is equivalent to say that q is valid w.r.t. A. Conversely, if $L(A) \subseteq L(q)$, then, for each $I \in chase(D,\Sigma)$, there exists $T \in L(A)$ which appears in I. Since $T \in L(q)$, we get that $I \models q$. But this implies that $D \cup \Sigma \models q$, and the claim follows. □

Notice that the constructed set Σ of DIDs in the above proof depends on the NTA A that we encode. Interestingly, a new construction can be devised in such a way that Σ is independent from A, and thus it is fixed. However, for such a construction we need the expressive power of UCQs.

Intuitively, in order to lose the dependency on the set of transition rules of the NTA that we encode, we replace each state (resp., alphabet symbol, string of states) by a chain of length at most n (resp., m, ℓ), where n (resp., m, ℓ) is the number of states (resp., alphabet symbols, strings of states that can be formed). Notice that such chains can easily be constructed via two simple DIDs. Then, we can access a state, or an alphabet symbol, or a string of states, by exploiting those chains, without explicitly encode them inside a predicate (as we did in the proof of Theorem 4.17). After applying this trick, the set of DIDs no longer depends on the NTA, and thus it is fixed as desired.

However, we need to guarantee that each chain encodes a valid state, or alphabet symbol, or string of states, and also that our chain-like encoding is consistent with the transition rules of the NTA. These properties can be checked via a UCQ. Let us now proceed with the formal result:

Theorem 4.17. UCQ-ANSWERING *under fixed sets of DIDs is 2EXPTIME-hard, even for predicates of arity at most two.*

Proof. Consider a CQ q which falls in the fragment CQ($child, parentorchild$), and an NTA $A = (S, \Lambda, \delta, F)$ over full binary trees. We are going to construct a database D, a set Σ of DIDs, and a UCQ Q such that $D \cup \Sigma \models Q$ iff q is valid w.r.t. A. Let us first give the schema of Σ.

The Schema of Σ. The schema $sch(\Sigma)$, apart from the binary predicates describing the tree axis, namely $child$ and $parentorchild$, includes also the following predicates:

- $chain(\cdot,\cdot)$; $chain(v,u)$ means that u is the successor of v in a chain;
- $next(\cdot)$; $next(v)$ says that v, which is a part of a chain, has a successor;
- $end(\cdot)$; $end(v)$ states that v is the last element of a chain;
- $state(\cdot,\cdot)$; $state(v,u)$ says that the chain encoding the state for v starts at u;
- $label(\cdot,\cdot)$; $label(v,u)$ means that the chain encoding the label for v starts at u;
- $string(\cdot,\cdot)$; $string(v,u)$ expresses that the chain encoding the string for v starts at u;
- $leaf(\cdot)$; $leaf(v)$ says that v is a leaf node;
- $notLeaf(\cdot)$; $notLeaf(v)$ means that v is an internal node; and
- $child^i(\cdot,\cdot)$, where $i \in \{1,2\}$; $child^i(v,u)$ says that u is the i-th child of v.

It is important to say that the above schema does not depend on A.

The Set Σ. We are now ready to define the set Σ of DIDs.

- We generate chains of nodes by using the following DIDs:

$$\begin{aligned} next(X) &\rightarrow \exists Y\, chain(X,Y) \\ chain(X,Y) &\rightarrow next(Y) \vee end(Y). \end{aligned}$$

- We say that the root is a node, and also that each node is a leaf or an internal node, by employing the following DIDs:

$$\begin{aligned} root(X) &\rightarrow node(X) \\ node(X) &\rightarrow leaf(X) \vee notLeaf(X). \end{aligned}$$

- We point to the first element of a chain by employing the following DIDs:

$$\begin{aligned} node(X) &\rightarrow \exists Y\, state(X,Y) \\ state(X,Y) &\rightarrow next(Y) \\ node(X) &\rightarrow \exists Y\, label(X,Y) \\ label(X,Y) &\rightarrow next(Y) \\ node(X) &\rightarrow \exists Y\, string(X,Y) \\ string(X,Y) &\rightarrow next(Y). \end{aligned}$$

- For each $i \in \{1,2\}$,

$$\begin{aligned} notLeaf(X) &\rightarrow \exists Y\, child^i(X,Y) \\ child^i(X,Y) &\rightarrow node(Y) \\ child^i(X,Y) &\rightarrow child(X,Y). \end{aligned}$$

- Finally, we have the IDs

$$\begin{aligned} child(X,Y) &\rightarrow parentorchild(X,Y) \\ child(X,Y) &\rightarrow parentorchild(Y,X). \end{aligned}$$

It is easy to verify that the above set of DIDs does not depend on A.

The UCQ Q. It remains to construct the UCQ Q. In what follows, we assume a fixed order on the states of S, on the symbols of Λ, and the set of strings $S^2 \cup \{\epsilon\}$. Given a state $s \in S$, let $\#_s$ be its position in the above order. The length of the chain which encodes s is $\#_s$; analogously for the symbols of Λ and the strings of $S^2 \cup \{\epsilon\}$. Intuitively, Q consists of the following disjuncts:

1. an adaptation of the CQ q, denoted q_{adapt}, obtained by replacing any occurrence of an a-predicate, where $a \in \Lambda$, with a chain of length $\#_a$;
2. Q_{root} for checking that the root is in an accepting state;
3. Q_{length} for checking that any chain following a state (resp., an alphabet symbol, a string of $S^2 \cup \{\epsilon\}$) has length at most $|S|$ (resp., $|\Lambda|$, $|S^2|+1$); and
4. Q_{trans} for checking the consistency w.r.t. the transition rules of A.

Let us now formalize the above four components of Q. Two useful queries that we are going to use are the following:

$$\begin{aligned} & length_n(X) \\ \equiv\ & \exists X_1 \ldots \exists X_{n-1} \left(next(X) \wedge chain(X,X_1) \wedge \bigwedge_{1 \leqslant i < n-1} chain(X_i, X_{i+1}) \wedge end(X_{n-1}) \right), \end{aligned}$$

stating that there exists a chain starting at X which has a chain of length n, and

$$longerThan_n(X) \equiv \exists X_1 \ldots \exists X_n \left(next(X) \wedge chain(X, X_1) \wedge \bigwedge_{1 \leqslant i < n-1} chain(X_i, X_{i+1}) \wedge chain(X_{n-1}, X_n) \right),$$

asserting that there exists a chain starting at X which is longer than n. The disjuncts of Q follow:

1. q_{adapt} is obtained from q by replacing each atom $a(X)$, where $a \in \Lambda$, with the conjunction of atoms $length_{\#_a}(X)$;

2. Q_{root} is defined as

$$\bigvee_{s \notin F} \exists X \exists Y \left(root(X) \wedge state(X, Y) \wedge length_{\#_s}(Y) \right);$$

3. Q_{length} is defined as

$$\exists X \exists Y \left(state(X, Y) \wedge longerThan_{|S|}(Y) \right) \vee \exists X \exists Y \left(label(X, Y) \wedge longerThan_{|\Lambda|}(Y) \right) \vee \exists X \exists Y \left(string(X, Y) \wedge longerThan_{(|S^2|+1)}(Y) \right);$$

4. Finally, Q_{trans} is defined as

$$\exists X \exists Y \left(string(X, Y) \wedge length_{\#_\epsilon}(Y) \wedge notLeaf(Y) \right) \vee \bigvee_{(s,a) \to L, \notin \delta, L \neq \epsilon} \exists X_1 \exists Y_1 \exists X_2 \exists Y_2 \exists X_3 \exists Y_3 \left(state(X_1, Y_1) \wedge length_{\#_s}(Y_1) \wedge label(X_2, Y_2) \wedge length_{\#_a}(Y_2) \wedge string(X_3, Y_3) \wedge length_{\#_L}(Y_3) \right).$$

Correctness. The database D consists of the atom $root(c)$, where $c \in \mathbf{C}$ is a constant which represents the root of the tree. Clearly, the set of instances

$$I_{valid} = \{ I \in chase(D, \Sigma) \mid I \not\models (Q_{root} \vee Q_{length} \vee Q_{trans}) \}$$

are exactly the instances of $chase(D, \Sigma)$ which encode a valid binary tree accepted by A. If $D \cup \Sigma \models Q$, then each $I \in I_{valid}$ entails q_{adapt}, and thus $L(A) \subseteq L(q)$, which means that q is valid w.r.t. A. Conversely, if $L(A) \subseteq L(q)$, then, for each $I \in I_{valid}$, $I \models q_{adapt}$. Clearly, each instance $I \in chase(D, \Sigma) \setminus I_{valid}$ entails $(Q_{root} \vee Q_{length} \vee Q_{trans})$. Therefore, for each $I \in chase(D, \Sigma)$, $I \models Q$, which implies that $D \cup \Sigma \models Q$, as needed. □

The question that comes up is whether the above result can be extended to CQs. Interestingly, this question is answered positively, as is shown by Lemma 4.6. Since the modified Σ' in the proof construction of the above lemma does not depend on the database or the query, but only on the original set Σ of DTGDs, and the arity is only increased by one, Theorem 4.17 and Lemma 4.6 together immediately imply the desired result:

Theorem 4.18. CQ-Answering *under fixed sets of DIDs is 2ExpTime-hard, even for predicates of arity at most three.*

Recall that answering UCQs under weakly-frontier-guarded DTGDs is in 2ExpTime in the combined complexity (Theorem 4.10). Hence, the picture of the computational complexity of query answering under our guarded-based classes of DTGDs is now complete.

Interestingly, Theorem 4.18 also closes an open question stated in the conference version of the paper [Bárány et al., 2014], originally published at LICS 2010, regarding the complexity of query answering under fixed GFO sentences. It was shown that the problem in question is PSpace-hard even for CQs, and in ExpTime in case of a restricted class of CQs. However, the exact complexity was left as an open problem. Clearly, the above result gives a 2ExpTime-completeness result since query answering under GFO is in 2ExpTime in general.

Corollary 4.19. (U)CQ-Answering *under fixed GFO sentences is 2ExpTime-complete.*

4.6 Answering Bounded (Hyper-)Treewidth Queries

Several subclasses of CQs have been considered in the literature with the aim of reducing the complexity of classical database problems such as query evaluation and query containment. Such a well-known fragment is the class of CQs of *bounded treewidth (BTWCQs)*, that is, the treewidth of their hypergraph is bounded by an integer constant; see, e.g. Chekuri and Rajaraman [2000]. Let us recall that the hypergraph $\mathcal{H}(q)$ of a query q is defined in the same way as the hypergraph $\mathcal{H}(I)$ of an instance I (see Section 2)—the relations of the atoms represent hyperedges, and terms inside the atoms represent the vertices. Our goal in this section is to understand how the complexity of answering (U)CQ under our guarded-based classes of DTGDs is affected if we focus on queries of bounded treewidth.

Interestingly, queries of bounded treewidth do not alter the complexity of our problem. Table 4.1, which summarizes the complexity for answering (U)CQs, holds even if we consider (U)BTWCQs. As you can observe, the data complexity for all the classes of DTGDs under consideration is obtained from existing results. More precisely, the co-NP-hardness for DIDs is obtained from [Calvanese et al., 2013, Theorem 4.5], where it is shown that CQ answering under a single DID of the form $p_1(X) \rightarrow p_2(X) \vee p_3(X)$, is already co-NP-hard, even if the input query is fixed (and thus of bounded treewidth). The co-NP upper bound for frontier-guarded DTGDs follows from Theorem 4.12. The ExpTime-hardness for weakly-guarded DTGDs is inherited from [Calì et al., 2013, Theorem 4.1], where it is shown that CQ answering under a

fixed set of weakly-guarded TGDs is ExpTime-hard, even if the input query is a single atom. The ExpTime upper bound for weakly-frontier-guarded DTGDs follows from Theorem 4.13.

Although the data complexity of our problem can be settled by exploiting known results, this is not true for the other settings of the problem. The best known upper bound is the 2ExpTime upper bound for answering arbitrary UCQs under weakly-frontier-guarded DTGDs (Theorem 4.10). This result, combined with the fact that CQ answering under guarded TGDs is 2ExpTime-hard in the combined complexity [Calì et al., 2013, Theorem 6.1], even for atomic queries of the form $\exists X p(X)$ (and thus of bounded treewidth), closes the combined complexity for (weakly-)(frontier-)guarded DTGDs. However, the above lower bound for guarded DTGDs is not strong enough to complete the complexity picture of our problem. The rest of this section is devoted to establish a strong lower bound which, together with the above 2ExpTime upper bound, gives us the complete picture of the complexity of the problem studied in this section. In particular, we show that:

Theorem 4.20. B(H)TWQ-Answering *under fixed sets of DIDs is 2ExpTime-hard.*

Proof. The proof is by a reduction from the non-acceptance problem of an alternating exponential space Turing machine M on the empty input. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ being the blank symbol, $\delta : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that M is well-behaved and never tries to read beyond its tape boundaries, always halts, and uses exactly 2^n tape cells. Furthermore, we assume that a rejecting configuration does not have a subsequent configuration, while an accepting configuration has only itself as a subsequent configuration. We also assume that $s_0 \in S_\exists$, and also that every universal configuration is followed by two existential configurations and vice versa. Finally, for technical reasons which will be clarified later, we assume that M never touches the first and the last cells of the tape, and also starts with its head at the second cell.

The above assumptions can be made, without sacrificing the generality of our proof, since the non-acceptance problem of M remains 2ExpTime-hard. Our goal is to construct a database D, a set Σ of DTGDs, and a (U)BTWCQ Q such that $D \cup \Sigma \models Q$ iff M rejects, and $\mathsf{N}(\Sigma)$ is a fixed set of DIDs. By Lemma 4.6, the obtained lower bound holds even if we focus on BTWCQs.

We follow the same approach as before, that is, to construct trees, which encode possible computation trees of M, by chasing D and Σ, and then check their consistency via the query Q. On each configuration node v, which represents the configuration C_v of M, we attach a configuration tree, that is, a full binary tree of depth n, and thus at its n-th level there are exactly 2^n nodes which represent the cells of the tape of M in C_v. Furthermore, for each cell we guess whether the cursor of M is at this cell, and if so, we attach a chain of length at most $|S|$, which encodes the state of C_v. The above informal description is illustrated in Figure 4.5. We now proceed with the formal construction of D, Σ and Q.

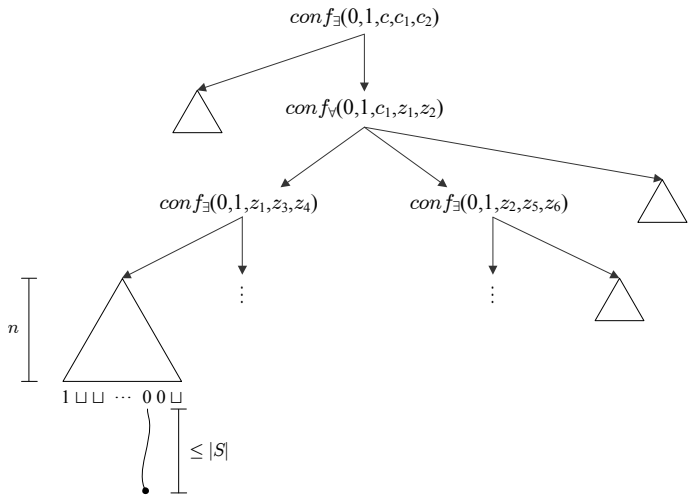


Figure 4.5: Representation of the computation tree in the proof of Theorem 4.20.

The Database D. Let $D = \{conf_\exists(0,1,c,c_1,c_2)\}$, where $\{c,c_1,c_2\} \subset \mathbf{C}$ are special constants which represent the initial configuration (c), and its two subsequent configurations (c_1 and c_2). Notice that, by assumption, the initial configuration is existential. Let us say that the two constants 0 and 1 are auxiliary constants which will allows us to have access to 0 and 1 without explicitly mention them in Σ; recall that DIDs are constant-free.

The Set Σ. The predicates that we are going to use are self-explanatory, and thus we proceed with the construction of Σ without describing the predicates of $sch(\Sigma)$.

- Each existential (resp. universal) configuration has one (resp. two) successor configuration(s) which is/are universal (resp. existential):

$$\begin{aligned} conf_\exists(Z,O,T,N_1,N_2) &\rightarrow \bigvee_{i\in\{1,2\}} \exists N_3 \exists N_4\, conf_\forall(Z,O,N_i,N_3,N_4), \\ conf_\forall(Z,O,T,N_1,N_2) &\rightarrow \exists N_3 \exists N_4 \exists N_5 \exists N_6\, conf_\exists(Z,O,N_1,N_3,N_4), \\ &\qquad conf_\exists(Z,O,N_2,N_5,N_6). \end{aligned}$$

  Roughly speaking, the atom $conf_x(1,0,z_1,z_2,z_3)$, where $x \in \{\exists,\forall\}$, says that z_1 is an x-configuration with z_2 and z_3 its subsequent configurations.

- Each configuration has a configuration tree (ct), that is, a full binary tree, which simulates its tape and encodes its state:

$$\begin{aligned} conf_x(Z,O,T,N_1,N_2) &\rightarrow ctnode(Z,O,T), \text{ for each } x \in \{\exists,\forall\}, \\ ctnode(Z,O,N) &\rightarrow \exists N_1 \exists N_2\, child(N,N_1), child(N,N_2), \\ &\qquad ctnode(Z,O,N_1), ctnode(Z,O,N_2), \\ &\qquad label(N_1,Z), label(N_2,O), \end{aligned}$$

$$
\begin{aligned}
ctnode(Z, O, N) &\rightarrow 0(N) \vee 1(N) \vee \sqcup(N), \\
ctnode(Z, O, N) &\rightarrow cursor(N) \vee notCursor(N), \\
cursor(N) &\rightarrow \exists S\, state(N, S), \\
state(S_1, S_2) &\rightarrow \exists S_3\, state(S_2, S_3) \vee end(S_2).
\end{aligned}
$$

The construction of Σ is now complete. It is easy to verify that Σ does not depend on M, and also that N(Σ) is a set of DIDs.

The (U)BTWCQ Q. We now proceed with the construction of Q which ensures that each model of $D \cup \Sigma$ encodes a valid computation tree. Q consists of the following disjuncts:

1. $Q_{initial}$ for checking that in the initial configuration the cursor is at the second cell, the tape is empty, and the state is s_0;
2. Q_{length} for checking that state-chains are of length at most $|S|$;
3. Q_{cursor} for checking that exactly one cell is pointed by the cursor;
4. Q_{trans} for checking the consistency w.r.t. the transition function of M; and
5. $Q_{inertia}$ for checking that the cells not under the cursor keep their old value during a transition.

In the sequel, we assume a fixed order on S. Given a state $s \in S$, let $\#_s$ be its position in this order. The length of the state-chain which encodes s is $\#_s$. We use $state^i(X, Y)$ as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1}\,(state(X, Z_1) \wedge \ldots \wedge state(Z_{i-1}, Y))$. A useful subquery that we are going to use is

$$
state[k](X) \equiv \exists Y \left(state^k(X, Y) \wedge end(Y)\right),
$$

stating that there exists a state-chain of length k. Let us now formalize Q.

1. $Q_{initial}$ is defined as

$$
\begin{aligned}
\exists X_1 \ldots \exists X_n \Bigg(& conf_{\exists}(0, 1, c, c_1, c_2) \wedge child(c, X_1) \wedge \bigwedge_{1 \leqslant i \leqslant n} child(X_i, X_{i+1}) \wedge \\
& \bigwedge_{1 \leqslant i < n} label(X_i, 0) \wedge label(X_n, 1) \wedge notCursor(X_n) \Bigg) \vee \\
\bigvee_{\alpha \in \Lambda \setminus \{\sqcup\}} & \exists X \left(conf_{\exists}(0, 1, c, c_1, c_2) \wedge child^n(c, X) \wedge \alpha(X)\right) \vee \\
\bigvee_{s \in S \setminus \{s_0\}} & \exists X \left(conf_{\exists}(0, 1, c, c_1, c_2) \wedge child^n(c, X) \wedge cursor(X) \wedge state[\#_s](X)\right).
\end{aligned}
$$

Recall that in the first two positions of $conf_x$, where $x \in \{\exists, \forall\}$, only the constants $0, 1$ can appear. Also, recall that $c \in \mathbf{C}$ represents the initial configuration of M, with $c_1, c_2 \in \mathbf{C}$ be its two subsequent configurations.

2. Q_{length} is defined as

$$\bigvee_{x\in\{\exists,\forall\}} \exists X \exists Y \exists Z \exists W \left(conf_x(0,1,X,Y,Z) \wedge child^n(X,W) \wedge cursor(W) \wedge state[|S|+1](W) \right).$$

3. Q_{cursor} is defined as

$$\bigvee_{x\in\{\exists,\forall\}} \bigvee_{1<i<n} \exists X \exists X' \exists X'' \exists X_{i-1} \exists X_i \exists X_n \exists Y_{i-1} \exists Y_i \exists Y_n \Big(conf_x(0,1,X,X',X'') \wedge child^{i-1}(X,X_{i-1}) \wedge child(X_{i-1},X_i) \wedge label(X_i,0) \wedge child(X_{i-1},Y_i) \wedge label(Y_i,1) \wedge child^{n-i}(X_i,X_n) \wedge cursor(X_n) \wedge child^{n-i}(Y_i,Y_n) \wedge cursor(Y_n) \Big).$$

Before giving the definition of Q_{trans} and $Q_{inertia}$, we first need to define the auxiliary subquery $\Phi(N_1, N_2, L_1, M_1, R_1, L_2, M_2, R_2)$ which intuitively says that, for the configurations N_1 and N_2, the triples (L_1, M_1, R_1) and (L_2, M_2, R_2) represent the same three consecutive tape cells in N_1 and N_2, respectively. In the definition of Φ, we will exploit the following crucial fact (which can be easily verified): every two consecutive tape cells v_n and u_n, where v_n comes before u_n, share a common ancestor v_i at level i in their configuration tree such that, v_{i+1} (resp., u_{i+1}) is labeled by 0 (resp., 1), and each of the nodes $v_{i+2}, \dots, v_{n-1}$ (resp., $u_{i+2}, \dots, u_{n-1}$) is labeled by 1 (resp., 0). We proceed by considering the two cases where the three consecutive nodes at level n in the configuration tree, which represent the three consecutive cells, are labeled by 010 or 101 (notice that, by construction, all the other cases are not applicable). We first focus on the first case, and we define the subquery Φ_{010} as follows:

$$\begin{aligned}
&\Phi_{010}(N_1^1, N_1^2, L_n^1, N_n^1, R_n^1, L_n^2, N_n^2, R_n^2) \\
\equiv\ &\bigvee_{1\leqslant i\leqslant n-1} \bigwedge_{j\in\{1,2\}} \exists N_2^j \dots \exists N_n^j \exists B_2 \dots \exists B_n \exists R_{i+1}^j \dots \exists R_n^j \\
&\Bigg(\bigwedge_{1\leqslant k<i} child(N_k^j, N_{k+1}^j) \wedge \bigwedge_{1<k\leqslant i} label(N_k^j, B_k) \wedge \bigwedge_{1\leqslant k\leqslant n-i} child(N_{i+k-1}^j, N_{i+k}^j) \wedge \\
&\quad child(N_i^j, R_{i+1}^j) \wedge \bigwedge_{1<k\leqslant n-i} child(R_{i+k-1}^j, R_{i+k}^j) \wedge child(N_{n-1}^j, L_n^j) \wedge \\
&\quad label(N_{i+1}^j, 0) \wedge \bigwedge_{i+2\leqslant k\leqslant n-1} label(N_k^j, 1) \wedge label(R_{i+1}^j, 1) \wedge \\
&\quad \bigwedge_{i+2\leqslant k\leqslant n-1} label(R_k^j, 0) \wedge label(L_n^j, 0) \wedge label(N_n^j, 1) \wedge label(R_n^j, 0) \Bigg).
\end{aligned}$$

Analogously, the subquery Φ_{101} can be defined as follows:

$$\begin{aligned}&\Phi_{101}(N_1^1, N_1^2, L_n^1, N_n^1, R_n^1, L_n^2, N_n^2, R_n^2)\\ \equiv &\bigvee_{1\leqslant i\leqslant n-1}\bigwedge_{j\in\{1,2\}} \exists N_2^j \ldots \exists N_n^j \exists B_2 \ldots \exists B_n \exists L_{i+1}^j \ldots \exists L_n^j\end{aligned}$$

$$\begin{aligned}\Bigg(&\bigwedge_{1\leqslant k<i} child(N_k^j, N_{k+1}^j) \wedge \bigwedge_{1<k\leqslant i} label(N_k^j, B_k) \wedge \bigwedge_{1\leqslant k\leqslant n-i} child(N_{i+k-1}^j, N_{i+k}^j) \wedge\\ &child(N_i^j, L_{i+1}^j) \wedge \bigwedge_{1<k\leqslant n-i} child(L_{i+k-1}^j, L_{i+k}^j) \wedge child(N_{n-1}^j, R_n^j) \wedge\\ &label(N_{i+1}^j, 0) \wedge \bigwedge_{i+2\leqslant k\leqslant n-1} label(N_k^j, 1) \wedge label(L_{i+1}^j, 0) \wedge\\ &\bigwedge_{i+2\leqslant k\leqslant n-1} label(L_k^j, 1) \wedge label(L_n^j, 1) \wedge label(N_n^j, 0) \wedge label(R_n^j, 1)\Bigg).\end{aligned}$$

Finally, the subquery Φ is defined as follows:

$$\Phi(N_1^1, N_1^2, L_n^1, N_n^1, R_n^1, L_n^2, N_n^2, R_n^2) \equiv \bigvee_{x\in\{010,101\}} \Phi_x(N_1^1, N_1^2, L_n^1, N_n^1, R_n^1, L_n^2, N_n^2, R_n^2).$$

We are now ready to give the definition of Q_{trans} and $Q_{inertia}$.

4. For the definition of Q_{trans}, it is more convenient to consider the transitions of δ as *rewriting assertions* of the form

$$(x, y, z) \to ((x_1, y_1, z_1), (x_2, y_2, z_2)),$$

where $x, z \in \Lambda$, $y \in (S \times \Lambda)$, $x_i, y_i \in (\Lambda \cup (S \times \Lambda))$ and $y_i \in \Lambda$, for each $i \in [2]$. Moreover, for each $i \in [2]$, $x_i \in \Lambda$ implies $z_i \in (S \times \Lambda)$, and $x_i \in (S \times \Lambda)$ implies $z_i \in \Lambda$. Intuitively, y represents the state of the machine and the symbol read by the cursor, while x and z are the symbols to the left and right of the cursor position. Such a partial configuration (zyz) then transforms into $(x_1y_1z_1)$ and $(x_2y_2z_2)$, according to the transition relation δ. Before defining Q_{trans}, we need to introduce some auxiliary notions. First, we define the complement of δ relative to universal and existential states, denoted $\bar{\delta_\forall}$ and $\bar{\delta_\exists}$, respectively. Formally, $\bar{\delta_\forall} : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ consists of all the transition rules of the form $(s, a) \to ((s_1, a_1, d_1), (s_2, a_2, d_2))$, where $s \in S_\forall$, not occurring in δ. Analogously, $\bar{\delta_\exists} : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ consists of all the transition rules of the form $(s, a) \to ((s_1, a_1, d_1), (s_2, a_2, d_2))$, where $s \in S_\exists$, for which there is no transition rule of the form $(s, a) \to ((s_1, a_1, d_1), \cdot)$ or $(s, a) \to (\cdot, (s_2, a_2, d_2))$ occurring in δ. Let $T_\forall = \bigcup_{\tau\in\bar{\delta_\forall}} f(\tau)$ and $T_\exists = \bigcup_{\tau\in\bar{\delta_\exists}} f(\tau)$, where, assuming that $\tau = (s, a) \to ((s_1, a_1, d_1), (s_2, a_2, d_2))$,

$$f(\tau) = \begin{cases} \bigcup_{x,y\in\Lambda}\{(x, (s, a), y) \to ((x, a_1, (s_1, y)), (x, a_2, (s_2, y)))\}, & \text{if } d_1 = +1, d_2 = +1\\ \bigcup_{x,y\in\Lambda}\{(x, (s, a), y) \to ((x, a_1, (s_1, y)), ((s_2, x), a_2, y))\}, & \text{if } d_1 = +1, d_2 = -1,\\ \bigcup_{x,y\in\Lambda}\{(x, (s, a), y) \to (((s_1, x), a_1, y), (x, a_2, (s_2, y)))\}, & \text{if } d_1 = -1, d_2 = +1,\\ \bigcup_{x,y\in\Lambda}\{(x, (s, a), y) \to (((s_1, x), a_1, y), ((s_2, x), a_2, y))\}, & \text{if } d_1 = -1, d_2 = -1.\end{cases}$$

Moreover, in the definition of Q_{trans} we make use of the function $g : S \to \{\exists, \forall\}$, where $g(s) = \exists$ if $s \in S_\exists$; otherwise, $g(s) = \forall$. We use the binary operator $\odot^s$, where $s \in S$, which is defined as $\vee$, if $s \in S_\exists$, and as $\wedge$, if $s \in S_\forall$. We also make use of the shorthand $is_x(X) \equiv a(X)$ in case that $x = a \in \Lambda$, and $is_x(X) \equiv a(X) \wedge state[\#_s](X)$ in case that $x = (s, a) \in (S \times \Lambda)$. Q_{trans} is defined as follows:

$$\bigvee_{(x,y,z)\to((x_1,y_1,z_1),(x_2,y_2,z_2))\in(T_\forall\cup T_\exists)} \overset{s}{\odot} \bullet_{i\in\{1,2\}} \exists T \exists N_1 \exists N_2 \exists L^T \exists M^T \exists R^T \exists L^{N_i} \exists M^{N_i} \exists R^{N_i}$$
$$\Big(conf_{g(s)}(0,1,T,N_1,N_2) \wedge \Phi(T,N_i,L^T,M^T,R^T,L^{N_i},M^{N_i},R^{N_i}) \wedge$$
$$is_x(L^T) \wedge is_y(M^T) \wedge is_z(R^T) \wedge$$
$$is_{x_i}(L^{N_i}) \wedge is_{y_i}(M^{N_i}) \wedge is_{z_i}(R^{N_i})\Big).$$

5. Finally, $Q_{inertia}$ is defined as

$$\bigvee_{x\in\{\exists,\forall\}} \bigvee_{(a,a')\in\Lambda\times\Lambda, a\neq a'} \bigvee_{i\in\{1,2\}} \exists T \exists N_1 \exists N_2 \exists L^T \exists M^T \exists R^T \exists L^{N_i} \exists M^{N_i} \exists R^{N_i}$$
$$\Big(conf_{g(s)}(0,1,T,N_1,N_2) \wedge \Phi(T,N_i,L^T,M^T,R^T,L^{N_i},M^{N_i},R^{N_i}) \wedge$$
$$notCursor(M^T) \wedge a(M^T) \wedge a'(M^{N_i})\Big).$$

Notice that, by employing the above query, we do not guarantee the inertia of the first and the last cells of the tape. This is not a problem since, by assumption, M never touches those cells, and also starts with its cursor at the second cell.

The definition of Q is now complete. It is not difficult to verify that indeed Q is of bounded treewidth.

Correctness. By construction, and by the assumption that a rejecting configuration does not have a subsequent configuration, but an accepting one has only itself as a subsequent configuration, we get that $D \cup \Sigma \models Q$ iff M rejects. More precisely, if M accepts, then there exists $I \in chase(D, \Sigma)$ which encodes an accepting computation tree of M. It therefore does not violate any of the constraints imposed by the queries and thus, $I \not\models Q$, which implies that $D \cup \Sigma \not\models Q$. Conversely, if M rejects, then, for each $I \in chase(D, \Sigma)$, $I \models Q_{trans}$. Towards a contradiction, assume that M rejects and there exists $I \in chase(D, \Sigma)$ such that $I \not\models Q_{trans}$. By construction, I encodes an infinite tree of configurations. But since a rejecting configuration does not have a subsequent configuration (by assumption), necessarily at least one disjunct of Q_{trans} will be entailed by I which is a contradiction. Therefore, $D \cup \Sigma \models Q$, and the claim follows. □

Another key class of queries is the class of CQs of *bounded hypertree-width (BHTWQs)*; see, e.g. Gottlob et al. [2002].The hypertree-width is a measure of how close to acyclic a hypergraph

	Combined Complexity	Bounded Arity	Fixed Theory	Data Complexity
DID	2ExpTime LB: Thm. 4.23	ExpTime	ExpTime LB: Thm. 4.25	co-NP LB: [Calvanese et al., 2013, Thm. 4.5]
L	2ExpTime	ExpTime	ExpTime	co-NP
ML	2ExpTime	ExpTime	ExpTime	co-NP
G	2ExpTime	ExpTime	ExpTime	co-NP
FG	2ExpTime	2ExpTime LB: Thm. 4.24	ExpTime	co-NP UB: Thm. 4.12
WG	2ExpTime	ExpTime UB: Thm. 4.21	ExpTime	ExpTime LB: [Calì et al., 2013, Thm. 4.1]
WFG	2ExpTime UB: Thm. 4.10	2ExpTime	ExpTime UB: Thm. 4.22	ExpTime UB: Thm. 4.13

Table 4.2: The complexity of (U)ACQ-Answering under guarded-based classes of DTGDs.

is, analogous to treewidth for graphs. The hypertree-width of a CQ is less than or equal to its treewidth. Since all the upper bounds in Table 4.1 hold for arbitrary (U)CQs, we get that arbitrary (U)CQs, (U)CQs of bounded treewidth and (U)CQs of bounded hypertree-width are indistinguishable w.r.t. the complexity of query answering under our guarded-based classes of DTGDs.

4.7 Answering Acyclic Queries

In this section, we focus on another important subclass of conjunctive queries, which has been proposed with the aim of reducing the complexity of query evaluation and query containment, namely the class of *acyclic CQs (ACQs)*; see, e.g. Chekuri and Rajaraman [2000]. Our goal is to understand how the complexity of (U)CQ answering under our guarded-based classes of DTGDs is affected if we focus on acyclic queries. Recall that the acyclicity of a CQ q is defined via the acyclicity of its hypergraph $\mathcal{H}(q)$.

We start by summarizing our results, and then proceed with our new upper and lower complexity bounds in Section 4.7.2 and 4.7.3, respectively.

4.7.1 Overview

Table 4.2 summarizes the complexity results for answering (U)ACQs under the various DTGD formalisms considered in this paper. Compared with the results in Table 4.1, it is immediately apparent that the combined and the data complexity do not change if we restrict ourselves

to acyclic queries. The complexity decreases from 2ExpTime to ExpTime for the less expressive classes of DTGDs, namely DIDs, (multi-)linear and guarded, in the case of predicates of bounded arity, and also for all the classes if we consider a fixed set of DTGDs. The novel results of this section follow:

Upper Bounds:

1. UACQ-Answering under weakly-guarded sets of DTGDs is in ExpTime if we focus on predicates of bounded arity (Theorem 4.21) — this is shown by a reduction to satisfiability of the loosely-guarded fragment of first-order logic, and then exploit the fact that the latter is in ExpTime in case of predicates of bounded arity, as shown by Grädel [1999]; and

2. UACQ-Answering under fixed weakly-frontier-guarded sets of DTGDs is in ExpTime (see Theorem 4.22) — by a reduction to UCQ-Answering under (non-fixed) GFO sentences with predicates of bounded arity, where in the constructed (partially acyclic) query the cyclic part is fixed.

Lower Bounds:

1. ACQ-Answering under DIDs is 2ExpTime-hard in combined complexity (Theorem 4.23) — this is established by simulating the behavior of an alternating exponential space Turing machine by means of a set of DIDs and an ACQ;

2. ACQ-Answering under frontier-guarded DTGDs is 2ExpTime-hard, even if we consider predicates of bounded arity (Theorem 4.24) — this is shown by a reduction from CQ answering under frontier-guarded DTGDs; and

3. ACQ-Answering under fixed sets of DIDs is ExpTime-hard (Theorem 4.25) — again by simulating the behavior of an alternating Turing machine, but one that only uses linear space.

The ExpTime upper bound for guarded DTGDs in the case of predicates of bounded arity, and for fixed sets of weakly-frontier-guarded DTGDs, and the ExpTime lower bound for fixed sets of DIDs, close the picture of the computational complexity of our problem for DIDs, (multi-)linear and guarded in the case of bounded arity, and for all classes in the case of a fixed set of DTGDs. The missing upper bounds are inherited from results of Section 4.5; for details see Table 4.2. Finally, the following results from the literature provide us with optimal lower bounds for the data complexity of our problem, and the entire picture of the complexity of (U)ACQ answering is completed.

Inherited Results:

1. ACQ-Answering under DIDs is co-NP-hard in data complexity [Calvanese et al., 2013, Theorem 4.5] — the CQ employed in the proof of this result is of the form

$$\exists X \exists Y_1 \exists Y_2 \exists Z_1 \exists Z_2 (\bigwedge_{i \in \{1,2\}} (p_i(X, Y_i) \wedge s(Y_i) \wedge r_i(X, Z_i) \wedge t(Z_i))),$$

   which is clearly acyclic; and

2. ACQ-Answering under weakly-guarded DTGDs is ExpTime-hard in data complexity [Calì et al., 2013, Theorem 4.1] — the CQ used in the proof of this result consists of a single atom, and thus is trivially acyclic.

From the results of this section, we observe that the acyclicity of the query does not make the query answering problem easier w.r.t. the combined, or the data complexity. However, in the cases of bounded arity (for the less expressive classes) and fixed sets of DTGDs, the complexity decreases from 2ExpTime to ExpTime. Let us now proceed to the formal proofs of our results.

4.7.2 Upper Bounds

We start this section by showing the following:

Theorem 4.21. (U)ACQ-Answering *under weakly-guarded sets of DTGDs is in ExpTime in case of predicates of bounded arity.*

Proof. Consider a database D, a weakly-guarded set Σ of DTGDs with predicates of bounded arity, and an (U)ACQ Q. The proof is by reduction to satisfiability of LGFO sentences. More precisely, we are going to construct, in polynomial time, a set Σ' of DTGDs such that $\Psi_{D,\Sigma',Q} = (D \wedge \Sigma' \wedge \neg Q)$ falls in LGFO, the arity of the underlying schema is increased by two (and therefore is bounded), and $D \cup \Sigma \models Q$ iff $\Psi_{D,\Sigma',Q}$ is unsatisfiable. Since the problem of deciding whether an LGFO sentence is in ExpTime in case of predicates of bounded arity, as shown by Grädel [1999], the claim follows.

Assume that $dom(D) = \{c_1, \ldots, c_n\}$, and let $wgp(\Sigma) \subseteq sch(\Sigma)$ be the predicates that appear in at least one $\sigma \in \Sigma$ as the predicate of the weak-guard of σ, that is, the atom that guards the body-variables of σ that appear at affected positions. The set Σ' is defined as follows. Given a DTGD $\sigma \in \Sigma$, let $\{W_1, \ldots, W_m\}$ be the set of variables occurring in at least one non-affected position in Σ. If $m = 0$ (which means that σ is guarded), then let $\hat{\sigma} = \sigma$; otherwise, assuming that σ is of the form $\phi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})$, $g(\mathbf{t}) \in body(\sigma)$ is the weak-guard of σ, and $\phi(\mathbf{X})^- = \phi(\mathbf{X}) \setminus \{g(\mathbf{t})\}$, we define the DTGD $\hat{\sigma}$ as follows:

$$\left(\bigwedge_{1 \leqslant i < j \leqslant m} \hat{g}(W_i, W_j, \mathbf{t}) \wedge \phi(\mathbf{X})^- \right) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y}).$$

Moreover, for each k-ary predicate $p \in wgp(\Sigma)$, we define the TGD σ_p as follows:

$$p(X_1, \ldots, X_k) \rightarrow \bigwedge_{1 \leqslant i,j \leqslant n} \hat{p}(c_i, c_j, X_1, \ldots, X_k).$$

The desired set of DTGDs is defined as

$$\Sigma' = \left(\bigcup_{\sigma \in \Sigma} \{\hat{\sigma}\} \cup \bigcup_{p \in wgp(\Sigma)} \{\sigma_p\} \right).$$

It is easy to verify that $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models Q$, which in turn implies that $D \cup \Sigma \models Q$ iff $\Psi_{D,\Sigma',Q}$ is unsatisfiable. Intuitively speaking, a clique (with self-loops) of all the constants of $dom(D)$ is attached to all weak-guards of Σ, and all the variables at non-affected positions must map to an element of such a clique. As these non-affected variables can, by definition, map only to constants of $dom(D)$, one can see the above construction as a simulation of a partial grounding of Σ.

It remains to show that $\Psi_{D,\Sigma',Q}$ falls in LGFO; in fact, we are going to show that $\Psi_{D,\Sigma',Q}$ is "almost" an LFGO sentence which, as we explain below, suffices for our purposes. Since D is a conjunction of ground atoms, it immediately falls in GFO, and thus in LGFO. The set $\bigcup_{p \in wgp(\Sigma)} \{\sigma_p\}$ trivially forms a GFO (and thus an LGFO) sentence. A DTGD $\hat{\sigma}$ as above can be rewritten as

$$\forall \mathbf{X} \forall W_1 \ldots \forall W_m \left(\left(\bigwedge_{1 \leqslant i < j \leqslant m} \hat{g}(W_i, W_j, \mathbf{t}) \right) \rightarrow (\phi(\mathbf{X})^- \rightarrow \exists \mathbf{Y} \psi(\mathbf{X}, \mathbf{Y})) \right).$$

The above sentence does not fall in LGFO due to the fact that the existentially quantified variables of $\mathbf{Y}$ are not necessarily loosely-guarded. However, the satisfiability algorithm proposed in [Grädel, 1999] is able to treat sentences which are "almost" LGFO sentences as the one above, even if the existentially quantified variables are not loosely-guarded, without increasing the complexity — this is explicitly stated in a remark on page 8 in [Grädel, 1999]. Finally, by hypothesis, the query Q is acyclic, which implies that there exists a UCQ which is equivalent to Q and falls in GFO, and hence in LGFO; see, e.g. [Gottlob et al., 2003]. It is clear that we have polynomially reduced our problem into a problem which is in ExpTime, and the claim follows. □

Although the above result does not hold for weakly-frontier-guarded DTGDs, the next theorem shows that we get an ExpTime upper bound if we focus on fixed weakly-frontier-guarded sets of DTGDs.

Theorem 4.22. UACQ-Answering *under fixed weakly-frontier-guarded sets of DTGDs is feasible in ExpTime.*

Proof. Consider a database D, a fixed weakly-frontier-guarded set of DTGDs, and an (U)ACQ Q. First, we reduce our problem to UCQ answering under GFO sentences in the usual way: (1) partially ground the DTGDs of Σ in such a way that the non-affected variables are replaced

by all possible combinations of constants of $dom(D)$, yielding an equivalent set of frontier-guarded DTGDs Σ'; and (2) in linear time reduce the problem of deciding whether $D \cup \Sigma' \models Q$ to the problem of deciding whether the GFO formula $(D \wedge \Psi^1_{\Sigma'})$ entails the UCQ $(Q \vee \Psi^2_{\Sigma'})$ by employing Lemma 4.11. Since Σ is fixed, we immediately get that Σ' is of polynomial size w.r.t. $|dom(D)|$. Thus, by construction, $(D \wedge \Psi^1_{\Sigma'})$ and $(Q \vee \Psi^2_{\Sigma'})$ are of polynomial size, and they use only predicates of bounded arity.

Before we proceed further, we need to recall some details regarding the problem of UCQ answering under GFO as presented in [Bárány et al., 2014].

Given a GFO formula ϕ and a UCQ Q', the way that the problem of deciding whether $\phi \models Q'$ is tackled is as follows: (1) Q' is rewritten as a GFO formula $\chi(Q')$ (this procedure is called *treeification*) such that $\phi \models Q'$ iff $\phi \models \chi(Q')$; and (2) it is checked whether the GFO formula $(\phi \wedge \neg\chi(Q'))$ is unsatisfiable. The latter check is feasible in time $2^{O((n+|\phi|+|\chi(Q')|)\cdot m^m)}$, where $|\cdot|$ denotes the size of a formula, n is the number of predicates occurring in $(\phi \wedge \neg\chi(Q'))$, and m is the maximum arity over all these predicates. Let us now explain how the desired upper bound is obtained.

Clearly, it suffices to check whether the formula $(D \wedge \Psi^1_{\Sigma'} \wedge \neg\chi(Q) \wedge \neg\chi(\Psi^2_{\Sigma'}))$ is unsatisfiable. Recall that $(D \wedge \Psi^1_{\Sigma'})$ is of polynomial size, and also the maximum arity of the underlying schema is bounded. It remains to show that $|\chi(Q)|$ and $|\chi(\Psi^2_{\Sigma'})|$ are of polynomial size. Since Q is acyclic, we know that the treeification of Q is feasible in polynomial time Bárány et al. [2014], and thus $|\chi(Q)|$ is polynomial. By construction, since the set Σ of DTGDs is fixed, we get that $|\Psi^2_{\Sigma'}|$ is constant. Hence, by definition of the treeification procedure, $|\chi(\Psi^2_{\Sigma'})|$ is constant, and the claim follows. □

This concludes the upper bound section for acyclic queries.

4.7.3 Lower Bounds

We start this section by showing that ACQ answering under DIDs is 2ExpTime-hard. The construction is rather tedious, and hinges on the fact that, using a polynomial number of DIDs, it is possible to generate and store an exponential number of integers in binary form that can be used to index the exponentially many tape cells of an alternating ExpSpace Turing machine. Then, the computation of such a machine can be simulated by means of a set of DIDs and a CQ, and the 2ExpTime-hardness result follows from the fact that alternating ExpSpace coincides with 2ExpTime.

Theorem 4.23. ACQ-Answering *under DIDs is 2ExpTime-hard in the combined complexity.*

Proof. The proof is by a reduction from the non-acceptance problem of an alternating exponential space Turing machine M on the empty input. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ being the blank

symbol, $\delta : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that M is well-behaved and never tries to read beyond its tape boundaries, always halts, and uses exactly 2^n tape cells. Furthermore, we assume that a rejecting configuration does not have a subsequent configuration, while an accepting configuration has only itself as a subsequent configuration. Finally, we assume that $s_0 \in S_\exists$, and also that every universal configuration is followed by two existential configurations and vice versa. The above assumptions can be made, without sacrificing the generality of our proof, since the non-acceptance problem of M remains 2ExpTime-hard.

Our goal is to construct a database D, a set Σ of DTGDs, and an (U)ACQ Q such that $D \cup \Sigma \models Q$ iff M rejects, and $\mathsf{N}(\Sigma)$ is a set of DIDs. By exploiting the construction in the proof of Lemma 4.6, a database D', a set Σ', and an ACQ q can be constructed in polynomial time such that $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models q$, and $\mathsf{N}(\Sigma')$ is a set of DIDs, which proves the claim. We proceed with the construction of D, Σ and Q.

The general idea of the proof is to construct trees, which encode possible computation trees of M, by chasing D with Σ, and then use the query Q to check the consistency of those trees. In order to represent configurations of the Turing machine M, we will use atoms of the form $conf[s](b_1, \dots, b_n, a, h, t, p, n_1, n_2)$, where $s \in S$ is the state of the encoded configuration and is part of the predicate, $(b_1, \dots, b_n) \in \{0, 1\}^n$ is an integer of $\{0, \dots, 2^n - 1\}$ in binary encoding which represents the index of the encoded cell, $h \in \{0, 1\}$ and $h = 1$ iff the cursor of M is at the encoded cell, and t, p, n_1 and n_2 represent the current (t for this), the previous and the next two configurations, respectively. For example, assuming that $n = 3$, $conf[s](1, 0, 1, \sqcup, 1, z_1, z_2, z_3, z_4)$, where $\{z_1, \dots, z_4\} \subset \mathbf{N}$, says that the state of the configuration z_1 (labeled nulls are used to represent configurations) is s, the fifth cell contains the blank symbol, the cursor is at the fifth cell, the previous configuration of z_1 is z_2, and the next two configurations of z_1 are z_3 and z_4.

The Database D. Let the database D contain a single atom $begin(0, 1, \sqcup, \mathbf{0}^{2n}, \mathbf{1}^n, 1, c)$; for a constant x, $\mathbf{x}^k$ denotes the sequence $x, \dots, x$ with k occurrences of x. The first three constants 0, 1 and $\sqcup$ will allow us to have access to the symbols of Λ without explicitly mention them in the DTGDs; recall that DIDs are constant-free. The $2n$ zeros and n ones are auxiliary constants which will allow us to generate, for each configuration of M, the 2^n integers in binary encoding that will be used to index the tape cells that M can touch. The constant 1 will be used to ensure that exactly one tape cell is pointed by the cursor. Finally, c is a special constant which represents the initial configuration.

The Set Σ. We proceed now with the construction of Σ. In the sequel, for brevity, we write $\mathbf{X}^k$ for the sequence of variables $X_1, \dots, X_k$.

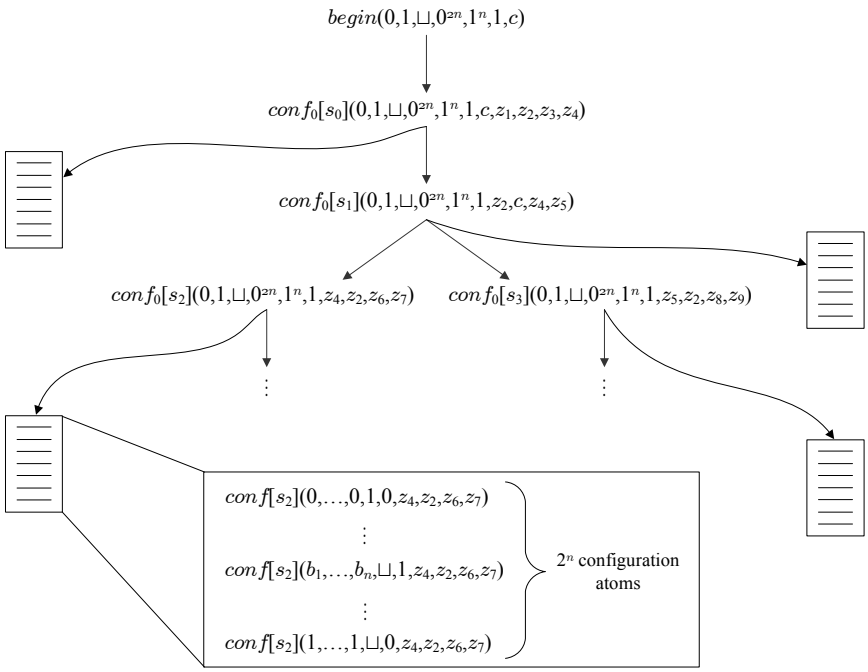


Figure 4.6: Computation tree.

- We first construct the initial configuration with the TGD

  $$begin(Z,O,B,\mathbf{Z}^{2n},\mathbf{O}^n,H,T) \rightarrow \exists P \exists N_1 \exists N_2\, conf_0[s_0](Z,O,B,\mathbf{Z}^{2n},\mathbf{O}^n,H,T,P,N_1,N_2).$$

  Notice that in the head of the above TGD we use the auxiliary predicate $conf_0[s]$, where $s \in S$, and not the predicate $conf[s]$. This is because we first want to generate all the possible trees that may encode a computation tree of M, where the nodes are labeled with auxiliary atoms of the form $conf_0[s](0,1,\sqcup,\mathbf{0}^{2n},\mathbf{1}^n,1,z_1,z_2,z_3,z_4)$. Such an atom, which is associated with the configuration z_1, contains all the auxiliary constants that will allow us to generate, via an additional set of DTGDs, all the 2^n atoms of the form $conf[s](b_1,\ldots,b_n,a,h,z_1,z_2,z_3,z_4)$ which perfectly describe the configuration z_1. The above informal description is illustrated in Figure 4.6.

- The trees labeled by atoms of the form $conf_0[s](\ldots)$ are generated by the following DTGDs: for each $s \in S_\forall$,

  $$conf_0[s](Z,O,B,\mathbf{Z}^{2n},\mathbf{O}^n,H,T,P,N_1,N_2) \rightarrow \bigvee_{(s_1,s_2)\in S\times S} \exists N_3 \exists N_4 \exists N_5 \exists N_6\, conf_0[s_1](Z,O,B,\mathbf{Z}^{2n},\mathbf{O}^n,H,N_1,T,N_3,N_4), conf_0[s_2](Z,O,B,\mathbf{Z}^{2n},\mathbf{O}^n,H,N_2,T,N_5,N_6),$$

  and for each $s \in S_\exists$,

  $$conf_0[s](Z,O,B,\mathbf{Z}^{2n},\mathbf{O}^n,H,T,P,N_1,N_2) \rightarrow \bigvee_{s'\in S} \bigvee_{i\in\{1,2\}} \exists N_3 \exists N_4\, conf_0[s'](Z,O,B,\mathbf{Z}^{2n},\mathbf{O}^n,H,N_i,T,N_3,N_4).$$

- The 2^n atoms, which perfectly describe a certain configuration, are generated as follows. First, we employ a stratified set of DIDs, consisting of n levels, in order to generate all the necessary tuples, and store them in predicates of the form $conf_n[s]$. The initial $conf_0[s]$-atom contains n ones and $2n$ zeros. Each application of a rule will generate two new atoms, one where we propagate a zero to the next bit of the binary address (represented by **X**) and one where we propagate a one. The disjunction is needed to generate the cursor position. Either the head is in the left subtree of the binary address, or it is in the right subtree. This is the reason why we need an additional n zeros, to mark where the head is not. To formalize this intuition, for each $i \in [n-1]$, and for each $s \in S$ we construct the following DID:

$$conf_i[s](Z, O, B, \mathbf{Z}^{2n-2i}, \mathbf{O}^{n-i}, \mathbf{X}^i, H, T, P, N_1, N_2) \rightarrow \phi_1 \vee \phi_2,$$

where

$$\begin{aligned}\phi_1 \quad = \quad & conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, Z_{2n-2i-1}, H, T, P, N_1, N_2), \\ & conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, O_{n-i}, Z_{2n-2i-1}, T, P, N_1, N_2)\end{aligned}$$

and

$$\begin{aligned}\phi_2 \quad = \quad & conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, Z_{2n-2i-1}, Z_{2n-2i}, T, P, N_1, N_2), \\ & conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, O_{n-i}, H, T, P, N_1, N_2).\end{aligned}$$

Now, having the relation $conf_n[s]$, for each $s \in S$, in place, we are ready to generate the completed $conf[s]$ atoms by guessing the cell content of each cell. This can be done via the following DTGDs: for each $s \in S$,

$$conf_n[s](Z, O, B, \mathbf{X}^n, H, T, P, N_1, N_2) \rightarrow \bigvee_{a \in \Lambda} conf[s](\mathbf{X}^n, f(a), H, T, P, N_1, N_2),$$

where $f(0) = Z$, $f(1) = O$ and $f(\sqcup) = B$.

The construction of Σ is now completed. It is easy to verify that N(Σ) is a set of DIDs. This is because, for each $\sigma \in \Sigma$, σ is constant-free, $|body(\sigma)| = 1$, and there are no repeated variables in the atoms occurring in σ.

The (U)ACQ Q. It remains to check, via the (U)ACQ Q, that each model of $D \cup \Sigma$ encodes a tree which represents a valid computation tree of M. Roughly speaking, Q consists of the following disjuncts:

1. $Q_{initial}$ for checking that in the initial configuration the cursor is at the first cell, and the tape is empty;

2. Q_{trans} for checking the consistency w.r.t. the transition function of M;

3. Q_{move} for checking that the cursor moves either to the left or to the right during a transition; and

4. $Q_{inertia}$ for checking that the tape cells not under the cursor keep their old values during a transitions.

Let us now formalize the components of Q.

1. $Q_{initial}$ is defined as (recall that $c \in \mathbf{C}$ represents the initial configuration)

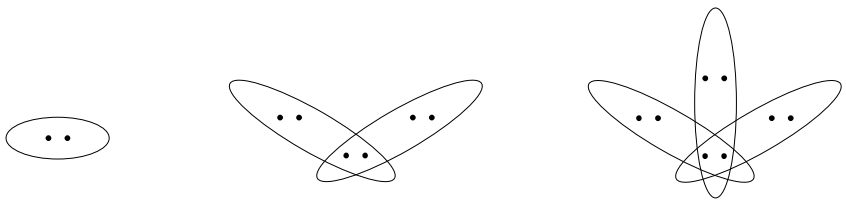
$$\exists C \exists P \exists N_1 \exists N_2 \, (conf[s_0](\mathbf{0}^n, C, 0, c, P, N_1, N_2)) \vee \bigvee_{a \in \Lambda \setminus \{\sqcup\}} \exists \mathbf{X}^n \exists H \exists P \exists N_1 \exists N_2 \, (conf[s_0](\mathbf{X}^n, a, H, c, P, N_1, N_2)) \, .$$

   Notice that the first disjunct encodes the fact that the cursor is not at the first cell. This is sufficient since, by construction, exactly one cell is pointed to by the cursor; cf. the rules used to generate $conf_{i+1}[s]$.

2. With Q_{trans} we need to check that during a transition of M the state and the cell content has changed appropriately, and the head has moved to the correct position. Note that two subsequent cell positions agree on some initial portion of their binary encoding, which for the preceding cell is then followed by a 0 and then all 1s, and the other way around for the succeeding cell. We can thus connect the cell under the head in one configuration and the same cell in the next configuration, plus the appropriate adjacent cell in the next configuration. In the definition of Q_{trans}, we make use of the functions $f, g : \{-1, +1\} \to \{0, 1\}$, where $f(-1) = g(+1) = 1$ and $f(+1) = g(-1) = 0$. Also, we make use of the binary operator $\odot^s$, where $s \in S$, which is defined as $\vee$ in case that $s \in S_\exists$, and as $\wedge$ in case that $s \in S_\forall$. Q_{trans} is defined as follows:

$$\bigvee_{(s,a) \to ((s_1,a_1,d_1),(s_2,a_2,d_2)) \notin \delta} \overset{s}{\odot}_{i \in \{1,2\}} \bigvee_{0 < j \leqslant n} \exists \mathbf{X}^{n-j} \exists T \exists P \exists N_1 \exists N_2 \exists N_1' \exists N_2' \exists N_1'' \exists N_2'' \exists C$$
$$\big(conf[s](\mathbf{X}^{n-j}, f(d_i), g(d_i)^{j-1}, a, 1, T, P, N_1, N_2) \wedge$$
$$conf[s_i](\mathbf{X}^{n-j}, f(d_i), g(d_i)^{j-1}, a_i, 0, N_i, T, N_1', N_2') \wedge$$
$$conf[s_i](\mathbf{X}^{n-j}, g(d_i), f(d_i)^{j-1}, C, 1, N_i, T, N_1'', N_2'')\big) \, .$$

3. The idea underlying Q_{move}, is to check that in a certain configuration the cursor points the k-th cell, while in the previous configuration the cursor is neither at the $(k+1)$-th



(a) (b) (c)

Figure 4.7: The UCQ Q in the proof of Theorem 4.23 is acyclic.

cell, nor at the $(k-1)$-th cell. The formal definition follows:

$$\bigvee_{0<i,j\leqslant n} \bigvee_{(s,s')\in S\times S} \exists \mathbf{X}^{n-i} \exists C \exists T \exists P \exists N_1 \exists N_2 \exists C' \exists P' \exists N_1' \exists N_2'$$
$$\exists \mathbf{Y}^{n-j} \exists C'' \exists P'' \exists N_1'' \exists N_2'' \exists C''' \exists P''' \exists N_1''' \exists N_2'''$$
$$\big(conf[s](\mathbf{X}^{n-i}, 0, \mathbf{1}^{i-1}, C, 1, T, P, N_1, N_2) \wedge$$
$$conf[s'](\mathbf{X}^{n-i}, 1, \mathbf{0}^{i-1}, C', 0, P, P', N_1', N_2') \wedge$$
$$conf[s](\mathbf{Y}^{n-j}, 1, \mathbf{0}^{j-1}, C'', 1, T, P'', N_1'', N_2'') \wedge$$
$$conf[s'](\mathbf{Y}^{n-j}, 0, \mathbf{1}^{i-1}, C''', 0, P, P'', N_1''', N_2''')\big).$$

Intuitively, the first and the third atom of a disjunct in the above query access the cell k, while the second and the fourth atom access the cell $(k+1)$ and $(k-1)$, respectively. Notice that, although we do not force $(\mathbf{X}^{n-i}, 0, \mathbf{1}^{i-1})$ and $(\mathbf{Y}^{n-j}, 1, \mathbf{0}^{j-1})$ to have the same image during the evaluation of the query (otherwise, the above query will be cyclic), this is guaranteed by the fact that only one cell is pointed by the cursor (this holds by construction).

4. Finally, $Q_{inertia}$ is defined as

$$\bigvee_{(s,s')\in S\times S} \bigvee_{(a,a')\in\Lambda\times\Lambda, a\neq a'} \bigvee_{i\in\{1,2\}} \bigvee_{j\in\{0,1\}} \exists \mathbf{X}^n \exists T \exists P \exists N_1 \exists N_2 \exists N_1' \exists N_2'$$
$$\big(conf[s](\mathbf{X}^n, a, 0, T, P, N_1, N_2) \wedge conf[s'](\mathbf{X}^n, a', j, N_i, T, N_1', N_2')\big)$$

This completes the construction of Q. It is not difficult to verify that every disjunct of Q is an ACQ. In fact, for each disjunct q of $Q_{initial}$ (resp., $Q_{inertia}$, $Q_{move} \vee Q_{inertia}$), the hypergraph $\mathcal{H}(q)$ is of the form shown in Figure 4.7(a) (resp., 4.7(b), 4.7(c)). It is clear that the GYO-reduct of those hypergraphs is $(\emptyset, \emptyset)$, and thus Q is acyclic.

Correctness. By construction, and the assumption that a rejecting configuration does not have a successor configuration, while an accepting configuration has only itself as a successor, it is not difficult to see that $D \cup \Sigma \models Q$ if and only if M rejects I (see the argument given in the proof of Theorem 4.20 which shows that the employed construction is correct). Note that we have constructed a UACQ Q. However, using the fact that Lemma 4.6 preserves acyclicity,

we immediately get that the lower bound presented for UACQs above also holds for ACQs. This completes the proof. □

We now focus on frontier-guarded DTGDs, and show that query answering remains hard for 2ExpTime, even if we consider acyclic queries and predicates of bounded arity.

Theorem 4.24. ACQ-Answering *under frontier-guarded DTGDs is 2ExpTime-hard, even for predicates of bounded arity.*

Proof. The claim can be easily established by exploiting the fact that a CQ can be conceived as a frontier-guarded TGD, and give a reduction from CQ answering under frontier-guarded DTGDs which is 2ExpTime-hard, even for predicates of bounded arity. Consider a database D, a set Σ of DTGDs, where the arity of the predicates of $sch(\Sigma)$ is bounded by an integer constant, and a CQ $q = \exists \mathbf{X}\, \phi(\mathbf{X})$. It is straightforward to see that $D \cup \Sigma \models q$ iff $D \cup \Sigma \cup \{\sigma_q\} \models q'$, where σ_q is the frontier-guarded TGD $\phi(\mathbf{X}) \rightarrow p$, with p be 0-ary predicate not occurring in $sch(\Sigma)$, and $q' = p$. Since q' is trivially acyclic, the claim follows. □

The last lower bound that we establish for answering ACQs is the ExpTime-hardness when we focus on fixed sets of DIDs. This is done by simulating the behavior of a alternating linear space Turing machine. The desired result follows since, already, alternating LinSpace coincides with ExpTime.

Theorem 4.25. ACQ-Answering *under fixed sets of DIDs is ExpTime-hard.*

Proof. The proof is by a reduction from the non-acceptance of an alternating linear space Turing machine M on input I. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ being the blank symbol, $\delta : S \times \Lambda \rightarrow (S \times \Lambda \times \{-1, 0, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We make the same assumptions for M as in the proof of Theorem 4.20, with the difference that now M uses exactly n tape cells, where $n = |I|$. We are going to construct a database D, a set Σ of DTGDs, and an (U)ACQ Q such that $D \cup \Sigma \models Q$ iff M rejects I, and $\mathsf{N}(\Sigma)$ is a fixed set of DIDs. By Lemma 4.6, the obtained lower bound holds even if we focus on ACQs.

The idea of the proof is along the lines of the one of Theorem 4.20, that is, to construct trees, which encode possible computation trees of M on the input string I, by chasing D with Σ, and then exploit Q to check their consistency. On each configuration node v, which represents the configuration C_v of M, we attach a chain of length n, which mimics the tape in C_v, and also a chain of length at most $|S|$, which encodes the state of C_v. The formal construction of D, Σ and Q follows.

The Database D. Let $D = \{conf_\exists(c)\}$, where $c \in \mathbf{C}$ is a special constant which represents the initial configuration (which is, by assumption, existential).

The Set Σ. The predicates that we are going to use are self-explanatory, and thus we proceed with the construction of Σ without describing the predicates of $sch(\Sigma)$.

- Each configuration has two successor configurations, such that a universal configuration is followed by existential configurations, and vice-versa. Note that for existential configurations, we generate two models for the successors, whereas for universal configurations, both are forced to appear in the same model:

$$\begin{aligned} conf_{\exists}(X) &\rightarrow \exists Y\, succ_1(X,Y), conf_{\forall}(Y) \vee \exists Y\, succ_2(X,Y), conf_{\forall}(Y), \\ conf_{\forall}(X) &\rightarrow \exists Y \exists Z\, succ_1(X,Y), conf_{\exists}(Y), succ_2(X,Z), conf_{\exists}(Z). \end{aligned}$$

- Each configuration has a cell- and state-chain which simulates its tape and encodes its state, respectively:

$$\begin{aligned} conf_x(X) &\rightarrow \exists Y\, cell(X,Y), \text{ for each } x \in \{\exists, \forall\}, \\ cell(X,Y) &\rightarrow \exists Z\, cell(Y,Z) \vee end(Y), \\ conf_x(X) &\rightarrow \exists Y\, state(X,Y), \text{ for each } x \in \{\exists, \forall\}, \\ state(X,Y) &\rightarrow \exists Z\, state(Y,Z) \vee end(Y). \end{aligned}$$

- Finally, we guess the content of each cell and the cursor position:

$$\begin{aligned} cell(X,Y) &\rightarrow 0(Y) \vee 1(Y) \vee \sqcup(Y), \\ cell(X,Y) &\rightarrow cursor(Y) \vee notCursor(Y). \end{aligned}$$

The construction of Σ is now completed. It is easy to see that Σ does not depend on M, and also that N(Σ) is a set of DIDs.

The UACQ Q. We now proceed with the construction of Q which ensures that each model of $D \cup \Sigma$ encodes a valid computation tree. Roughly, Q consists of the following disjuncts:

1. $Q_{initial}$ for checking that in the initial configuration the tape contains the input string $I = a_1 \dots a_n$, the state is s_0, and the cursor is at the first cell;
2. Q_{length} for checking that cell-chains are of length n, and state-chains are of length at most $|S|$;
3. Q_{cursor} for checking that exactly one cell is pointed by the cursor;
4. Q_{trans} for checking the consistency w.r.t. the transition function of M; and
5. $Q_{inertia}$ for checking that the tape cells not under the cursor keep their old value during a transition.

We assume a fixed order on S. Given a state $s \in S$, let $\#_s$ be its position in this order. The length of the state-chain which encodes s is $\#_s$. For a predicate $p \in \{cell, state\}$, $p^i(X,Y)$ is used as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1}\,(p(X,Z_1) \wedge \ldots \wedge p(Z_{i-1},Y))$. A useful subquery that we are going to use is

$$length[p]_k(X) \quad \equiv \quad \exists Y \left(p^k(X,Y) \wedge end(Y)\right)$$

stating that there exists a p-chain of length k. Having all the necessary ingredients in place, we are now ready to formalize the components of Q.

1. $Q_{initial}$ is defined as (recall that $c \in \mathbf{C}$ represents the initial configuration)

$$\bigvee_{i\in[n]} \bigvee_{a\in\Lambda\setminus\{a_i\}} \exists X \left(conf_{\exists}(c) \wedge cell^i(c,X) \wedge a(X)\right) \vee \bigvee_{s\in S\setminus\{s_0\}} \left(conf_{\exists}(c) \wedge length[state]_{\#_s}(c)\right) \vee \bigvee_{1<i\leqslant n} \exists X \left(conf_{\exists}(c) \wedge cell^i(c,X) \wedge head(X)\right).$$

2. Q_{length} is defined as

$$\bigvee_{1\leqslant i<n} \exists X\, length[cell]_i(X) \vee \exists X \exists Y\, cell^{n+1}(X,Y) \vee \exists X\, length[state]_{|S|+1}(X).$$

3. Q_{cursor} is defined as

$$\bigvee_{1\leqslant i<j\leqslant n} \exists X \exists Y \exists Z \left(cell^i(X,Y) \wedge cell^j(X,Z) \wedge cursor(Y) \wedge cursor(Z)\right).$$

4. In the definition of Q_{trans}, we use the function $g : S \to \{\exists, \forall\}$, where $g(s) = \exists$ if $s \in S_\exists$; otherwise, $g(s) = \forall$. Moreover, we use the binary operator $\odot^s$, where $s \in S$, which is defined as $\vee$ in case that $s \in S_\exists$, and as $\wedge$ in case that $s \in S_\forall$. Recall that $\bar{\delta_\forall}$ and $\bar{\delta_\exists}$ denote the complement of δ relative to universal and existential, respectively; for details, see the proof of Theorem 4.20. Q_{trans} is defined as follows:

$$\bigvee_{(s,a)\to((s_1,a_1,d_1),(s_2,a_2,d_2))\in(\bar{\delta_\forall}\cup\bar{\delta_\exists})} \overset{s}{\odot} \bigvee_{i\in\{1,2\}} \bigvee_{j\in[n]} \exists X \exists Y \exists Z \exists V \exists W$$
$$\Big(conf_{f(s)}(X) \wedge length[state]_{\#_s}(X) \wedge cell^j(X,Y) \wedge cursor(Y) \wedge a(Y) \wedge succ_i(X,Z) \wedge$$
$$length[state]_{\#_{s_i}}(Z) \wedge cell^j(Z,V) \wedge a_i(V) \wedge cell^{j+d_i}(Z,W) \wedge cursor(W)\Big).$$

5. Finally, $Q_{inertia}$ is defined as follows:

$$\bigvee_{x\in\{\exists,\forall\}} \bigvee_{(a,a')\in\Lambda\times\Lambda, a\neq a'} \bigvee_{i\in[n]} \exists X \exists Y \exists Z_1 \exists Z_2 \left(conf_x(X) \wedge (succ(X,Z) \vee succ_1(X,Z) \vee succ_2(X,Z)) \wedge cell^i(X,Z_1) \wedge cell^i(Y,Z_2) \wedge notCursor(Z_1) \wedge a(Z_1) \wedge a'(Z_2)\right).$$

	Combined Complexity	Bounded Arity	Fixed Theory	Data Complexity
DID	ExpTime LB: Thm. 4.41	PTime LB: Thm. 4.43	in AC_0	in AC_0
L	ExpTime UB: Thm. 4.36	PTime UB: Thm. 4.38	in AC_0 UB: Thm. 4.40	in AC_0
ML	2ExpTime LB: Thm. 4.42	ExpTime LB: Thm. 4.44	co-NP	co-NP LB: [Alviano et al., 2012, Thm. 7]
G	2ExpTime	ExpTime	co-NP	co-NP
FG	2ExpTime	2ExpTime LB: Thm. 4.24	co-NP UB: Thm. 4.12	co-NP
WG	2ExpTime	ExpTime UB: Thm. 4.21	ExpTime	ExpTime LB: [Calì et al., 2013, Thm. 4.1]
WFG	2ExpTime UB: Thm. 4.10	2ExpTime	ExpTime UB: Thm. 4.13	ExpTime

Table 4.3: The complexity of (U)CQ_1-Answering under guarded-based classes of DTGDs.

This completes the construction of Q. It is easy to verify that each disjunct of Q is an acyclic query.

Correctness. By construction, and the assumption that a rejecting configuration does not have a successor configuration, while an accepting configuration has only itself as a successor, it is not difficult to see that $D \cup \Sigma \models Q$ if and only if M rejects I (see the argument given in the proof of Theorem 4.20 which shows that the employed construction is correct). □

4.8 Answering Atomic Queries

We continue our complexity analysis of query answering under the respective guarded-based classes of DTGDs by concentrating on *atomic CQs (CQ_1)*, that is, CQs consisting of a single atom. As usual, we start by summarizing our results, and then proceed with our new upper and lower complexity bounds in Section 4.8.2 and 4.8.3, respectively.

4.8.1 Overview

Let us first clarify that the setting where the given set of DTGDs is fixed coincides with the setting where both the set of DTGDs and the query are fixed (i.e. the data complexity). By fixing the set of DTGDs, we implicitly fix the arity of the underlying schema, and thus the given atomic query. Therefore, the setting where the set of DTGDs is fixed is not relevant for answering

atomic queries. Note also, that a union of atomic queries can always be reduced to a single atomic query: simply add, for each disjunct $q(\mathbf{X})$, a rule $q(\mathbf{X}) \rightarrow p$ to the set of rules and ask for the propositional atomic query p. Clearly this works for all our guarded-based classes of DTGDs, and in all complexity cases. We will therefore not consider the question of UCQ_1-Answering separately. Rather, all the results in this section hold for UCQ_1-Answering as well as CQ_1-Answering. Table 4.3 summarizes the complexity of CQ_1 answering. Compared with the results in Table 4.2, it is apparent that for the formalisms which allow more than one atoms in the body of the DTGDs (i.e. all the considered classes excluding DIDs and linear DTGDs) the complexity of our problem does not change if we focus on atomic CQs (apart from the case where the set of DTGDs is fixed which, as discussed above, is not relevant for CQ_1 answering). However, for DIDs and linear DTGDs the complexity decreases significantly, and in fact we obtain our first tractability results. More precisely, the combined complexity decreases from 2ExpTime to ExpTime, in the case of predicates of bounded arity the complexity decreases from ExpTime to PTime, and the data complexity decreases from co-NP to AC_0. The novel results of this section follow:

Upper Bounds:

1. CQ_1 answering under linear DTGDs is in ExpTime in the combined complexity (Theorem 4.36) — this is shown by exhibiting an alternating algorithm which, at each step of its computation, uses polynomial space;

2. CQ_1 answering under linear DTGDs is in PTime if we focus on predicates of bounded arity (Theorem 4.38) — the alternating algorithm mentioned above uses only logarithmic space when the arity is fixed; and

3. CQ_1 under linear DTGDs is in AC_0 in the data complexity (Theorem 4.40) — by showing that our problem is first-order rewritable, that is, it can be reduced to the problem of evaluating a first-order query over the given database.

Lower Bounds:

1. CQ_1 answering under DIDs is ExpTime-hard in the combined complexity (Theorem 4.41) — this is established by simulating the behavior of an alternating linear space Turing machine;

2. CQ_1 answering under multi-linear DTGDs is 2ExpTime-hard in the combined complexity (Theorem 4.42) — by adapting the proof of Theorem 4.23 in such a way that each disjunct of the employed UCQ can be viewed as a multi-linear DTGD which can be added in the constructed set of DTGDs, and eventually answer a single propositional atom;

3. CQ_1 answering under DIDs is PTIME-hard, even for predicates of bounded arity (Theorem 4.43) — by simulating a logarithmic space Turing machine; and

4. CQ_1 answering under multi-linear DTGDs is EXPTIME-hard, even if we focus on predicates of bounded arity (Theorem 4.44) — by simulating an alternating polynomial space Turing machine; in fact, the proof of this result is an adaptation of the proof of Theorem 4.42 (which in turn is an adaptation of the one of Theorem 4.23) in such a way that the cells are encoded as part of the predicate name, and thus the arity becomes fixed. As we can only encode polynomially many cells in predicate names, we simulate an APSPACE instead of an AEXPSPACE Turing machine.

Several missing upper and lower bounds are obtained from results of Sections 4.5 and 4.7 (for details see Table 4.3). Finally, the following two results from the literature provide us with optimal lower bounds for the data complexity of our problem, and the entire picture of the complexity of CQ_1 answering is completed.

Inherited Results:

1. CQ_1 answering under multi-linear DTGDs is CO-NP-hard in data complexity [Alviano et al., 2012, Theorem 7] — this is shown by a reduction from the 3-UNSAT problem; and

2. CQ_1 answering under weakly-guarded DTGDs is EXPTIME-hard in data complexity [Calì et al., 2013, Theorem 4.1] — in fact this result already holds for the non-disjunctive case.

From the results of this section, we observe that atomic queries do not make the query answering problem easier under the classes of DTGDs which allow more than atoms in the body of a DTGD. However, the complexity in the case of DIDs and linear DTGDs significantly decreases, and in fact we obtain the first (and, to the best of our knowledge, the only known) tractability results for query answering under classes of DTGDs. Let us now proceed with the formal proofs of our results.

4.8.2 Upper Bounds

Up to now, all the upper bounds presented in the previous sections have been established by reducing our problem to the problem of reasoning (either query answering or satisfiability) under expressive computation logics such as the guarded fragment of first-order logic. However, for CQ_1 answering under linear DTGDs this is not possible. All the upper bounds that can be obtained by following the usual approach are not optimal, and thus more refined techniques are needed. Our plan of attack is to reduce CQ_1 answering under linear DTGDs to the problem of deciding whether a proof-tree exists, that is, a tree structure which encodes the finite part of each model of the given database w.r.t. the given set of DTGDs due to which the query

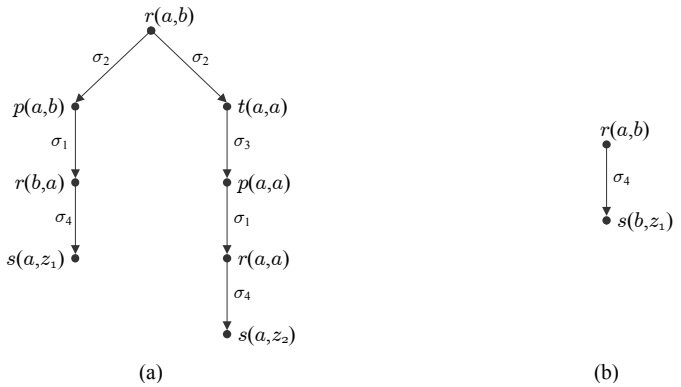


Figure 4.8: Possible proof-trees from $r(a, b)$ w.r.t. Σ.

is entailed, and then exhibit an alternating algorithm for deciding whether such a structure exists.

Proof-Trees

Let us start by introducing the notion of the proof-tree from an atom w.r.t. a set of DTGDs, and establish some key properties. In what follows, we consider linear DTGDs in normal form (see Chapter 2) in order to simplify our technical definitions and proofs.

Definition 4.26 (Proof-tree)**.** *Consider an atom $\underline{a}$, where $dom(\underline{a}) \subset \mathbf{C}$, and a set Σ of linear DTGDs in normal form. Let T be a labeled binary tree $(N, E, \lambda_1, \lambda_2)$, where the nodes of N are labeled by λ_1 with atoms that can be formed using predicates of $sch(\Sigma)$ and terms of $(dom(\underline{a}) \cup \mathbf{N})$, and the edges of E are labeled by $\lambda_2 : E \to \Sigma$. We say that T is a* proof-tree *from $\underline{a}$ w.r.t. Σ if:*

- *The root is labeled by $\underline{a}$;*
- *For each $v \in N$, there exists $\sigma \in \Sigma$ such that each edge of the form $(u, v) \in E$ is labeled by σ, and the out-degree of v is $|head(\sigma)|$, that is, the number of disjuncts in the head of σ;*
- *For each $v \in N$, if v has a single outgoing edge (v, u) labeled by $p(\mathbf{X}) \to \exists Y\, r(\mathbf{X}, Y)$, then there is a homomorphism h such that $h(p(\mathbf{X})) = \lambda_1(v)$, and there exists $h' = h \cup \{Y \to t \mid t \in \mathbf{N}, t \notin dom(h(p(\mathbf{X})))\}$ such that $\lambda_1(u) = h'(r(\mathbf{X}, Y))$; and*
- *For each $v \in N$, assuming that the outgoing edges of v are labeled by a DTGD σ without an existentially quantified variable, there is a homomorphism h such that $h(body(\sigma)) = \lambda_1(v)$, and $\{h(\underline{b}) \mid \underline{b} \in head(\sigma)\} = \{\lambda_1(u) \mid (v, u) \in E\}$.*

A proof-tree T from $\underline{a}$ w.r.t. Σ is valid *w.r.t. to a CQ_1 of the form $\exists \mathbf{X}\, p(\mathbf{X})$ if, for each leaf u of T, there exists a homomorphism h such that $\lambda_1(u) = h(p(\mathbf{X}))$.*

Example 4.27. *Consider the set Σ of linear DTGDs consisting of*

$$\sigma_1 : \; p(X,Y) \to r(Y,X) \qquad \sigma_2 : \; r(X,Y) \to p(X,Y) \vee t(X,X)$$
$$\sigma_3 : \; t(X,Y) \to p(X,Y) \qquad \sigma_4 : \; r(X,Y) \to \exists Z\, s(Y,Z).$$

Possible proof-trees from the atom $r(a,b)$ w.r.t. Σ are shown in Figure 4.8. In particular, proof-tree (a) is valid w.r.t. the atomic queries $\exists X\, s(a,X)$ and $\exists X \exists Y\, s(X,Y)$, while proof-tree (b) is valid w.r.t. $\exists X\, s(b,X)$ and $\exists X \exists Y\, s(X,Y)$.

The following key lemma, established in [Alviano et al., 2012, Lemma 4], shows that for CQ_1 answering under linear DTGDs one can focus on a single database atom:

Lemma 4.28. *Consider a database D, a set Σ of linear DTGDs in normal form, and a CQ_1 q. Then, $D \cup \Sigma \models q$ if and only if there exists $\underline{a} \in D$ such that $\{\underline{a}\} \cup \Sigma \models q$.*

Proof. Let $D = \{\underline{a}_1, \ldots, \underline{a}_n\}$, and $q = \exists \mathbf{X}\, p(\mathbf{X})$. Before we proceed further, we need to establish the following auxiliary claim:

Claim 4.29. *It holds that,*

$$models(D,\Sigma) = \underbrace{\left\{ \bigcup_{i\in[n]} M_i \mid (M_1,\ldots,M_n) \in models(\{\underline{a}_1\},\Sigma) \times \ldots \times models(\{\underline{a}_n\},\Sigma) \right\}}_{\mathcal{M}}.$$

Proof. ($\subseteq$) Fix $M \in models(D,\Sigma)$. Clearly, for each $i \in [n]$, $M \in models(\{\underline{a}_i\},\Sigma)$. Thus, $(M,\ldots,M) \in models(\{\underline{a}_1\},\Sigma) \times \ldots \times models(\{\underline{a}_n\},\Sigma)$ which implies that $M \in \mathcal{M}$.

($\supseteq$) Conversely, fix a tuple $(M_1,\ldots,M_n) \in models(\{\underline{a}_1\},\Sigma) \times \ldots \times models(\{\underline{a}_n\},\Sigma)$, and let $M = \cup_{i\in[n]} M_i$. To show that $M \in models(D,\Sigma)$, it suffices to show that, whenever for a DTGD $\sigma \in \Sigma$ there exists a homomorphism h such that $h(body(\sigma)) \subseteq M$, then $h(head(\sigma)) \in M$. Fix a DTGD $\sigma \in \Sigma$ and a homomorphism h such that $h(body(\sigma)) \subseteq M$. Due to the linearity of σ, there exists $i \in [n]$ such that $h(body(\sigma)) \in M_i$. But since $M_i \in models(\{\underline{a}_i\},\Sigma)$, $h(head(\sigma)) \in M_i$. The claim follows since $M_i \subseteq M$. □

Let us now conclude the proof of our lemma.
($\Rightarrow$) Assume that, for each $i \in [n]$, $\{\underline{a}_i\} \cup \Sigma \not\models q$, and thus there exists $M_i \in models(\underline{a}_i,\Sigma)$ such that $M_i \not\models q$. Therefore, $(M_1 \cup \ldots \cup M_n) \not\models q$, and by Claim 4.29 we get that $D \cup \Sigma \not\models q$.
($\Leftarrow$) By hypothesis, there exists $i \in [n]$ such that $\{\underline{a}_i\} \cup \Sigma \models q$. Consequently, for each $M \in models(\{\underline{a}_i\},\Sigma)$, $M \models \underline{a}$. Hence, by Claim 4.29, each instance of $models(D,\Sigma)$ entails q, and the claim follows. □

Although we can focus on a single database atom, in general we have to consider infinitely many models. The key idea underlying our approach is to use "representatives" of all these models in order to be able to encode them in a single structure, namely the proof-tree defined above. This can be achieved by considering the skolemized version of the given set of DTGDs. Given a set Σ of linear DTGDs, we define F_Σ as the set of *skolem functions* $\{f_\sigma \mid \sigma \in \Sigma\}$, where the arity of each f_σ is the number of universally quantified variables of σ. Let Σ_f denote the set of rules obtained from Σ by replacing each TGD σ of the form $p(\mathbf{X}) \rightarrow \exists Y\, r(\mathbf{X},Y)$ with the rule $\rho : p(\mathbf{X}) \rightarrow r(\mathbf{X}, f_\sigma(\mathbf{X}))$ (we will denote skolemized rules in Σ_f using the lower-case letter

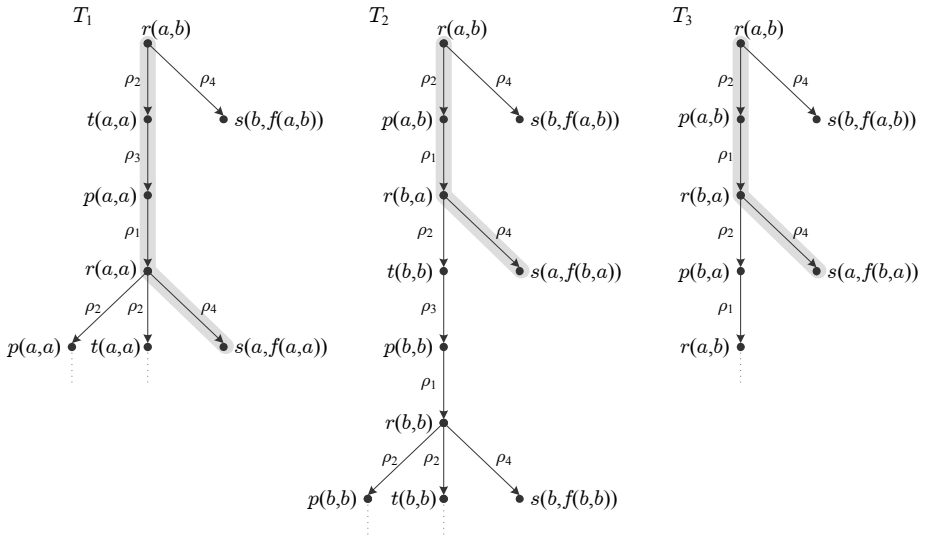


Figure 4.9: Possible trees of Example 4.31.

ρ. From the fact that skolemization preserves satisfiability (i.e. produces an equisatisfiable theory) we have that the following holds: given a database D, a set Σ of linear DTGDs in normal form, and a CQ_1 q, $D \cup \Sigma \models q$ iff $D \cup \Sigma_f \models q$. The set of *skolem terms* Γ_Σ is recursively defined as follows: each term of $\mathbf{C}$ belongs to Γ_Σ, and if $f_\sigma \in F_\Sigma$ has arity $n > 0$ and $t_1, \ldots, t_n$ are terms of Γ_Σ, then $f_\sigma(t_1, \ldots, t_n) \in \Gamma_\Sigma$. The notion of homomorphism naturally extends to atoms that contain functional terms of the form $f(t_1, \ldots, t_n)$, where each t_i is a variable of $\mathbf{V}$ or a skolem term; in particular, given a homomorphism h, $h(f(t_1, \ldots, t_n)) = f(h(t_1), \ldots, h(t_n))$. In what follows, it is more convenient to conceive models as trees. In fact, as we shall see, this will allows us to easily build the desired proof-tree from an atom w.r.t. a set of DTGDs.

Definition 4.30 (ρ-tree)**.** *Consider a set Σ of linear DTGDs in normal form, an atom $\underline{a}$, where $dom(a) \subset \Gamma_\Sigma$, and an instance $M \in models(\{\underline{a}\}, \Sigma_f)$. The* tree *of M, denoted $tree(M)$, is a labeled rooted tree $(N, E, \lambda_1, \lambda_2)$, where N is the node set, E is the edge set, $\lambda_1 : N \to M$ is a node-labeling function, and $\lambda_2 : E \to \Sigma_f$ is an edge-labeling function, such that:*

- *The root of $tree(M)$ is labeled by $\underline{a}$; and*
- *For each node $v \in N$, and rule $\rho \in \Sigma_f$ for which there exists a homomorphism h that maps $body(\rho)$ into $\lambda_1(v)$, the following holds: for each $\underline{b} \in head(\sigma)$, if $h(\underline{b}) \in M$, then there exists $u \in N$ with $\lambda_1(u) = h(\underline{b})$, $e = (v, u)$ belongs to E, and $\lambda_2(e) = \rho$.*

*The ρ-*tree *of M, denoted ρ-$tree(M)$, is the tree obtained from $tree(M)$ by keeping the root node v, each edge $e = (v, u)$ which is labeled by ρ, and the subtree rooted at u. Let $(\rho\text{-})trees(\underline{a}, \Sigma_f) = \{(\rho\text{-})tree(M) \mid M \in models(\{\underline{a}\}, \Sigma_f)\}$. By abuse of notation, given a $(\rho\text{-})$tree T and a CQ_1 q, we write $T \models q$ if $\{\lambda_1(v)\}_{v \in N} \models q$.*

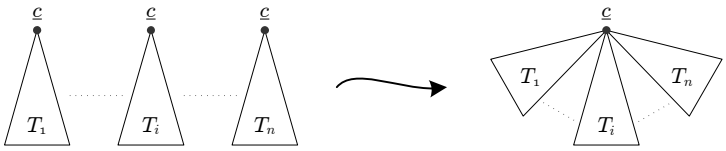


Figure 4.10: Rooted tree obtained from the disjoint union of $T_1, \ldots, T_n$ after merging the root nodes.

Example 4.31. *Consider the set Σ_f obtained by skolemizing Σ from Example 4.27, consisting of:*

$$\rho_1 : \; p(X,Y) \rightarrow r(Y,X) \qquad \rho_2 : \; r(X,Y) \rightarrow p(X,Y) \vee t(X,X)$$
$$\rho_3 : \; t(X,Y) \rightarrow p(X,Y) \qquad \rho_4 : \; r(X,Y) \rightarrow s(Y, f(X,Y))$$

Possible models of $\{r(a,b)\} \cup \Sigma_f$ are:

$$M_1 = \{r(a,b), s(b,f(a,b)), t(a,a), p(a,a), r(a,a), s(a,f(a,a))\}$$
$$M_2 = \{r(a,b), s(b,f(a,b)), p(a,b), r(b,a), t(b,b), p(b,b), r(b,b), s(a,f(b,a)), s(b,f(b,b))\}$$
$$M_3 = \{r(a,b), s(b,f(a,b)), p(a,b), r(b,a), p(b,a), s(a,f(b,a))\}$$

Notice that, for each other model M of $\{r(a,b)\} \cup \Sigma_f$, it holds that $M_i \subseteq M$, for at least one $i \in [3]$; the tree T_i of M_i is depicted in Figure 4.9. Observe that each T_i has exactly one ρ_2-tree and one ρ_4-tree. Moreover, the shaded paths form (modulo null renaming) the proof-tree from $r(a,b)$ w.r.t. Σ_f shown in Figure 4.8(a), which is valid w.r.t. $s(a,X)$.

Observe that the first edge of each shaded path in Figure 4.9 is labeled by the same rule. As shown in the next technical lemma, this is a general property that holds whenever the given database and set of DTGDs entail the given atomic query.

Lemma 4.32. *Consider a set Σ of linear DTGDs in normal form, an atom $\underline{a}$, where $dom(\underline{a}) \subset \Gamma_\Sigma$, and a CQ$_1$ q. It holds that, $\{a\} \cup \Sigma_f \models q$ iff there exists $\rho \in \Sigma_f$ such that $T \models q$, for each $T \in \rho\text{-}trees(\underline{a}, \Sigma_f)$.*

Proof. Let $\Sigma_f = \{\rho_1, \ldots, \rho_n\}$. By definition of the tree of a model, $trees(\underline{a}, \Sigma_f)$ is isomorphic to the set

$$\mathcal{T} = \left\{ \biguplus_{i \in [n]} T_i \;\middle|\; (T_1, \ldots, T_n) \in \rho_1\text{-}trees(\underline{a}, \Sigma_f) \times \ldots \times \rho_n\text{-}trees(\underline{a}, \Sigma_f) \right\},$$

where $\uplus_{i \in [n]} T_i$ is the rooted tree obtained from the disjoint union of $T_1, \ldots, T_n$ after merging the root nodes (see Figure 4.10). We are now ready to show our lemma.

($\Rightarrow$) Assume that for each $\rho \in \Sigma_f$ there exists $T \in \rho\text{-}trees(\underline{a}, \Sigma_f)$ such that $T \not\models q$. Therefore, there exists $T' \in \mathcal{T}$ such that $T' \not\models q$. Clearly, there exists $T'' \in trees(\underline{a}, \Sigma_f)$ isomorphic to T'. Assuming that $T'' = tree(M)$, for some $M \in models(\{\underline{a}\}, \Sigma_f)$, $M \not\models \underline{a}$; thus, $\{\underline{a}\} \cup \Sigma_f \not\models \underline{a}$.

($\Leftarrow$) This direction follows immediately. □

Given an atom $\underline{a}$, where $dom(a) \subset \mathbf{C}$, a set Σ of linear DTGDs, and a CQ$_1$ q, by applying recursively the property guaranteed by Lemma 4.32, one can build a proof-tree from $\underline{a}$ w.r.t. Σ which is valid w.r.t. q. This is exactly the key idea underlying the proof of the next crucial result.

Lemma 4.33. *Consider a database D, a set Σ of linear DTGDs in normal form, and a CQ_1 q. It holds that, $D \cup \Sigma \models q$ if and only if there exists a proof-tree from $\underline{a}$ w.r.t. Σ, for some $\underline{a} \in D$, which is valid w.r.t. q.*

Proof. ($\Rightarrow$) The claim is shown by giving a proof-tree from some $\underline{a} \in D$ w.r.t. Σ which is valid w.r.t. q; assume that $q = \exists \mathbf{X} p(\mathbf{X})$. By Lemmas 4.28 and 4.32, there exist $\underline{a} \in D$ and $\rho \in \Sigma_f$ such that in every $T \in \rho\text{-}trees(\underline{a}, \Sigma_f)$ there exists a path π_T from the root to a node u which is labeled by $h(p(\mathbf{X}))$, where h is a homomorphism, and the first edge e_T of π_T is labeled by a rule ρ of the form $\underline{b} \rightarrow \underline{b}_1 \vee \underline{b}_2$ or $\underline{b} \rightarrow \underline{b}_1$. Let us now construct a rooted binary tree $T' = (N, E, \lambda_1, \lambda_2)$; initially, $N = \{v\}$, $E = \emptyset$, $\lambda_1 = \{v \rightarrow \underline{a}\}$, and $\lambda_2 = \emptyset$. Clearly, $\rho\text{-}trees(\underline{a}, \Sigma_f)$ can be partitioned into S_1 and S_2 such that, for each $T \in S_1$, if $e_T = (u, w)$ and h maps $\underline{b}$ into $\underline{a}$ (which is the label of u), then w is labeled by $h(\underline{b}_1)$. For each $i \in \{1, 2\}$, assume that the second node of π_T, for each $T \in S_i$, is labeled by $\underline{g}_i$. Let $N = N \cup \{w_1, w_2\}$, $E = E \cup \{(v, w_i)\}_{i \in \{1,2\}}$, $\lambda_1 = \lambda_1 \cup \{w_i \rightarrow \underline{g}_i\}_{i \in \{1,2\}}$, and $\lambda_2 = \lambda_2 \cup \{(v, w_i) \rightarrow \rho\}_{i \in \{1,2\}}$. Due to the fact that $\{\underline{g}_i\} \cup \Sigma_f \models q$, for each $i \in [2]$, Lemma 4.32 can be recursively applied finitely many times as described above, and eventually T' is constructed. The desired proof-tree is obtained from T' by replacing the skolem terms occurring in T' with distinct nulls of $\mathbf{N}$.

($\Leftarrow$) By hypothesis, there exists $\underline{a} \in D$ and a proof-tree T from $\underline{a}$ w.r.t. Σ which is valid w.r.t. q. It is possible to construct from T a rooted binary tree T' such that the nodes are labeled by atoms with skolem terms (instead of nulls), the edges are labeled by rules of Σ_f, and the structural properties of T are preserved. Assume that the label of each outgoing edge of the root of T' is ρ. By Lemmas 4.28 and 4.32, it suffices to show that each ρ-tree from $\underline{a}$ w.r.t. Σ_f entails q. This follows from the fact that, for each $T'' \in \rho\text{-}trees(\underline{a}, \Sigma_f)$, there is a path from the root to a leaf of T' that can be mapped into T''. □

By Lemma 4.33 we get that CQ_1 answering is equivalent to the problem of deciding whether a proof-tree exists. The latter can be tackled by exploiting an alternating algorithm.

The Alternating Algorithm

The alternating algorithm called SearchPT accepts as input a database D, a set Σ of linear DTGDs, and a CQ_1 q, and decides whether a proof-tree from $\underline{a}$ w.r.t. $\mathsf{N}(\Sigma)$, for some $\underline{a} \in D$, which is valid w.r.t. q exists (recall that $\mathsf{N}(\Sigma)$ denotes the normal form of Σ as defined in Chapter 2). The formal definition of SearchPT is depicted in Figure 4.11, while an example of its computation follows.

Example 4.34. *Consider the database $D = \{r(a, b)\}$, the set Σ of linear DTGDs*

$$\sigma_1 : \; r(X,Y) \rightarrow p(X) \vee t(Y,X) \qquad \sigma_2 : \; p(X) \rightarrow r(X,X)$$
$$\sigma_3 : \; t(X,Y) \rightarrow \exists Z \, s(Y,Z) \qquad \sigma_4 : \; s(X,Y) \rightarrow \exists Z \, s(Y,Z),$$

and the CQ_1 $q = \exists \mathbf{X} \, p(\mathbf{X})$. Figure 4.12 shows an initial part of the alternating computation of $\mathsf{SearchPT}(D, \Sigma, q)$*. Observe that in the shaded edge exactly three (i.e. maximum arity plus one) nulls appear.*

Algorithm SearchPT(D, Σ, q)

Input: A database D, a set Σ of linear DTGDs, and a CQ_1 $q = \exists \mathbf{X}\, p(\mathbf{X})$.
Output: *Accept* iff there exists a "small" proof-tree from $\underline{a}$ w.r.t. $\mathsf{N}(\Sigma)$, for some $\underline{a} \in D$, which is valid w.r.t. q.

1. $\Sigma := \mathsf{N}(\Sigma)$;
2. $N := \{z_i \in \mathbf{N}\}_{i \in [m+1]}$, where m is the maximum arity over all predicates of $sch(\Sigma)$;
3. Guess an atom $\underline{a} \in D$;
4. Guess to either execute step 5 or to skip to step 6;
5. If there exists a homomorphism h such that $h(p(\mathbf{X})) = \underline{a}$, then *accept*;
6. Guess a DTGD $\sigma \in \Sigma$ of the form $r(\mathbf{X}) \rightarrow \exists Y\, \psi_i(\mathbf{X}, Y)$ (Y may not be present), and a homomorphism h such that $h(body(\sigma)) = \underline{a}$; if there is no such σ and h, then *reject*;
7. Universally choose each disjunct $\underline{b}$ of $head(\sigma)$ and do the following:
   a) If Y is present in σ, then $h' := h_{|\mathbf{x}} \cup \{Y \rightarrow t \mid t \in N \text{ and } t \notin dom(h(r(\mathbf{X})))\}$; otherwise, $h' := h$;
   b) $\underline{a} := h'(\underline{b})$ and goto step 4.

Figure 4.11: The alternating algorithm SearchPT.

Correctness of SearchPT follows by construction. Thus, Lemma 4.33 implies the following:

Proposition 4.35. *Consider a database D, a set Σ of linear DTGDs, and a CQ_1 q. Then, $D \cup \Sigma \models q$ iff* SearchPT(D, Σ, q) *accepts.*

Complexity Upper Bounds

Equipped with the above machinery, we are now ready to establish the desired complexity upper bounds for our problem.

Theorem 4.36. CQ_1-Answering *under linear DTGDs is in* ExpTime *in the combined complexity.*

Proof. Consider a database D, a set Σ of linear DTGDs, and a CQ_1 q. Recall that alternating PSpace coincides with ExpTime. Thus, by Proposition 4.35, and since SearchPT is an alternating procedure, it suffices to show that SearchPT(D, Σ, q) uses polynomial space in general. Clearly, the set of nulls N can be maintained in $O(m \log m)$ space, where m is the maximum arity over all predicates of $sch(\mathsf{N}(\Sigma))$. Furthermore, the atom $h'(\underline{b})$ can be maintained in $O(m \log m + m \log n)$ space, where $n = |dom(D)|$. This is because $h'(\underline{b})$ contains at most m terms, and each such term can be represented using $\log m$ bits, if its a null of N, or $\log n$ bits, if its a constant of $dom(D)$. Therefore, at each step of its computation the algorithm SearchPT uses $O(m \log m + m \log n)$ space. □

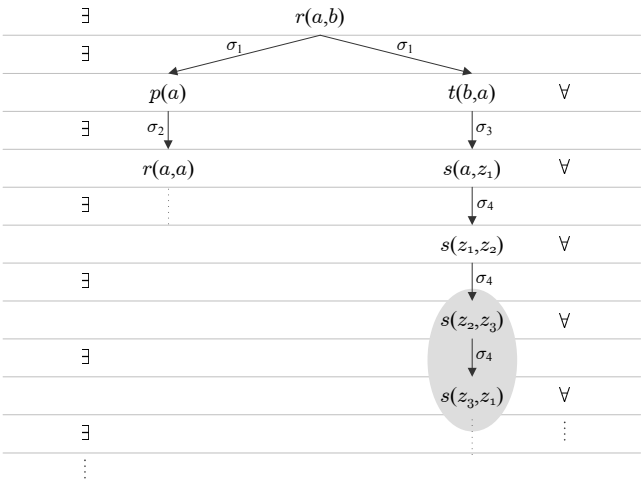


Figure 4.12: Computation of the algorithm SearchPT.

We proceed to show that CQ_1 answering under linear DTGDs is in PTime in case of predicates of bounded arity. Since alternating LogSpace coincides with PTime, it suffices to show that SearchPT(D, Σ, q) uses only logarithmic space. Recall that our algorithm uses $O(m \log m + m \log n)$ space, where m is the maximum arity over all predicates of $sch(\mathsf{N}(\Sigma))$. However, the existing normalization procedure (see Chapter 2) introduces new auxiliary predicates of unbounded arity, and thus m is not an integer constant. Thus, we need a normalization procedure which does not increase the arity of the original schema. Although it is not clear how such a normalization procedure works in general, for CQ_1 answering under linear DTGDs such a procedure can easily be defined. The key idea is that we do not need to preserve the joins among nulls since linear DTGDs can have only one body-atom, and also the query to be answered is a single atom. In order to exploit this observation, we will redefine the function $\mathsf{N}(\cdot)$ that computes the normal form of a set of DTGDs. Let us explain how this adapted normalization procedure works.

Consider a linear DTGD σ. First, we construct the set $\mathsf{N}_1(\sigma)$ by exhaustively applying the following two replacement rules, starting from σ, until each assertion is either a single-atom-head TGD, or a DTGD with a disjunction of two atoms in its head:

1. A linear DTGD $p(\mathbf{X}) \rightarrow \vee_{i=1}^{n} \exists \mathbf{Y}_i \, (p_1^i(\mathbf{X}, \mathbf{Y}_i), \ldots, p_{k_i}^i(\mathbf{X}, \mathbf{Y}_i))$ is replaced by

$$
\begin{aligned}
p(\mathbf{X}) &\rightarrow p_1^\star(\mathbf{X}) \vee p_{2..n}^\star(\mathbf{X}) \\
p_1^\star(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_1 \, p_1^1(\mathbf{X}, \mathbf{Y}_1) \\
&\vdots \\
p_1^\star(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_1 \, p_{k_1}^1(\mathbf{X}, \mathbf{Y}_1) \\
p_{2..n}^\star(\mathbf{X}) &\rightarrow \bigvee_{i=2}^{n} \exists \mathbf{Y}_i \, (p_1^i(\mathbf{X}, \mathbf{Y}_i), \ldots, p_{k_i}^i(\mathbf{X}, \mathbf{Y}_i)),
\end{aligned}
$$

where $p_1^\star$ and $p_{2..n}^\star$ are $|\mathbf{X}|$-ary auxiliary predicates not introduced so far.

2. A linear TGD $p(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, p_1(\mathbf{X}, \mathbf{Y}_1), \ldots, p_n(\mathbf{X}, \mathbf{Y}_n)$ is replaced by

$$\begin{aligned} p(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_1\, p_1(\mathbf{X}, \mathbf{Y}_1) \\ &\vdots \\ p(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_n\, p_n(\mathbf{X}, \mathbf{Y}_n). \end{aligned}$$

Now, from $\mathsf{N}_1(\sigma)$, we obtain the redefined $\mathsf{N}(\sigma)$ by employing the construction presented in Chapter 2 for transforming a set of DTGDs into a set of DTGDs with only one occurrence of an existentially quantified variable. Finally, given a set Σ of DTGDs, $\mathsf{N}(\Sigma)$ is defined as $\bigcup_{\sigma \in \Sigma} \mathsf{N}(\sigma)$. We will henceforth use this redefined normal form whenever we deal with linear DTGDs and atomic queries. The next lemma follows by construction:

Lemma 4.37. *Consider a set Σ of linear DTGDs. Then,*

1. $\mathsf{N}(\Sigma)$ *is in normal form;*
2. $\mathsf{N}(\Sigma)$ *is constructible in logarithmic space;*
3. $\max_{p \in sch(\mathsf{N}(\Sigma))}\{arity(p)\} = \max_{p \in sch(\Sigma)}\{arity(p)\}$*; and*
4. *for every database D and CQ_1 q, $D \cup \Sigma \models q$ if and only if $D \cup \mathsf{N}(\Sigma) \models q$.*

With the above normalization procedure in place, Lemma 4.37 implies the following:

Theorem 4.38. CQ_1-Answering *under linear DTGDs is in PTime in case where the predicate arity is bounded.*

The rest of this subsection is devoted to show that CQ_1 answering under linear DTGDs is in AC_0 in data complexity. We do this by establishing that our problem is first-order rewritable, that is, it can be reduced to the problem of evaluating a first-order query over a database. First-order rewritability was first introduced in the context of description logics by Calvanese et al. [2007].

Consider a set Σ of linear DTGDs, and a CQ_1 q. Let C be the constants occurring in q, and $N = \{z_1, \ldots, z_m\}$ be a set of nulls, where m is the maximum arity over all predicates of $sch(\Sigma)$. Let $base(q, \Sigma)$ be the set of all atoms that can be formed using terms of $C \cup N$ and predicates of $sch(\Sigma)$. Let $B = \{\underline{b} \mid \underline{b} \in base(q, \Sigma) \text{ and } \{\underline{b}\} \cup \Sigma \models q\}$, and μ be a renaming substitution that maps each $z \in N$ into a distinct variable $X_z \in \mathbf{V}$. We define the first-order query q_Σ as

$$\bigvee_{\underline{b} \in B} \exists X_{z_1} \ldots \exists X_{z_m}\, \mu(\underline{b}).$$

In fact, q_Σ is a union of CQs, where each disjunct is an atomic CQ. It is easy to see that $|B| \leqslant |sch(\Sigma)| \cdot (2m)^m$. Since, by Theorem 4.36, CQ_1 answering under linear DTGDs is feasible in exponential time, we conclude that the set B, and thus the query q_Σ, can be constructed in exponential time. In what follows, we show that q_Σ is a sound and complete rewriting:

Lemma 4.39. *Consider a set Σ of linear DTGDs in normal form, and a CQ_1 q. Then, for every database D, $D \cup \Sigma \models q$ iff $D \models q_\Sigma$.*

Proof. Fix a database D and let $q = \exists \mathbf{X}\, p(\mathbf{X})$. By construction of q_Σ, it suffices to show that $D \cup \Sigma \models q$ if and only if there exists $\underline{b} \in base(q, \Sigma)$ and a homomorphism h such that $h(\underline{b}) \in D$ and $\{\underline{b}\} \cup \Sigma \models q$.
($\Rightarrow$) By Lemma 4.28, there exists an atom $\underline{a} \in D$ such that $\{\underline{a}\} \cup \Sigma \models q$. Therefore, for each $M \in models(\{\underline{a}\}, \Sigma)$, there exists a homomorphism h_M such that $h_M(p(\mathbf{X})) \subseteq M$. Moreover, by construction, there exists $\underline{b} \in base(q, \Sigma)$ and a bijective homomorphism λ such that $\lambda(\underline{b}) = \underline{a}$; clearly, $\lambda(\underline{b}) \in D$. Let $\gamma : dom(\underline{a}) \to dom(\underline{b})$ be the substitution $\{t \to t' \mid t' \to t \in \lambda_{|_{dom(\underline{a})}}\}$. By induction on the depth of the trees of the models, it can be shown that for each $M' \in models(\{\underline{b}\}, \Sigma)$, there exists $M \in models(\{\underline{a}\}, \Sigma)$ such that $\gamma(M) = M'$; thus, the homomorphism $\gamma \circ h_M$ maps $p(\mathbf{X})$ into M'.
($\Leftarrow$) This direction can be shown by providing a similar argument. □

In the case of data complexity, the set of DTGDs Σ and the CQ_1 q are fixed, and thus q_Σ is fixed. Since, as shown by Vardi [1995], the evaluation of first-order queries is feasible in AC_0 when the query is fixed, Lemma 4.39 implies the following complexity result:

Theorem 4.40. CQ_1-Answering *under linear DTGDs is in AC_0 in the data complexity.*

This concludes the upper bound section for atomic queries.

4.8.3 Lower Bounds

We proceed now to establish the desired complexity lower bounds for CQ_1 answering under DIDs. The following theorem is obtained by simulating an alternating polynomial space Turing machine, using the ideas of the PSpace-hardness proof of the implication problem of (non-disjunctive) IDs given in [Casanova et al., 1984].

Theorem 4.41. CQ_1-Answering *under DIDs is ExpTime-hard in the combined complexity.*

Proof. The proof is by reduction from the acceptance problem of an alternating linear space Turing machine M on input $I = a_1 \ldots a_{|I|}$. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ being the blank symbol, $\delta : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that M is well-behaved and never tries to read beyond its tape boundaries, always halts, and uses exactly $n = |I|$ tape cells. Also, for each transition $(s, a) \to ((s', a', d'), (s'', a'', d''))$, we assume that $d' = d''$; the latter is a technical assumption which will allow us to represent the transitions of δ in a more convenient form (in the same way as in the PSpace-hardness proof of the implication problem of (non-disjunctive) IDs given in [Casanova et al., 1984]). We represent configurations using a string $\Lambda^\star S \Lambda^+$, that is, the state of the configuration is placed to

the immediate left of the cursor position. In this notation, the initial configuration is $s_0 a_1 \dots a_{|I|}$. Finally, we assume that the accepting configuration of M is $s_a \sqcup^n$.

Our goal is to construct a database D, a set Σ of DIDs, and a CQ_1 q such that $D \cup \Sigma \models q$ if and only if M accepts. We employ the predicate $conf$ of arity $ar = (|S| + 3) \cdot (n + 1)$ to represent objects in the relation $(S \cup \Lambda) \times [n + 1]$. Such an object (x, i) represents the fact that the i-th symbol on the tape of M is x (i.e. the integer part represents the tape cell, and the $(S \cup \Lambda)$ part represents the tape content and the head position). Assume an order on the elements of $(S \cup \Lambda) \times [n + 1]$, and let $\#_{(x,i)}$ denote the position of the element (x, i) in this order. The position $conf[\#_{(x,i)}]$ will contain the constant 1 (resp. an element from $(\{0\} \cup \mathbf{N})$) if (x, i) is true (resp. false). Intuitively, we want that the first $conf$-atom, contained in the database D, to represent the starting configuration of M, by having the constant 1 appear at each relevant position (x, i), and 0 everywhere else. We will then use the set of DIDs Σ to propagate the constant 1 to its relevant new positions in subsequent configurations. We proceed with the construction of D, Σ and q.

The Database D. D consists of a single atom which represents the initial configuration of M: $s_0 a_1 \dots s_{|I|}$. Given two elements $x, y \in \Lambda$, $f(x, y) = 1$ if $x = y$; otherwise, $f(x, y) = 0$. The database atom is defined as $conf(\mathbf{t})$, where $\mathbf{t} \in \{0, 1\}^{ar}$ is the tuple:

$$1, \overbrace{0, \dots, 0}^{|S|-1}, 0, 0, 0, \overbrace{0, \dots, 0}^{|S|}, f(a_1, 0), f(a_1, 1), f(a_1, \sqcup), \dots,$$
$$\underbrace{0, \dots, 0}_{|S|}, f(a_{|I|}, 0), f(a_{|I|}, 1), f(a_{|I|}, \sqcup).$$

The construction of D is now completed.

The Set of DIDs Σ. With Σ we encode the transitions of δ. These transitions can be described by expressions of the form $x_1 y_1 z_1 \rightarrow x_2 y_2 z_2; x_3 y_3 z_3$ with $x_i, y_i, z_i \in (S \cup \Lambda)$, for each $i \in \{1, 2, 3\}$. For example, the transition $(s, a) \rightarrow ((s', a', -1), (s'', a'', -1))$ corresponds to $xsa \rightarrow s'xa'; s''xa''$, for each $x \in \Lambda$, while the transition $(s, a) \rightarrow ((s', a', +1), (s'', a'', +1))$ corresponds to $sax \rightarrow a's'x; a''s''x$, for each $x \in \Lambda$. Let T_δ be all such expressions corresponding to transitions of δ.

Consider an expression $\tau \in T_\delta$ of the form $x_1 y_1 z_1 \rightarrow x_2 y_2 z_2; x_3 y_3 z_3$. For each $i \in \{1, 2, 3\}$ and $j \in [n - 1]$, we define the atom $g_i^j(\tau)$ as $conf(\mathbf{t})$, where $\mathbf{t} \in \mathbf{V}^{ar}$ is as follows:

- At positions $\#_{(x_i,j)}$, $\#_{(y_i,j+1)}$ and $\#_{(z_i,j+2)}$ of $\mathbf{t}$ the variables X, Y and Z occur, respectively;
- For each $(x, \ell) \in (\Lambda \times \{1, \dots, j - 1, j + 3, \dots, n + 1\})$, at position $\#_{(x,\ell)}$ of $\mathbf{t}$ the variable X_ℓ occurs; and
- At each position of $\mathbf{t}$ not considered above a distinct variable occurs; these variables form the set $\mathbf{Y}_i$.

Having the above atoms in place, we are now ready to define the set Σ of DIDs. For each $\tau = x_1y_1z_1 \rightarrow x_1y_2z_2; x_3y_3z_3$ of T_δ, where $s \in S$ is the state occurring in $x_1y_1z_1$, and for each $j \in [n-1]$:

- If $s \in S_\forall$, then Σ includes the DID

$$g_1^j(\tau) \rightarrow \exists \mathbf{Y}_2\, g_2^j(\tau) \vee \exists \mathbf{Y}_3\, g_3^j(\tau).$$

- If $s \in S_\exists$, then Σ includes the two (non-disjunctive) IDs

$$g_1^j(\tau) \rightarrow \exists \mathbf{Y}_2\, g_2^j(\tau) \qquad \text{and} \qquad g_1^j(\tau) \rightarrow \exists \mathbf{Y}_3\, g_3^j(\tau).$$

This completes the construction of Σ.

The atomic query q. The query q is defined as $\exists \mathbf{X}\, conf(\mathbf{t})$, where $\mathbf{t} \in (\{1\} \cup \mathbf{X})^{ar}$ is as follows:

- At positions $\#_{(s_a,1)}, \#_{(\sqcup,2)}, \ldots, \#_{(\sqcup,n+1)}$ the constant 1 occurs; and
- At all the other positions a distinct variable of $\mathbf{X}$ occurs.

The construction of q is now completed.

Correctness. By construction, whenever a transition $\tau \in \delta$ would take place, at least one of the DIDs of Σ will propagate the constant 1 in such a way that the derived atom will represent the successor configuration. If the machine thus reaches the accepting configuration, this implies that there exists a sequence of DIDs that will derive the atom corresponding to that accepting configuration, namely the CQ_1 q. Conversely, if $D \cup \Sigma \models q$, then there exists a sequence of DIDs as above, and the sequence of corresponding transitions in δ yields an accepting computation of M. □

We now focus on multi-linear DTGDs, and show that query answering remains 2ExpTime-hard, even if we consider atomic queries.

Theorem 4.42. CQ_1-Answering *under multi-linear DTGDs is 2ExpTime-hard in the combined complexity.*

Proof. Let D, Σ and Q be the database, the set of DIDs, and the UCQ employed in the proof of Theorem 4.23, where it is shown that ACQ answering under DIDs is 2ExpTime-hard. Our goal is to rewrite Σ and Q into Σ' and Q', respectively, such that: (1) Σ' is a set of DIDs; (2) for each disjunct ϕ of Q', and for each pair of atoms $\underline{a}, \underline{b}$ of ϕ, $var(\underline{a}) = var(\underline{b})$; and (3) $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models Q'$. Having Σ' and Q' in place our claim follows. More precisely, the set Σ'' of multi-linear DTGDs can be constructed by adding to Σ' the multi-linear (non-disjunctive) DTGD $\phi \rightarrow p^\star$, where $p^\star \notin sch(\Sigma')$, for each disjunct ϕ of Q', such that $D \cup \Sigma' \models Q'$ iff $D \cup \Sigma'' \models q$, where $q = p^\star$.

Before defining Σ' and Q', let us explain why we cannot directly add the disjuncts of Q to Σ and get the desired result. Consider, for example, the disjunct $Q_{inertia}$

$$\bigvee_{(s,s')\in S\times S} \bigvee_{(a,a')\in\Lambda\times\Lambda, a\neq a'} \bigvee_{i\in\{1,2\}} \bigvee_{j\in\{0,1\}} \exists \mathbf{X}^n \exists T \exists P \exists N_1 \exists N_2 \exists N_1' \exists N_2' \\ \big(conf[s](\mathbf{X}^n, a, 0, T, P, N_1, N_2) \wedge conf[s'](\mathbf{X}^n, a', j, N_i, T, N_1', N_2')\big),$$

which checks that the tape cells not under the cursor keep their old values during a transition. Clearly, to perform such a check we just need to have access to a certain configuration and to one of its subsequent configurations. However, in an atom $conf[s](\cdot)$ we store more information than necessary, namely the previous configuration, and not just one but both subsequent configurations. This is precisely the reason why the atoms of $Q_{inertia}$ contain different variables, and its disjuncts will give rise to TGDs which are not multi-linear. Thus, we need to project out the useless variables via some (non-disjunctive) IDs, and rewrite the disjuncts of Q analogously. The projection IDs, which are the ones that we add in Σ in order to get Σ', are as follow:

$$\begin{aligned} conf[s](\mathbf{X}^n, C, H, T, P, N_1, N_2) &\rightarrow conf_P[s](\mathbf{X}^n, C, H, T, P) \\ conf[s](\mathbf{X}^n, C, H, T, P, N_1, N_2) &\rightarrow conf_L[s](\mathbf{X}^n, C, H, T, N_1) \\ conf[s](\mathbf{X}^n, C, H, T, P, N_1, N_2) &\rightarrow conf_R[s](\mathbf{X}^n, C, H, T, N_2), \end{aligned}$$

where the subscripts P, L and R stand for previous, (next-)left and (next-)right, respectively. Now, the query $Q_{inertia}$ can be rewritten as follows:

$$\bigvee_{x\in\{L,R\}} \bigvee_{(s,s')\in S\times S} \bigvee_{(a,a')\in\Lambda\times\Lambda, a\neq a'} \bigvee_{j\in\{0,1\}} \exists \mathbf{X}^n \exists T \exists N \\ \big(conf_x[s](\mathbf{X}^n, a, 0, T, N) \wedge conf_P[s'](\mathbf{X}^n, a', j, N, T)\big).$$

The same approach can be applied also for Q_{trans} and $Q_{inertia}$, without violating the multi-linearity of the added TGDs. However, this is not true for Q_{move}. After rewriting Q_{move} as above, we get

$$\bigvee_{x\in\{L,R\}} \bigvee_{0<i,j\leqslant n} \bigvee_{(s,s')\in S\times S} \bigvee_{(c_1,c_2,c_3)\in\Lambda^3} \exists \mathbf{X}^{n-i} \exists \mathbf{Y}^{n-j} \exists T \exists N \\ \Bigg(\underbrace{conf_x[s](\mathbf{X}^{n-i}, 0, \mathbf{1}^{i-1}, c_1, 1, T, N) \wedge conf_P[s'](\mathbf{X}^{n-i}, 1, \mathbf{0}^{i-1}, c_2, 0, N, T)}_{\varphi_1} \wedge \\ \underbrace{conf_x[s](\mathbf{Y}^{n-j}, 1, \mathbf{0}^{j-1}, c_1, 1, T, N) \wedge conf_P[s'](\mathbf{Y}^{n-j}, 0, \mathbf{1}^{i-1}, c_3, 0, N, T)}_{\varphi_2} \Bigg),$$

and the TGD $\varphi_1, \varphi_2 \rightarrow p^\star$ is not multi-linear due to the variables $\mathbf{X}^{n-i}$ and $\mathbf{Y}^{n-j}$. Nevertheless, since we just need to join the variables T and N, it suffices to add to Σ the multi-linear TGDs $\varphi_1 \rightarrow aux_1(T, N)$, $\varphi_2 \rightarrow aux_2(T, N)$ and $aux_1(T, N), aux_2(T, N) \rightarrow p^\star$, where aux_1 and aux_2 are auxiliary predicates not occurring in $sch(\Sigma)$. □

Although atomic query answering under DIDs is hard for ExpTime, in case of predicates of bounded arity we can only show that is PTime-hard; in fact, this is true even if we focus on unary predicates.

Theorem 4.43. CQ_1-Answering *under DIDs is PTime-hard, even for unary predicates.*

Proof. The proof is by reduction from the acceptance problem of an alternating logarithmic space Turing machine M on input $I = a_1 \dots a_{|I|}$. Recall that logarithmic space Turing machines are equipped with a read-only input tape and a read/write work tape, where on the input tape only the cells which hold the input can be visited, while on the work tape only logarithmically many cells can be used. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ being the blank symbol, $\delta : S \times \Lambda \times \Lambda \to (S \times \Lambda \times \{-1, +1\} \times \Lambda \times \{-1, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that M is well-behaved and never tries to read beyond its tape boundaries, and uses exactly $n = \log |I|^k$ cells of the work tape, where $k > 0$.

We proceed to construct a database D, a set Σ of DIDs, where $sch(\Sigma)$ consists of unary predicates, and a CQ_1 q such that $D \cup \Sigma \models q$ if and only if M accepts. Each configuration of M is represented using a unary predicate of the form

$$conf[s\#a_1 \dots a_{|I|} x_1 \dots x_{|I|} \# c_1 \dots c_n y_1 \dots y_n],$$

where $s \in S$, $a_1 \dots a_{|I|}$ is the input tape, $x_1 \dots x_{|I|} \in \{0, 1\}^{|I|}$ indicates the position of the cursor on the input tape (e.g. $x_1 \dots x_{i-1} x_{i+1} \dots x_{|I|} = 0^{|I|-1}$ and $x_i = 1$ implies that the cursor is at the i-th cell), $c_1 \dots c_n \in \Lambda^n$ is the work tape, and $y_1 \dots y_n \in \{0, 1\}^n$ indicates the position of the cursor on the work tape. Since the number of configurations of M on I is polynomial, we only need polynomially many predicates. We proceed with the construction of D, Σ and q.

The Database D. Let D be the database consisting of the single atom

$$conf[s_0\#a_1 \dots a_{|I|} 1 \overbrace{0 \dots 0}^{|I|-1} \# \underbrace{\sqcup \dots \sqcup}_{n} 1 \underbrace{1 \dots 1}_{n}](c),$$

where c is an arbitrary constant of $\mathbf{C}$.

The Set Σ. With Σ we encode the transitions of δ. Let $f : \{0, 1\} \to 2^{[|I|]}$ be the function such that $f(x)$, where $x \in \{0, 1\}$, is the set of cells of the input tape which contain the symbol x. For each transition $(s, a, b) \to ((s_1, a, +1, b_1, -1), (s_2, a, -1, b_2, +1))$ of δ, we add in Σ the following DIDs: for each $i \in f(a), j \in [n-1]$ and $x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_n \in \Lambda^{n-1}$, $p_1(X) \to p_2(X) \vee p_3(X)$,

if $s \in S_\forall$; else, $p_1(X) \rightarrow p_2(X)$ and $p_1(X) \rightarrow p_3(X)$, where

$$p_1 = conf[s\#a_1 \dots a_{i-1}aa_{i+1} \dots a_{|I|} \underbrace{0 \dots 0}_{i-1} 1 \underbrace{0 \dots 0}_{|I|-i} \#x_1 \dots x_{j-1}a_1x_{j+1} \dots x_n \underbrace{0 \dots 0}_{j-1} 1 \underbrace{0 \dots 0}_{n-j}]$$

$$p_2 = conf[s_1\#a_1 \dots a_{i-1}aa_{i+1} \dots a_{|I|} \underbrace{0 \dots 0}_{i} 1 \underbrace{0 \dots 0}_{|I|-i-1} \#x_1 \dots x_{j-1}b_1x_{j+1} \dots x_n \underbrace{0 \dots 0}_{j-2} 1 \underbrace{0 \dots 0}_{n-j+1}]$$

$$p_3 = conf[s_2\#a_1 \dots a_{i-1}aa_{i+1} \dots a_{|I|} \underbrace{0 \dots 0}_{i-2} 1 \underbrace{0 \dots 0}_{|I|-i+1} \#x_1 \dots x_{j-1}b_2x_{j+1} \dots x_n \underbrace{0 \dots 0}_{j} 1 \underbrace{0 \dots 0}_{n-j-1}].$$

Similar DIDs are added to Σ for all the other forms of transitions occurring in δ. Finally, for each predicate of the form $conf[s_a \dots]$ introduced above, which represents an accepting configuration, we add the (non-disjunctive) ID

$$conf[s_a \dots](X) \rightarrow accept(X).$$

This completes the construction of Σ.

The atomic query q. With q we just need to ask wether an accepting configuration has been reached, that is, $\exists X\, accept(X)$, where $X \in \mathbf{V}$.

Correctness. It is easy to see, by construction, that $D \cup \Sigma \models q$ if and only f M accepts. It remains to show that the total number of DIDs that we need to construct is polynomial w.r.t. $|I|$. Recall that $n = \log |I|^k$, where $k > 0$. Clearly,

$$|\Sigma| \leqslant |I| \cdot (n-1) \cdot 3^n \cdot 2 = |I| \cdot (\log |I|^k - 1) \cdot 3^{\log |I|^k} \cdot 2 \leqslant |I| \cdot (\log |I|^k - 1) \cdot |I|^k \cdot 2,$$

and the claim follows. □

The last lower bound that we establish for CQ_1 answering is the ExpTime-hardness when we focus on multi-linear DTGDs and predicates of bounded arity. This result can be obtained by adapting the proof of Theorem 4.42. Since we have to simulate an alternating polynomial space (and not exponential space) Turing machine, the polynomially many cells (assume that the machine uses n cells) can be encoded in the predicates, and thus the arity becomes fixed. More precisely, we use the predicates

$$conf[s\# \underbrace{0 \dots 0}_{i-1} 1 \underbrace{0 \dots 0}_{n-i}] \quad \text{and} \quad conf_x[s\# \underbrace{0 \dots 0}_{i-1} 1 \underbrace{0 \dots 0}_{n-i}],$$

where $x \in \{P, L, R\}$, which correspond to the i-the cell of the encoded configuration. Now, it is straightforward to see how the set of multi-linear DTGDs given in the proof of Theorem 4.42 can be adapted, and the next result follows:

Theorem 4.44. CQ_1-Answering *under multi-linear DTGDs is ExpTime-hard, even for predicates of bounded arity.*

4.9 Summary

In this chapter, we have thoroughly investigated the complexity of the query answering problem under different flavors of guarded-based sets of DTGDs and query languages. It turned out that the extension of the guarded-based classes of rules with disjunction was fairly straightforward, which led us to start with a preliminary investigation into the query answering problem under the discussed guarded-based classes of DTGDs. We first showed that in most cases, UCQs do not add any power to the query language when compared to CQs, which simplified our subsequent complexity analysis. Given this information, we then proceeded to give a thorough complexity evaluation for the discussed guarded-based classes of DTGDS (namely, DIDs, linear, multi-linear, guarded, frontier-guarded, weakly-guarded, and weakly-frontier-guarded sets of DTGDs), where we treated each of the relevant query languages in turn.

For arbitrary CQs, we showed a strong 2ExpTime lower bound that even holds for query answering under fixed sets of DIDs, the weakest formalism we investigated. Together with straightforward data complexity results that range from co-NP-completeness for non-weak classes to ExpTime-completeness for the weak classes of DTGDs, this result, in addition to a 2ExpTime upper bound for the most expressive fragment, completed the entire complexity picture. In general, we can thus observe that for expressive guarded-based classes of rules, where query answering is already 2ExpTime-complete for TGDs, disjunction comes essentially for free, in terms of complexity. For less expressive formalisms, the cost of allowing disjunction increases dramatically; for example in the case of DIDs, we see an increase from NP-completeness to 2ExpTime-completeness.

For queries of bounded treewidth or hypertree width, we have seen that in fact, there is no benefit in terms of complexity when restricting the queries in this way. We still have 2ExpTime-completeness for all relevant query answering tasks, except for the aforementioned co-NP and ExpTime complexities in the data complexity, which also remain unchanged.

For acyclic queries, we observed the first reduction in complexity in some cases. In particular, when the arity is bounded by a constant, we climb one step lower on the exponential time hierarchy for the non-frontier-guarded classes of DTGDs, and even for them if we consider the entire set of rules fixed. However, combined and data complexity results still remain unchanged.

Finally, when restricting the query language to simple queries that only consist of a single atom, we observe a marked decrease in complexity, at least for the less expressive classes of DTGDs. For DIDs and linear DTGDs in particular, we were able to show a number of tractability results when the arity is bounded by a constant, and even membership in the highly parallelizable complexity class of AC_0 when considering the data complexity. Also the combined complexity decreases to ExpTime for these classes. The more expressive classes, starting with multi-linear DTGDs, however, retain their 2ExpTime complexity in the unrestricted case, and even in the bounded arity case, when considering the frontier-guarded classes. In all other

cases, we see a drop in complexity that now varies from co-NP-completeness to ExpTime-completeness, depending on the expressiveness of the class.

In conclusion, it can be observed that while disjunction does not have an impact on the decidability of the query answering problem for guarded-based classes of rules, it is a strong cause of complexity in this setting. Reducing the expressive power of the query language only provides a limited relief to this, except when the restrictions are very tight. However, in certain cases, tractability of the query answering problem, even in a non-deterministic setting, is still possible.

5 Sticky Sets of DTGDs

In this chapter, we extend the syntactic restriction of sticky sets of TGDs with disjunction, thereby yielding the notion of sticky sets of DTGDs. The extension also works for the other sticky-based restrictions introduced in Section 3.3. In case of the guarded-based classes discussed previously in Chapter 4, the finite treewidth model property also holds for disjunctive theories, based on the guarded fragment of first-order logic. However, the sticky property, which provides us with decidability in the deterministic case, is not strong enough to straightforwardly guarantee the same for non-deterministic theories. In fact, as we will see in this section, the combination of the limited joins allowed for by sticky theories and disjunction lead to undecidability of the query answering problem, even in more restricted scenarios. It thus seems that in the world of existential rules, the combination of joins and non-determinism is a cause of undecidability.

5.1 Extending Sticky Sets of TGDs with Disjunction

In order to extend sticky sets of TGDs with disjunction, we have to extend the syntactic condition of stickiness to also work for sets of DTGDs. Recall from Section 3.3 that stickiness is determined by employing a marking algorithm called SMarking. In fact, the formulation of the SMarking procedure given in that section is general enough to also treat DTGDs. The following is the definition of the SMarking procedure adapted to sets of DTGDs. Recall that for a DTGD σ, $head(\sigma)$ denotes all the atoms occurring in the head of σ, independent of any disjunctions σ may contain.

Definition 5.1. SMarking *takes a set Σ of DTGDs as input and returns a set of marked DTGDs where for each marked DTGD, every body variable is either marked, or not. The procedure works in two steps, where the second step is exhaustively applied until a fixpoint is reached.*

Initial Marking Step *For each DTGD $\sigma \in \Sigma$ and each variable V in $body(\sigma)$, if there exists an atom $\underline{a}$ in $head(\sigma)$ that does not contain V, mark V in σ.*

Propagation Step *For each TGD $\sigma \in \Sigma$ and each variable V in $body(\sigma)$, if there exists a $\sigma' \in \Sigma$ such that $head(\sigma)$ and $body(\sigma')$ both contain a relation r and the positions where V appears in r in $head(\sigma)$ coincide exclusively with positions of marked variables in $body(\sigma')$, then mark V in σ.*

Analogously to the definition of sticky sets of TGDs, we can now give the same definition of DTGDs.

Definition 5.2. *A set Σ of DTGDs is* sticky *if there is no TGD $\sigma \in \mathsf{SMarking}(\Sigma)$ such that a marked variable occurs in $body(\sigma)$ more than once.*

As can be seen from the above two definitions, the extension of the stickiness condition for sets of TGDs to sets of DTGDs is straightforward. In particular, the extended SMarking procedure should guarantee that the sticky property is valid in each and every model of the disjunctive chase vis-a-vis a sticky set of DTGDs.

5.2 Query Answering under Sticky Sets of DTGDs

As in the previous chapter, we would like to investigate different versions of the query answering problem, organized by the expressiveness of the query language. We again treat full, arbitrary conjunctive queries, bounded (hyper-)treewidth queries, acyclic queries and atomic queries. However, even in the case of sticky sets of (non-disjunctive) TGDs, we can see that in fact all of these problems are equivalent.

Before showing this, however, we will first deal with unions of the above-mentioned query types. The lemma below shows that, analogously to Lemma 4.6, we can show that UCQ answering can always be reduced to CQ answering. However, as opposed to Lemma 4.6, we can show that for sticky sets of DTGDs, this can be done without increasing the arity.

Lemma 5.3. *Consider a database D, a sticky set Σ of DTGDs, and a UCQ Q over schema $\mathcal{R}$. A database D', a sticky set Σ' of DTGDs and a CQ q over a schema $\mathcal{R}'$, with $arity(\mathcal{R}) = arity(\mathcal{R}')$ can be constructed in polynomial time such that $D \cup \Sigma \models Q$ if and only if $D' \cup \Sigma' \models q$.*

Proof. We assume, without loss of generality, that the CQs occurring in Q do not have variables in common. The proof is an adaptation of the proof of Lemma 4.6. Intuitively, we will reify the ternary or-relation the proof of Lemma 4.6 so that it is represented by three binary relations. In addition we keep track of the "real" chase by using the cross-product to associate "real" derivations with a marker constant. In this way, we eliminate the need to increase the predicate arity by one, as in Lemma 4.6. In the rest of the proof, let c, o_1, o_2, o_3, t and f be constants of $\mathbf{C}$ not occurring in D. Also, let λ be a substitution mapping all variables to the constant c. We are now ready to give the formal reduction:

The Schema $\mathcal{R}'$. The new schema $\mathcal{R}'$ is obtained from $\mathcal{R}$ by "doubling" it in addition to some auxiliary relations. Specifically, we define the new schema in the following way: $\mathcal{R}' = \mathcal{R} \cup \{r' \mid r \in \mathcal{R}\} \cup \{or_1, or_2, or_3, real, true, false\}$.

The Database D'. The new database D' is defined as follows:

$$\begin{aligned} & \{r'(\mathbf{t}) \mid r(\mathbf{t}) \in D\} \\ \cup \; & \{real(t,t), true(t), false(f)\} \\ \cup \; & \{or_1(o_1,t), or_2(o_1,t), or_2(o_1,f), or_3(o_1,t)\} \\ \cup \; & \{or_1(o_2,t), or_1(o_2,f), or_2(o_2,t), or_3(o_2,t)\} \\ \cup \; & \{or_1(o_3,f), or_2(o_3,f), or_3(o_3,f)\} \\ \cup \; & \{\lambda(q) \mid q \in Q\} \end{aligned}$$

The Set Σ'. The new set Σ' of DTGDs is obtained from Σ by replacing each atom of the form $r(\mathbf{X})$ occurring in a DTGD with the atom $r'(\mathbf{X})$. In addition, for every relation $r \in \mathcal{R}$, we add the following rules to Σ':

- $r'(\mathbf{X}) \rightarrow r(\mathbf{X})$, and
- $r'(X_1, \ldots, X_k), true(Z) \rightarrow real(X_i, Z)$, for each $0 < i \leq k$.

The CQ q. Assume that $Q = q_1 \vee \ldots \vee q_n$. For a CQ $q_i \in Q$, we define $q_i[X_i]$ as the CQ obtained from q_i by adding, for each variable $Y \in var(q_i)$, the atom $real(Y, X_i)$ to q_i, where X_i is a fresh variable not appearing in Q. For example, the query $\exists X \exists Y\, r(X,Y)$ would become $\exists X \exists Y\, r(X,Y), real(X,X_i), real(Y,X_i)$. The target CQ q is defined as follows:

$$\exists O_1 \ldots O_n \exists X_1 \ldots \exists X_n \exists Y_1 \ldots \exists Y_{n+1} false(Y_1) \wedge \bigwedge_{q_i \in Q} (q_i[X_i] \wedge or_1(O_i, Y_i) \wedge or_2(O_i, X_i) \wedge or_3(O_i, Y_{i+1})) \wedge true(Y_{n+1}).$$

Correctness. By construction, for each $q_i[X_i]$, a homomorphism exists that maps $q_i[X_i]$ to the database D', since we have added its image to D' (note that this homomorphism would always map X_i to f). However, this fact does not imply that the query q is trivially entailed by $chase(D', \Sigma')$ since, in order to satisfy the atom $true(Y_{n+1})$, at least one X_i must be mapped to the constant t. Thus, by construction, the only way to entail q is to map at least one sub-query $q_i[X_i]$ to $chase(D', \Sigma')$ via a homomorphism h, where also $h(X_i) = t$. Notice that the only atoms in $chase(D', \Sigma')$ containing t obtained from the original, renamed copy of D. Thus, a sub-query $q_i \in Q$ is entailed by an instance $I \in chase(D, \Sigma)$ if and only if the corresponding instance I' of $chase(D', \Sigma')$ entails q. Since the above construction is feasible in polynomial time, the claim follows. □

With the above proof in place, we can now proceed to show that, as stated, the considered query languages do in fact coincide in all relevant settings considered here, as the next statement shows:

Lemma 5.4. *The following problems, under sticky sets of TGDs (and thus also under sticky sets of DTGDs), are LogSpace-equivalent in both the combined and the data complexity:*

1. (U)CQ-Answering
2. (U)B(H)TWQ-Answering
3. (U)ACQ-Answering
4. (U)CQ_1-Answering

Proof. As atomic queries are acyclic which are in turn of bounded (hyper-)treewidth and thus a subclass of arbitrary CQs, the steps (4) ⇒ (3) ⇒ (2) ⇒ (1) are trivial. It thus only remains to show (1) ⇒ (4), that is that, under sticky sets of TGDs, CQ-Answering can be reduced to CQ_1-Answering.

In order to show this, let $\langle D, \Sigma, q \rangle$ be an instance of the CQ-Answering problem under sticky sets of TGDs over schema $\mathcal{R}$. We will construct an instance $\langle D, \Sigma', q' \rangle$ of the CQ_1-Answering problem, such that $D \cup \Sigma \models q$ if and only if $D \cup \Sigma' \models q'$. To this end, it is relevant to note that sticky TGDs allow for full cross-joins (i.e. cross products or Cartesian products) between relations.

Let $q = \exists \mathbf{X}\, \phi(\mathbf{X})$ be the original conjunctive query, where $\mathbf{X}$ is a sequence of variables, and $\phi(\mathbf{X}) = r_1(X_1^1, \ldots, X_{arity(r_1)}^1), \ldots, r_k(X_1^k, \ldots, X_{arity(r_k)}^k)$, where each X_j^i is a not necessarily distinct variable from $\mathbf{X}$.

Now, let $\Sigma' = \Sigma \cup \{\sigma\}$, where σ is the following TGD, and q is a new, $|\mathbf{X}|$-ary relation not appearing in $\mathcal{R}$.

$$r_1(X_1^1, \ldots, X_{arity(r_1)}^1), \ldots, r_k(X_1^k, \ldots, X_{arity(r_k)}^k) \rightarrow q(\mathbf{X})$$

Furthermore, we let $q' = \exists \mathbf{X}\, q(\mathbf{X})$. This completes the construction.

The construction is correct. Assume that $D \cup \Sigma \models q$. Then, there is a way to homomorphically map the query q into $chase(D, \Sigma)$. But this means that the body of σ can also be satisfied, and thus $chase(D, \Sigma')$ would apply rule σ and generate an atom $\underline{q}$ satisfying the head of σ. But this atom, by construction, satisfies the query q'.

Conversely, assume that $D \cup \Sigma' \models q'$. The only way to satisfy the query q' is to have an atom $\underline{q}$ appear in the chase that satisfies the head of the TGD σ. But, as the relation of $\underline{q}$ appears nowhere else in D or Σ', it must have been introduced by the chase via the application of the TGD σ. Thus there exists a homomorphism mapping $body(\sigma)$ into $chase(D, \Sigma')$, and by the fact that $body(\sigma)$ coincides with the query q, and σ does not take part in any recursive cycle in Σ, we have that $\Sigma = \Sigma' \setminus \{\sigma\}$ must satisfy q.

It can be easily verified that the above reduction is possible to perform in LogSpace. It is also obvious from the construction that, when both Σ and q are fixed, the reduction above implies that Σ' and q' are also fixed. The construction thus also works if we consider the case of data complexity. Together with Lemma 5.3, this completes the proof. □

We have thus shown that we can restrict our attention to arbitrary CQs, and all the complexity results obtained for the CQ-Answering problem indeed also apply to the other three cases, which we thus don't have to treat separately. It also shows that sticky TGDs provide an amount of expressivity that captures that of a conjunctive query, and thus the expressive power of query answering under sticky sets of TGDs is independent of the query language used. This is especially shown by the fact that, in the reduction above, we did not need to change the database at all. After establishing the above results, in the next section we will now continue on to show that disjunction actually destroys the decidability of query answering for sticky theories.

5.3 Undecidability of Query Answering

In this section, we will establish that the combination of stickiness and non-determinism leads to undecidability. In fact the problem lies in the combination of simple joins like cross-joins, and disjunction. Allowed to interact arbitrarily, as is the case in sticky sets of DTGDs, these two features lead to undecidability of query answering. We thus do not even need the full expressive power that stickiness provides. In fact it is sufficient to take a set of DIDs (which are clearly sticky sets of DTGDs, as they do not contain any joins or repeated variables), and allow a single, unary Cartesian product to appear. We thus define the class of $\times$-DIDs as follows:

Definition 5.5. *A set Σ of DTGDs is a set of $\times$-DIDs, if Σ can be partitioned into two sets Σ_{DID} and $\Sigma_{\times}$, such that Σ_{DID} is a set of DIDs, and $\Sigma_{\times}$ is a set of cartesian products, that is, TGDs of the form*

$$r(\mathbf{X}), s(\mathbf{Y}) \rightarrow rs(\mathbf{X}', \mathbf{Y}'),$$

where r, s and rs are relations and $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sequences of variables without repetition, with $\mathbf{X}' \subseteq \mathbf{X}$ and $\mathbf{Y}' \subseteq \mathbf{Y}$.

Clearly, $\times$-DIDs are still sticky sets of DTGDs: no variables can appear in the body more than once, and therefore the sticky condition can never be violated. With the above definition in place, we can now proceed to show our first undecidability result. In order to show undecidability of CQ-Answering under $\times$-DIDs, we will simulate a general, deterministic Turing machine and thus provide a reduction from the halting problem. The general idea is to use two IDs to force two infinite chains to appear in the chase. Using a cross product rule, the chains are then combined into an infinite grid. For each grid cell, a DID will then guess the tape contents. Using a union of conjunctive queries, we will then verify that the grid actually represents a valid sequence of computations. Reduction from the halting problem is one of the classical ways of showing undecidability, which gives us the following result:

Theorem 5.6. CQ-Answering *under $\times$-DIDs over a schema $\mathcal{R}$ is undecidable, even when the arity of $\mathcal{R}$ is bounded by the constant* 2*.*

Proof. We will show this by reduction from the co-halting problem. Let $M = \langle S, \Lambda, \delta, s_0 \rangle$ be a deterministic Turing machine, where S is a set of states with $s_0, s_h \in Q$ being the starting and halting state, $\Lambda = \{1, 0, \sqcup\}$ it's alphabet with $\sqcup$ the blank symbol and $\delta : S \times \Lambda \to S \times \Lambda \times \{-1, +1, 0\}$ the transition function. We assume that there is only one halting state s_h and that when the machine halts, it accepts, and has a do-nothing self-loop on s_h. We also assume that M takes no input, that is, the work tape contains only blanks, and that the head is, for technical reasons, at the second position of the tape and never moves further left than that. We will construct a database D, a set of $\times$-DIDs Σ and a CQ q over a schema $\mathcal{R}$, such that $D \cup \Sigma \models q$ if and only if M does not halt.

The idea of the construction is as follows: we will construct an infinite grid, where each row represents a configuration of the Turing machine and each column a position on the tape. Using a disjunctive rule, we will guess, for each position and configuration, which symbol there is on the tape. With a CQ q, we will then check that at least one model of the chase represents a valid computation of M and that this computation halts. To this end, let $S' = (S \times \Lambda) \cup \Lambda$.

The Schema $\mathcal{R}$. The schema $\mathcal{R}$ contains the following predicates:

- $init(\cdot, \cdot)$: This predicate encodes the initial two configurations and cells.
- $conf(\cdot, \cdot)$: This predicate encodes two subsequent configurations.
- $cell(\cdot, \cdot)$: This predicate encodes two subsequent tape cells.
- $confcell(\cdot, \cdot)$: This predicate encodes the combination of a configuration and a tape cell.
- $s(\cdot, \cdot)$ for each $s \in S'$: This predicate encodes that a combination of a configuration and a tape cell is labelled with s.

The Database D. In our database D we will simply store two constants a, b in a relation $init(a, b)$.

The Set of $\times$-DIDs Σ. First, we construct our grid, using the following rules. Note that for brevity, the rules below are not strictly speaking DIDs, but can be easily transformed into single-atom head rules via, for example, the normalization procedure presented in Chapter 4.

- $init(X, Y) \to conf(X, Y)$
- $init(X, Y) \to cell(X, Y)$
- $conf(X, Y) \to \exists Z\, conf(Y, Z), null(Z)$
- $cell(X, Y) \to \exists Z\, cell(Y, Z), null(Z)$
- $conf(X_1, Y_1), cell(X_2, Y_2) \to confcell(X_1, X_2)$

We add a DID that guesses which element from S' is true at each $confcell$:

- $confcell(X_1, X_2) \rightarrow \bigvee_{s \in S'} s(X_1, X_2)$

This concludes the construction of Σ.

The CQ q. In order to construct q, we will first construct a UCQ Q, which makes it easier to grasp what conditions the query checks, as the different disjuncts in Q check for certain, independent conditions. Intuitively, Q consists of the following disjuncts:

1. $Q_{initial}$ that checks that every tape cell in the first configuration is labelled with ⊔, and the head is at position 2, with state s_0.
2. Q_{trans} that checks the consistency of two subsequent configurations w.r.t. the transition function δ.
3. Q_{inert} that checks the inertia of the tape cells not affected by a transition of the machine.
4. Q_{halt} that checks whether we reach the halting state of the machine.

In the following, we give the formal definition of Q. In order to check the validity of the transitions, and model the inertia rules, notice that we can represent transitions in δ as a six-tuple $abc \rightarrow a'b'c'$ mapping a 3-tuple of symbols from S' (i.e. including the state) in one configuration to a 3-tuple of symbols in the next configuration, where thus $a, c \in \Lambda$, $b \in S \times \Lambda$ and $a', b', c' \in S'$, obviously with only one of a', b', c' from $S \times \Lambda$. With this observation in place, we can now give the formal definition of Q:

1. $Q_{initial}$ is made up of three parts as follows, checking the first, second, and subsequent positions of the tape respectively:
   - $\exists X \exists Y\, init(X, Y) \wedge (0(X, Y) \vee 1(X, Y))$
   - $\bigvee_{[s,\lambda] \in (S' \setminus \{[s_0, \sqcup]\})} \exists X \exists Y\, init(X, Y) \wedge [s, \lambda](X, Y)$
   - $\exists X \exists Y \exists Z\, init(X, Y) \wedge null(Z) \wedge (0(X, Y) \vee 1(X, Y))$
2. Q_{trans} is made up of multiple disjuncts as follows. For each of the possible rewriting rules $abc \rightarrow a'b'c'$ that do not occur in δ, we add the following disjunct, that will thus make the query true if the transition function has been violated.

$$\exists X \exists Y \exists X_1 \ldots X_3\, conf(X, Y) \wedge cell(X_1, X_2) \wedge cell(X_2, X_3) \wedge$$
$$a(X, X_1) \wedge b(X, X_2) \wedge c(X, X_3) \wedge a'(Y, X_1) \wedge b'(Y, X_2) \wedge c'(Y, X_3)$$

3. Q_{inert} checks for a triple of symbols from Λ on the tape and checks that the next configuration contains the same ones. It is thus similar in structure to Q_{trans}. For each triple abc of constants from $\Lambda \times \Lambda \times \Lambda$, and constant $d \in \Lambda \setminus \{b\}$, we add the following disjunct:

$$\exists X \exists Y \exists X_1 \dots X_3 \, conf(X, Y) \wedge cell(X_1, X_2) \wedge cell(X_2, X_3) \wedge a(X, X_1) \wedge b(X, X_2) \wedge c(X, X_3) \wedge d(Y, X_2)$$

4. $Q_{halt} \equiv \exists X \exists Y [s_h, 0](X, Y) \vee [s_h, 1](X, Y) \vee [s_h, \sqcup](X, Y)$

This completes the construction of the UCQ Q.

Correctness. The above construction is correct. Assume that $D \cup \Sigma$ does not entail Q. Then this means that there is an instance $I \in chase(D, \Sigma)$, such that none of the disjuncts of Q are true in I. But because I, by construction and non-entailment of $Q_{initial}$, Q_{trans} and Q_{inert}, represents a valid computation of M, and I does not entail Q_{halt}, this means that there must be an infinite computation of M that does not reach the halting state. Note that as M is deterministic, there is exactly one such I. Conversely, assume that the query Q holds in every model of $D \cup \Sigma$. This means that every model either encodes an invalid computation, or a valid computation that reaches the halting state, by construction of Q_{halt}. We thus have as desired that $D \cup \Sigma \models Q$ if and only if M does not halt. Also note that the above construction uses only $\times$-DIDs, and the schema has at most arity 2.

However, the proof yields a UCQ instead of our desired CQ. It remains to show that the above result also holds for CQs. Indeed, this step follows immediately from Lemma 5.3, as discussed previously, which does not increase the arity and also works for $\times$-DIDs, as can be easily verified. This concludes the proof. □

In the above proof, we have shown that the unconstrained combination of disjunction and cross products is a definite cause for undecidability. However, as can be easily seen from the construction of the proof, both the query q and the theory Σ depend on the simulated Turing machine. We have seen that a maximum arity of 2 is sufficient for the construction, but what about the case where either q or Σ or both are fixed? The next result adapts the above proof to also hold in the case of data complexity, at the cost of increasing the maximum arity by four to at most 6.

Theorem 5.7. CQ-ANSWERING *of fixed CQs under fixed sets of $\times$-DIDs over a schema $\mathcal{R}$ is undecidable, even when the arity of $\mathcal{R}$ is bounded by the constant* 6*.*

Proof. The proof is by reduction from the co-halting problem of a deterministic Turing machine. This proof is an adaptation of the proof of Theorem 5.6. Again, let M be a deterministic Turing machine, as in the previous proof. Without loss of generality, we will use the same assumptions on M. Again, let $S' = (S \times \Lambda) \cup \Lambda$.

The Schema $\mathcal{R}$. The schema $\mathcal{R}$ contains the same predicates as in the previous proof, except for the ones dependant on M. For completeness, we will list them again here, and in addition, we will make use of some other predicates as listed below:

- $init(\cdot,\cdot)$: This predicate encodes the initial two configurations and cells.
- $conf(\cdot,\cdot)$: This predicate encodes two subsequent configurations.
- $cell(\cdot,\cdot)$: This predicate encodes two subsequent tape cells.
- $confcell(\cdot,\cdot)$: This predicate encodes the combination of a configuration and a tape cell.
- $illegal(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$ will hold six-tuples representing rewriting rules of the form $abc \rightarrow a'b'c'$ that do not appear in the transition function δ of M, when interpreting the δ as a set of such rewriting rules.
- $symbol(\cdot)$ will hold all the symbols from S' as dedicated constants from **C**.
- $sneq(\cdot,\cdot)$ will represent the non-equivalence relation over the elements in S'.
- $[no]label(\cdot,\cdot,\cdot)$ will hold information on whether or not a certain $confcell$ is labelled with a symbol from S' or not.
- $confcellsym(\cdot,\cdot,\cdot)$ will be used as an auxilliary atom, whose meaning will become clear during the construction of the proof.

The Database D. In our database D we will store two constants a, b from **C** in a relation $init(a,b)$ as before. In addition, we add to D an atom $symbol(s)$, where s is a fresh constant from **C**, for each element $s \in S'$, and for any two such elements s_1 and s_2, where $s_1 \neq s_2$, we add the atom $sneq(s_1,s_2)$. Finally, we again interpret the transition function δ of M as a set of rewriting rules of the form $abc \rightarrow a'b'c'$, and add, for each possible rewriting rule that does not appear in δ, an atom $illegal(a,b,c,a',b',c')$ to D. This concludes the construction of D.

The Set Σ of $\times$-DIDs. Regarding the set Σ, we will use the exact same rules as proposed in the previous proof, except for the last one, as it depends on M. In its stead we will use two new rules to guess which symbol from S' appears at a given position of the infinite grid. The two rules are as follows, which concludes the adaptation of Σ.

- $confcell(X_1,X_2), symbol(Y) \rightarrow confcellsym(X_1,X_2,Y)$
- $confcellsym(X_1,X_2,Y) \rightarrow label(X_1,X_2,Y) \vee nolabel(X_1,X_2,Y)$

The Query q. Again, we will use a UCQ Q to make construction easier. By Lemma 5.3 we then get the translation from UCQ Q to CQ q. Note that, when looking at the construction of the previous proof of Theorem 5.6, only the sub-query Q_{trans} depends on the Turing machine M. Thus, we will only adapt this sub-query, and leave the others in place as they are. However, first, we need to add another sub-query to Q, namely the sub-query q_{label} that checks that no $conf\,cell$ is labelled with two different symbols from S'. Note that in the previous construction, this could not have been the case. To this end, we define q_{label} as follows:

$$q_{label} \equiv \exists X_1 \exists X_2 \exists Y \exists Z\, label(X_1, X_2, Y) \wedge label(X_1, X_2, Z) \wedge sneq(Y, Z)$$

Finally, we adapt the sub-query Q_{trans} to have only one disjunct as follows:

$$\begin{aligned} Q_{trans} \equiv \exists V_1 \exists V_2 \exists W_1 \exists W_2 \exists W_3 \exists X \exists Y \exists Z \exists X' \exists Y' \exists Z'\, conf(V_1, V_2) \wedge \\ cell(W_1, W_2) \wedge cell(W_2, W_3) \wedge \\ label(V_1, W_1, X) \wedge label(V_1, W_2, Y) \wedge label(V_1, W_3, Z) \wedge \\ label(V_2, W_1, X') \wedge label(V_2, W_2, Y') \wedge label(V_2, W_3, Z') \wedge illegal(X, Y, Z, X', Y', Z') \end{aligned}$$

The sub-query Q_{trans} now checks precisely the same condition as in the proof of Theorem 5.6, but instead of relying on multiple disjuncts, one for each illegal combination, now the query is static, and the illegal combinations are stored in the database. This concludes the construction of Q.

Correctness. It is now not difficult to verify that the adapted construction, as given above, still fulfills all the properties that the original construction also exhibits, that is, again, $D \cup \Sigma \models Q$ if and only if M does not halt. □

We thus have shown, as desired, that even for fixed sets of ×-DIDs and fixed queries, that is, in the data complexity, the CQ-Answering problem is still undecidable. This result provides further evidence to the postulate, that the combination of cross joins and disjunction is a strong cause of undecidability. Unfortunately, it seems that without stringent restrictions, decidability cannot be achieved. The following corollary follows immediately from the above results, and the fact that sets of ×-DIDs are sticky sets of DTGDs:

Corollary 5.8. (U)CQ-Answering, (U)B(H)TWQ-Answering, (U)ACQ-Answering *as well as* (U)CQ_1-Answering *under sticky sets of DTGDs are undecidable, even if we consider only the data complexity.*

5.4 Summary

To summarize, we have shown in the present chapter that already sticky TGDs are a very expressive formalism powerful enough to capture arbitrary conjunctive queries. In Lemma 5.4,

we have shown that CQ-Answering can be reduced to CQ_1-Answering, and together with Lemma 5.3, we have that the same even holds for UCQ-Answering. Thus, for complexity investigations, it is sufficient to focus on one particular type of query, arbitrary UCQs in our case.

In Section 5.3, we have established that even for $\times$-DIDs, a sub-formalism of sticky sets of DTGDs that contains only DIDs and cross product TGDs, the UCQ-Answering problem, and thus CQ-Answering, B(H)TWQ-Answering, ACQ-Answering and CQ_1-Answering are all undecidable, even when the arity of the schema is bounded by two. Further, at the cost of increasing the maximum arity to six, we have shown that these problems remain undecidable even in the data complexity, that is, both the set of DTGDs and the query are fixed.

In conclusion this shows that for reasoning under existential rules, the combination of joins and non-deterministic constructs is a strong indicator and cause of undecidability of the main reasoning tasks, query answering in particular.

6 Weakly-Acyclic Sets of DTGDs

In this chapter, we extend the syntactic restrition of weakly-acyclic sets of TGDs with disjunction, thereby yielding the notion of weakly-acyclic sets of DTGDs. We will herein focus solely on the notion weak-acyclicity, and omit discussing the case of full TGDs, for the reason that they do not exhibit the crucial feature of existential quantification, as mentioned in the thesis title. We also note that full TGDs with disjunction have been studied under the name *Disjunctive Datalog*, and we refer the interested reader to the relevant literature; see, e.g. Eiter et al. [1997]; Dantsin et al. [2001]. As we will see, the extension of the syntactic condition to DTGDs is straightforward, and guarantees termination of the chase, as in the deterministic case. Therefore, we get decidability for free, as we can simply compute the (finite) universal model set and answer the query over each model in it in finite time. However, the question of the exact complexity of query answering remains unanswered. While some results can be inherited from existing results, several questions remain open. In the following sections we will flesh out the exact complexity of answering arbitrary queries, bounded (hyper-)treewidth queries, acyclic and atomic queries, and unions of them, in the traditionally relevant cases: Combined complexity, bounded arity, fixed set of DTGDs and data complexity. We will see that there is an increase in complexity from the non-disjunctive to disjunctive case, nicely reflecting the change from deterministic to non-deterministic reasoning. The first section below will deal with the formal extension of the notion of weak-acyclicity to DTGDs.

6.1 Extending Weakly-Acyclic Sets of TGDs with Disjunction

Analogously to the previous chapters, we will first adapt the syntactic condition of weak-acyclicity to DTGDs. Recall from Section 3.2 that to determine whether a given set of TGDs is weakly-acyclic, we have to build its dependency graph, as described in Definition 3.13. We will first extend this notion to DTGD. In order to do this, we need to guarantee that every branch of the chase leads to a finite model. While a trivial extension of the definition of the dependency graph (i.e. one where we replace the word “TGD” by “DTGD”) would work, this would turn out to be overly restrictive. The main idea of the extension is that the disjuncts in the heads of the DTGDs can be treated separately, instead of combining them and treating them as a single unit. The definition of the dependency graph for a set of DTGDs is given below:

Definition 6.1. *The* dependency graph $G = \langle V, E = E_N \uplus E_S \rangle$ *of a set* Σ *of DTGDs over a schema* $\mathcal{R}$ *is a multigraph, where the multiset of edges is partitioned into normal edges* E_N *and special edges* E_S*. G is constructed as follows:*

- *Let V be the set of positions $r[i]$ occurring in $\mathcal{R}$.*
- *Add a normal edge $(r[i], s[j])$ to E_N, if there exists a DTGD $\sigma \in \Sigma$, such that relation r occurs in the body of σ and relation s occurs in the head of σ, and the positions $r[i]$ and $s[j]$ are occupied by the same (universally quantified) variable in σ.*
- *Add a special edge $(r[i], s[j])$ to E_S, if there exists a DTGD $\sigma \in \Sigma$, such that relation r occurs in the body of σ and relation s occurs in the disjunct ϕ of the head of σ, and furthermore, the (universally quantified) variable at position $r[i]$ in σ also appears in ϕ, and an existentially quantified variable appears at position $s[j]$ in ϕ.*

Notice that, intuitively, the dependency graph of a set Σ of DTGDs is the same as the dependency graph of a set $\Sigma' = \pi(\Sigma)$ of TGDs obtained from Σ by means of a transformation π, where $\pi(\Sigma)$ replaces every DTGD $\sigma \in \Sigma$ of the form $\phi(\mathbf{X}) \rightarrow \bigvee_i \exists \mathbf{Y}_i \psi_i(\mathbf{X}, \mathbf{Y}_i)$ with the set of TGDs $\phi(\mathbf{X}) \rightarrow \exists \mathbf{Y}_i \psi_i(\mathbf{X}, \mathbf{Y}_i)$, for all i ocurring in σ. That is, we can, in a way, treat the disjunctions as conjunctions, in order to construct the dependency graph of Σ.

Having the notion of the dependency graph in place, we can now define weakly-acyclic sets of DTGDs. In fact, with the adapted version of the dependency graph, the definition is the same as for sets of TGDs:

Definition 6.2. *A set Σ of DTGDs is called* weakly-acyclic *if and only if its dependency graph does not contain a cycle through a special edge.*

From the above definition, we immediately get that every model in the chase is finite.

Theorem 6.3. *Given a weakly-acyclic set Σ of DTGDs, then, for any database D, $chase(D, \Sigma)$ terminates, and every model $U \in chase(D, \Sigma)$ is of at most double-exponential size, and at most polynomial size when Σ is fixed.*

Proof. Let D be an arbitrary database. Using the transformation π introduced above, the set $\Sigma' = \pi(\Sigma)$ is a weakly-acyclic set of TGDs. It is already implicit in [Fagin et al., 2005; Larson, 2006], and explicitly proven in [Pieris, 2011, Lemma 6.1], that the chase of a database with a weakly-acyclic set of TGDs can be built in double-exponentially many steps, and in polynomially many steps if the set of TGDs is fixed. This implies that the maximal size of any universal model that the chase builds is also bounded in the same way. In particular, this thus applies to the universal model $M = chase(D, \Sigma')$.

It is easy to verify that for every model $U \in chase(D, \Sigma)$ it holds that $U \subseteq M$, up to isomorphism. In fact, by construction of the transformation π, any atom derived in any such model U, or an isomorphic equivalent, will necessarily be contained in M. Towards a contradiction, assume that some $U \in chase(D, \Sigma)$ contains an atom $\underline{a}$, such that $\underline{a}$, or an atom isomorphic to $\underline{a}$, does not occur in M. Unless $\underline{a} \in D$, which is obviously a contradiction, $\underline{a}$ must have been derived by the application of a sequence of DTGDs $(\sigma_1, k_1), \ldots, (\sigma_n, k_n)$, where (σ_i, k_i) represents the application of the DTGD σ_i via the k_i-th disjunct of its head. In fact, the sequence can

therefore be interpreted as a sequence of TGD applications. However, by the construction of transformation π, all TGDs of that sequence also exist in Σ', and the chase would hence derive $\underline{a}$, or an atom isomorphic to $\underline{a}$ in M, a contradiction.

As the size of M is at most double-exponential in general, or polynomial in case of a fixed set Σ, and by the fact that every model $U \in chase(D, \Sigma)$ is a subset of M up to isomorphism, we have that the size of U is bounded by the size of M, which completes the proof. □

We thus have that the same upper bounds that apply to the size of the single universal model that the chase builds in case of TGDs, also applies to each model in the universal model set in case of DTGDs. We will make use of the above theorem to establish complexity upper bounds for query answering in Section 6.4. But let us first proceed to investigate, in the next section, some general observations regarding query answering under weakly-acyclic sets of DTGDs.

6.2 Query Answering under Weakly-Acyclic Sets of DTGDs

As in the previous chapters, we would like to investigate different versions of the query answering problem, where the expressivity of the query language varies. Again, we will treat arbitrary conjunctive queries, bounded (hyper-)treewidth queries, acyclic queries and atomic queries, and we would also like to consider the cases where we deal with unions of queries of each type. However, analogously to the case of sticky sets of DTGDs, we can show that we can reduce answering unions of arbitrary conjunctive queries to answering a single atomic query.

Lemma 6.4. *The following problems, under weakly-acyclic sets of TGDs (and thus also DTGDs), are LogSpace-equivalent in both the combined and the data complexity:*

1. (U)CQ-Answering
2. (U)B(H)TWQ-Answering
3. (U)ACQ-Answering
4. (U)CQ_1-Answering

Proof. Analogously to the case of sticky sets of DTGDs, we only need to provide a reduction from answering arbitrary UCQs to answering a single atomic query, as all other reductions needed to complete the cycle are trivial. Assume thus, that D is a database, Σ a set of weakly-acyclic TGDs and Q an arbitrary UCQ, all over schema $\mathcal{R}$. We will construct a weakly-sticky set of TGDs Σ' and a (propositional) atomic query $q = p$ over a schema $\mathcal{R}'$, such that $D \cup \Sigma \models Q$ if and only if $D \cup \Sigma' \models q$.

In fact, the reduction is straightforward, and uses the same construction as in the proof of Lemma 4.5. Let $Q = q_1 \vee \ldots q_n$, where q_i is a disjunct in Q. Let $\mathcal{R}' = \mathcal{R} \cup \{p\}$, where p is a propositional relation. Let $\Sigma' = \Sigma \cup \{q_i \rightarrow p \mid 1 \leq i \leq n\}$, that is, the original set Σ and a set of

query-rules that represent the query Q. It is indeed easy to verify that, as desired, $D \cup \Sigma \models Q$ if and only if $D \cup \Sigma' \models q$.

It remains to show that Σ' is in fact weakly-acyclic. If it were not, this means that by the addition of the query-rules to Σ, a cycle through a special edge was introduced into the dependency graph. However, as p is a propositional atom that appears nowhere in Σ, no additional edges were introduced into the dependency graph by the addition of the query-rules, and thus no cycle in the dependency graph could have been created by their addition to Σ.

It is easy to verify that this reduction can be completed in LogSpace, and that, if the set Σ and the query Q are fixed, it yields a fixed set Σ' and a fixed atomic query q. It thus works both in the combined and in the data complexity as desired, which completes the proof. □

Again, it seems we can focus our attention on UCQs, when determining the complexity of query answering. However, notice that the above reduction does not work if only the set of rules Σ is fixed, but the query Q is not. Thus, in this case, we have to treat the different query languages separately, as the respective query answering problems are not equivalent. However, notice that this is indeed the only case warranting separate treatment. In case of bounded arity, the above reduction still works, as it does not modify the maximum arity of the schema in any way, as it only introduces a single, propositional relation into it.

With the above general results in place, we can now proceed to thoroughly investigate the complexity of the query answering problem under weakly-acyclic sets of DTGDs. The next section will provide an overview of the complexity results obtained.

6.3 Complexity Overview

	Combined Complexity	Bounded Arity	Fixed Theory	Data Complexity
(U)CQ$_1$s	co-N2ExpTime	co-N2ExpTime LB: Thm. 6.7	co-NP	co-NP LB: Thm. 6.9
(U)ACQs	co-N2ExpTime	co-N2ExpTime	co-NP	co-NP
(U)B(H)TWQs	co-N2ExpTime	co-N2ExpTime	co-NP UB: Thm. 6.6	co-NP
(U)CQs	co-N2ExpTime UB: Thm. 6.5	co-N2ExpTime	Π_2^P UB: Thm. 6.6 LB: [Bárány et al., 2014]	co-NP UB: Thm. 6.6

Table 6.1: The complexity of (U)CQ-Answering under weakly-acyclic sets of DTGDs.

Table 6.1 summarizes the complexity results for answering conjunctive queries of different types under weakly-acyclic sets of DTGDs. Each row represents a query language, ordered from top to bottom by increasing expressive power. Recall that (U)CQ$_1$s are (unions of)

atomic queries, (U)ACQs are (unions of) acyclic queries, (U)B(H)TWQs are (unions of) queries of bounded (hyper-)treewidth, and (U)CQs are (unions of) conjunctive queries. This means that all the results represented in the table hold both for queries of the respective type, and unions of such queries. All cells in the table represent completeness results.

In each cell of the table, we have indicated where to find the corresponding results (UB and LB stands for upper and lower bound, respectively). Note that the missing references for the upper (resp., lower) bounds are immediately inherited from the first lower-left (resp., upper-right) cell in which a reference is given. The results of this section, together with a brief description of the employed techniques, are outlined below:

Upper Bounds:

1. UCQ-Answering under weakly-acyclic sets of DTGDs is in co-N2ExpTime in the combined complexity (Theorem 6.5) — this is established by exhibiting a guess-and-check algorithm that runs in non-deterministic double-exponential time;
2. UCQ-Answering under weakly-acyclic sets of DTGDs is in Π_2^{P} in case where the set of DTGDs is fixed (Theorem 6.6) — this is established by exhibiting a co-$\mathrm{NP}^{\mathrm{NP}}$ guess-and-check algorithm; and
3. UCQ-Answering under weakly-acyclic sets of DTGDs is in co-NP in case where the query is either of bounded (hyper-)treewidth, or fixed (Theorem 6.6) — this is established by exhibiting a co-NP guess-and-check algorithm.

Lower Bounds:

1. CQ_1-Answering is shown to be co-N2ExpTime-hard, even for predicate arities of at most three (Theorem 6.7) — this is established by a reduction from the Square-Tiling problem for a square of double-exponential size in a given number n; and
2. CQ_1-Answering is shown to be co-NP-hard in the data complexity (Theorem 6.9) — the result follows from classical data complexity results for query answering over Description Logics established by Calvanese et al. [2013], in combination with the fact that acyclic queries can be reduced to atomic queries in the data complexity, as established by Lemma 6.4.

Clearly, the co-N2ExpTime upper bound for UCQ-Answering in the combined complexity and the co-N2ExpTime lower bound for CQ_1-Answering in case of bounded arity close the complexity picture for these two columns of the table. For fixed sets of DTGDs, the co-NP upper bounds in case of either fixed or bounded (hyper-)treewidth queries, plus the co-NP lower bound for CQ_1-Answering in the data complexity close all remaining cells except for the cell of (U)CQ-Answering in case where the set of DTGDs is fixed. A Π_2^{P} upper bound is provided, and existing results provide us with the missing optimal lower bounds in this case.

Inherited Results:

1. CQ-Answering under guarded first-order logic sentences is Π_2^{P}-hard in case where the sentence is fixed [Bárány et al., 2014, Proposition 5.5] — in fact, it is shown in this result that answering a CQ under a fixed sentence of the form $\forall v\, X(v,t,f) \rightarrow V(v,t) \bigoplus V(v,f)$, which, under the notion of query answering considered in this thesis is semantically equivalent to the DTGD $r(X,t,f) \rightarrow s(X,t) \vee s(X,f)$, is Π_2^{P}-hard.

From the results of this section, we can observe that there exists an intuitive increase in the complexity of the query answering problem, when we allow the construct of disjunction to appear. In fact the combined complexity increases from (deterministic) 2ExpTime under TGDs to co-N2ExpTime under DTGDs. Similarly, the data complexity increases from polynomial time feasible under TGDs to co-NP under DTGDs. We can thus say that for weakly-acyclic sets of rules, the addition of disjunction causes a jump in complexity from the respective class of deterministic time to the corresponding class of non-deterministic time.

Let us now proceed with the formal proofs of our results.

6.4 Upper Bounds

In this section, we will give proofs for the relevant complexity upper bounds pertaining to the query answering problem under weakly-acyclic sets of DTGDs. We will start by showing a co-N2ExpTime complexity upper bound for our most general case, UCQ-Answering under weakly-acyclic sets of DTGDs. In fact, by the equivalence shown in Lemma 6.4, we only need to show the upper bound for the CQ_1-Answering problem, which is simpler as we do not need to separately treat a complex conjunctive query. The idea of the proof is to exhibit a N2ExpTime guess-and-check algorithm for the co-problem, that guesses a model of at most double-exponential size and checks that the (now atomic) query is not satisfied in it.

Theorem 6.5. UCQ-Answering *under weakly-acyclic sets of DTGDs is in co-N2ExpTime in the combined complexity.*

Proof. In order to show membership in co-N2ExpTime, we will show membership of the co-problem in N2ExpTime. To start, we will first focus on the co-CQ_1-Answering problem. We thus have to proof the existence of a non-deterministic double-exponential time Turing machine that, given a database D, a weakly-acyclic set Σ of DTGDs and an atomic query q, decides whether there exists a model M of $D \cup \Sigma$ (or, in this case equivalently, of $chase(D,\Sigma)$), such that no homomorphic image of q appears in M.

By Theorem 6.3, we know that each model $M \in chase(D,\Sigma)$ is at most of double-exponential size. It is not difficult to see that therefore, a non-deterministic double-exponential time Turing machine can build each such model M. It only remains to show that each M can be checked, in double-exponential time, to not contain a homomorphic image of q. In fact, the straightforward approach will suffice: scan through the model, atom by atom, and for each such atom

$\underline{a}$, check whether q can be homomorphically mapped to $\underline{a}$. Clearly, checking whether a homomorphism from one atom to another atom exists is possible in polynomial time. As every model M contains at most a double-exponential number of atoms, we get the desired non-deterministic algorithm.

The above proof works only for atomic queries. However, by the fact that there exists a LogSpace-reduction from the UCQ-Answering problem to the CQ_1-Answering problem as shown in Lemma 6.4, we get that the upper bound holds also for UCQ-Answering, as desired. This concludes the proof. □

As we have seen above, a non-deterministic double-exponential time Turing machine can decide our problem in general. Compared to CQ-Answering under weakly-acyclic sets of TGDs, we can see that an intuitive increase in complexity from deterministic double-exponential time to non-deterministic double-exponential time takes place, when we allow disjunctive rules. However, the above proof only provides us an upper bound for the combined complexity. As we will see in the next section, restricting the arity of the schema will not decrease the complexity.

The missing results are therefore the case where the entire set of DTGDs is fixed, and the case of the data complexity. The next theorem below will provide us with the relevant upper bounds for all of these cases. Note that Lemma 6.4 does not work in the case where the set of DTGDs is fixed. We therefore have to provide a separate complexity investigation for each query answering problem. However, as we will soon see, most of the cases feature the same complexity.

The general idea of the proof is to use the fact that every model that the chase builds is at most of polynomial size when the set of DTGDs is fixed. We can thus use an NP Turing machine to construct all of these models. Further, we will use an oracle to check whether the query is satisfied in a certain model. This oracle must be powerful enough to solve the traditional query answering problem over databases, that is, without DTGDs. This can be done by treating the models that the chase builds as databases.

Theorem 6.6. *The* UCQ-Answering *problem under weakly-acyclic sets of DTGDs is in* Π_2^P *in case of a fixed set of DTGDs, and in co-NP if, in addition, the query is fixed, or a query of bounded (hyper-)treewidth.*

Proof. Let D be the database, Σ the fixed, weakly-acyclic set of DTGDs and Q the union of conjunctive queries of the UCQ-Answering problem. Analogously to the argument in the proof of Theorem 6.5, we can show the following: by Theorem 6.3, we know that each model $M \in chase(D, \Sigma)$ is at most of polynomial size when the set of DTGDs is fixed. It is not difficult to see that therefore, a non-deterministic polynomial time Turing machine can build each such model M.

To check whether for every $M \in chase(D, \Sigma)$ it holds that $M \models Q$, we thus need a co-NP$^{\mathcal{C}}$ algorithm, where $\mathcal{C}$ is the complexity class needed to answer the question whether $M \models Q$. As

the size of M is guaranteed to be polynomial, we can treat this as the classical query answering problem over databases, without DTGDs. It turns out that there are only three cases which need be considered:

- In case of a non-fixed, arbitrary union of conjunctive queries, the question whether $M \models Q$ is well-known to be in NP, as shown by Chandra and Merlin [1977]. Thus, $D \cup \Sigma \models Q$, or equivalently, $chase(D, \Sigma) \models Q$ is in co-$\text{NP}^{\text{NP}} = \Pi_2^{\text{P}}$.
- In the case where Q is fixed (i.e. in the data complexity), query answering is known to be feasible in polynomial time (actually even in AC_0; see Vardi [1995]). We thus have that $D \cup \Sigma \models Q$, or equivalently, $chase(D, \Sigma) \models Q$ is in co-$\text{NP}^{\text{AC}_0} =$ co-NP.
- In the case where Q is acyclic, it is well-known that the query answering problem can be solved in polynomial time, as shown by Yannakakis [1981]. In [Gottlob et al., 2001], the exact complexity of the problem is shown to be LogCFL-complete, and further it is shown that this completeness result holds even for queries of bounded (hyper-)treewidth. As LogCFL $\subseteq$ PTime, we have that $D \cup \Sigma \models Q$, or equivalently, $chase(D, \Sigma) \models Q$ is in co-$\text{NP}^{\text{LogCFL}} =$ co-NP.

This concludes the proof. □

In case of the data complexity, we can see that disjunction causes a jump from polynomial time complexity to non-deterministic polynomial time. This is an analogous complexity jump to the combined complexity, where we had double-exponential time in the deterministic case, and non-deterministic double-exponential time when disjunction is allowed. Again, the impact of disjunction on the complexity of the query answering problem is very intuitive.

It remains to show that the above results are indeed completeness results. This will follow from the corresponding lower bounds, which we will investigate in the next section.

6.5 Lower Bounds

This section contains the relevant lower bounds to complete the complexity picture of the query answering problem under weakly-acyclic sets of DTGDs. In fact we will see that all the upper bounds discussed in the previous section are indeed matched by corresponding lower bounds.

We will start the section with a co-N2ExpTime lower bound for the query answering problem under fixed predicate arities. Recall that by Lemma 6.4, we can show hardness for the UCQ-Answering problem, and such a result immediately extend also to the CQ_1-Answering problem. The idea of the proof is to simulate the square tiling problem for a square of double-exponential size. Given a set of tiles, and the knowledge which tiles are allowed to be next to each other, the problem is to determine whether a square of double-exponential size in some

number n has a valid tiling w.r.t. these restrictions. The concrete version of the square tiling problem that we will use to show the desired results is defined as follows:

SQUARE-TILING

Instance: A set $T = \{t_0, \dots, t_k\}$ of tiles, two binary relations $H, V \subseteq T \times T$ called the *horizontal* and *vertical compatibility relations*, and an integer n.

Question: Does there exist a function $f : \{1, \dots, 2^{2^n}\}^2 \to T$, s.t. $f(1,1) = t_0$ and for all other $1 \leq i, j \leq 2^{2^n}$ it holds that $(f(i,j), f(i+1,j)) \in H$ and $(f(i,j), f(i,j+1)) \in V$?

The SQUARE-TILING problem (for squares of double-exponential size) is known to be complete for N2EXPTIME; see, e.g. Papadimitriou [1994]. In fact, for a polynomial-sized square, the problem is NP-complete, and for an exponential-sized square, the problem is NEXPTIME-complete. With the above problem definition in place, the general idea of proving our desired CO-N2EXPTIME lower bound is as follows: exploiting a construction given in [Pieris, 2011, Proof of Theorem 6.2], we can, using an appropriate weakly-guarded set of DTGDs, construct a double-exponential number of distinct nulls in a successor relation. Doing so twice, and building the cross product of the two successor relations, yields a grid of double-exponential size, representing our square that is to be tiled. Using a disjunctive rule, we can guess a tile for each position of the grid and then, using a union of conjunctive queries, we can check whether the thus obtained tiling violates the compatibility relations. This approach gives us the following statement. The formal proof can be found below.

Theorem 6.7. *The* CQ-ANSWERING *problem under weakly-guarded sets of DTGDs is hard for co-N2EXPTIME, even if the maximum arity is bounded by the constant* 3*.*

Proof. The proof is by reduction from the co-SQUARE-TILING problem, where we have that $T = \{t_0, \dots, t_k\}$ is the set of tiles, the relations $H, V \subseteq T \times T$ are the *horizontal* and *vertical compatibility relations*, and the integer n is the double-exponent of the size of the square. We will construct a database D, a weakly-acyclic set of DTGDs Σ and a UCQ Q over a schema $\mathcal{R}$, such that $D \cup \Sigma \models Q$ if and only if there does not exist a tiling for the square.

The Schema $\mathcal{R}$.

- $r_k^i(\cdot), succ_k^i(\cdot,\cdot,\cdot), min_k^i(\cdot)$ and $max_k^i(\cdot)$, for each $i \in \{1,2\}$, are auxiliary relations that are used to generate two ordered sequences of double-exponentially many nulls from **N**.
- $horizontal(\cdot,\cdot)$ and $vertical(\cdot,\cdot)$ will, after generating them with the relations above, encode the two sequences of double-exponentially many nulls from **N**, that represent the horizontal and vertical index of positions on the square, respectively.
- $starth(\cdot)$ and $startv(\cdot)$ represent, respectively, the first index of the horizontal and vertical index sequence of the square, and thus the start of the $horizontal$, resp. $vertical$ sequence of nulls.

- $grid(\cdot,\cdot)$ represents a position on a grid, where the first term represents the horizontal index, and the second term represents the vertical index on the grid. Thus, the $grid$ relation encodes all the positions of the double-exponential square.
- $t(\cdot,\cdot)$, for each $t \in T$, represents the fact that a position on the grid is labelled, and thus assigned, a tile t from the set T of tiles of the given SQUARE-TILING problem.

The Database D. Our database contains the following atoms, for each $i \in \{1,2\}$, where c_0 and c_1 are constants from **C**: $r_1^i(c_0), r_1^i(c_1), min_1^i(c_0), max_1^i(c_1), succ_1^i(c_0,c_1)$. The specified atoms will be used to initialize the generation of the successor-relation which will ultimately contain double-exponentially many entries.

The Set Σ of DTGDs. The set of DTGDs contains the rules to generate a grid of double exponential size and guess which position in the grid is assigned which tile. First, we add, for each $i \in \{1,2\}$ and $k \in \{1,\dots,n\}$, the following rules, which generate an ordered sequence of double-exponentially many values:

- $r_k^i(X), r_k^i(Y) \rightarrow \exists Z\, s_k^i(X,Y,Z)$,
- $s_k^i(X,Y,Z) \rightarrow r_{k+1}^i(Z)$,
- $s_k^i(X,Y,Z), s_k^i(X,Y',Z'), succ_k^i(Y,Y') \rightarrow succ_{k+1}^i(Z,Z')$,
- $s_k^i(X,Y,Z), s_k^i(X',Y',Z'), max_k^i(Y), max_k^i(Y'), succ_k^i(X,X') \rightarrow succ_{k+1}^i(Z,Z')$,
- $s_k^i(X,X,Y), min_k^i(X) \rightarrow min_{k+1}^i(Y)$,
- $s_k^i(X,X,Y), max_k^i(X) \rightarrow max_{k+1}^i(Y)$.

The above set of rules force double-exponentially many null values from **N** to appear in the chase. In fact, for $k = n$ levels, the cross product of all the values at the previous level is computed, and for each pair introduced thusly, a new null value is generated. Therefore, the number of null values at each level grow quadratically. Over n levels, this gives us the desired result of 2^{2^n} null values in total. It is not difficult to verify that the rules above are indeed weakly-acyclic, as they are non-recursive: when constructing the dependency graph, indeed there are no cycles present, and therefore no cycles through a special edge can occur.

In order to generate our grid, we will use the following rules:

- $succ_n^1(X,Y) \rightarrow horizontal(X,Y)$,
- $min_n^1(X) \rightarrow starth(X)$,
- $succ_n^2(X,Y) \rightarrow vertical(X,Y)$,
- $min_n^2(X) \rightarrow startv(X)$,

- $horizontal(X, X'), vertical(Y, Y') \rightarrow grid(X, Y)$.

With the rules above, we use one of the double-exponential successor relations generated previously as the horizontal index, and one as the vertical index. The cross product between the two, generated with the last rule above, can then be used to access positions on our grid of double-exponential size in n.

It remains to guess which tiles are assigned to which position on the grid. To this end, we use the following rule:

- $grid(X, Y) \rightarrow \bigvee_{t \in T} t(X, Y)$.

The above rule will force the chase to explore, for each tape cell, all the options of tiles that can be assigned to it. This completes the construction of the set Σ.

The UCQ Q. Now that the construction of our rules is complete, it remains to construct a query that checks that the compatibility relations are adhered to. Recall that we are reducing from the co-problem, so we have to construct a UCQ Q that is true if and only if compatibility is violated. Roughly, Q consists of the following disjuncts:

- Q_{init} that ensures that the first cell of the grid is labelled with the tile t_0, as required by the SQUARE-TILING problem,
- $Q_{horizontal}$ that ensures that the horizontal compatibility relation is satisfied, and
- $Q_{vertical}$ that ensures that the vertical compatibility relation is satisfied.

In what follows, we will give the precise definition of these query disjuncts of Q. As we encode the co-problem, the disjuncts will actually check for the exact converse of the descriptions above.

- Q_{init} consists of the following disjuncts for each $t_i \in T$:

$$\exists X \exists Y \; starth(X) \wedge startv(Y) \wedge t_i(X, Y)$$

- $Q_{horizontal}$ consists of the following disjuncts for each $(t_i, t_j) \in (T \times T) \setminus H$:

$$\exists X \exists Y \exists Z \; t_i(X, Y) \wedge horizontal(X, Z) \wedge t_j(Z, Y)$$

- $Q_{vertical}$ consists of the following disjuncts for each $(t_i, t_j) \in (T \times T) \setminus V$:

$$\exists X \exists Y \exists Z \; t_i(X, Y) \wedge vertical(Y, Z) \wedge t_j(X, Z)$$

This concludes the construction of Q.

Correctness. The above construction is correct. Clearly, by construction, every model $M \in chase(D, \Sigma)$ represents one possible tiling of a double-exponentially sized square, by constructing a grid of the size of the square, and then guessing, via a disjunctive rule, which tile is to be assigned to which position on the grid.

Now, assume that there is a solution to the SQUARE-TILING problem. Then there must, by construction, exists a model $M \in chase(D, \Sigma)$, which represents a valid tiling of the SQUARE-TILING problem. However, a valid tiling satisfies all the horizontal and vertical compatibility constraints of H and V. But then, Q cannot be true in M, as it checks precisely that at least two adjacent tiles in the grid do not satisfy either H or V.

Conversely, assume that there does not exist a solution to the SQUARE-TILING problem. Then, every model $M \in chase(D, \Sigma)$ will represent an invalid tiling, where at least two adjacent tiles in the grid do not satisfy H or V. But then, one of the disjuncts in either $Q_{horizontal}$ or $Q_{vertical}$ will be true, as it checks precisely for such violations.

Finally, by observing that the disjuncts in Q_{init} ensures that the tile at position $(1, 1)$ of the square is labelled with tile t_0 as required, we have that $D \cup \Sigma \models Q$ if and only if SQUARE-TILING does not have a solution, as desired. □

The above result provides us with the appropriate hardness result for query answering under weakly-acyclic sets of DTGDs, when considering the combined complexity, or the case where the arity is bounded by a constant — three, in this particular case.

Let us now turn to the case where the set of DTGDs is fixed. In order to complete the complexity picture, two complexity results are still missing: a general Π_2^{P}-hardness result for the CQ-ANSWERING problem, and a CO-NP-hardness result for the CQ_1-ANSWERING problem, when also the query is fixed. The former can actually be obtained from an existing result proven by Bárány et al. [2014], where a Π_2^{P} lower bound is exhibited. They show that it is possible to encode the Π_2^{P}-hard problem of FORALL-EXISTS-SAT, with a database D, a query q and a fixed theory Σ consisting of the single rule $x(V, T, F) \rightarrow v(V, T) \oplus v(V, F)$, where $\oplus$ represents an exlcusive disjunction, also called XOR. It turns out that in our query-answering setting, the semantics of exclusive disjunctions and (normal) disjunctions coincide. We can thus reformulate the rule as $x(V, T, F) \rightarrow v(V, T) \vee v(V, F)$. This immediately gives us the following statement:

Theorem 6.8 (follows from [Bárány et al., 2014])**.** CQ-ANSWERING *under fixed weakly-acyclic sets of DTGDs is* Π_2^P*-hard.*

For the CQ_1-ANSWERING problem, we obtain hardness by the fact that the classical paper about the data complexity of query answering under Description Logic (DL) knowledge bases in [Calvanese et al., 2013, Theorem 4.5] exhibits a proof that shows CO-NP-hardness for a fixed theory Σ consisting only of the DL inclusion $A \sqsubseteq B \sqcup C$, which translates into the DTGD $a(X) \rightarrow b(X) \vee c(X)$, and a fixed ACQ q. Given that the ACQ-ANSWERING problem can be reduced to CQ_1-ANSWERING, as shown by Lemma 6.4, we immediately get the following:

Theorem 6.9. CQ_1-Answering *under weakly-acyclic sets of DTGDs is co-NP-hard in the data complexity.*

The above observations complete our complexity investigation of the query answering problem under weakly-acyclic sets of DTGDs. All relevant hardness results have been obtained to give us the complete complexity picture of the problem under each of the query langauges considered in this thesis.

6.6 Summary

In this chapter, we have investigated and completed the complexity picture of the query answering problem under weakly-acyclic sets of DTGDs. Firstly, in order to simplify the subsequent complexity investigation, we show in Lemma 6.4 that in fact in the combined complexity, the case of bounded arity, and in the data complexity it is sufficient to concentrate on one form of conjunctive query, as it is possible to find a reduction from any one type to any other. Thus, we have then proceeded to show our upper bounds for the weakest problem, namely CQ_1-Answering, and our lower bounds for the most expressive problem, namely UCQ-Answering. By Lemma 6.4, we then get that the upper (resp. lower) bound also holds for the most expressive (resp. weakest) problem. We thus found that in the combined complexity and bounded arity cases, the problem of query answering, irrespective of the query language considered, is complete for the complexity class co-N2ExpTime. In the data complexity, the same holds true for the complexity class co-NP.

The only case where the above approach does not work is in the case where the set of DTGDs, but not the query, is fixed. In this case, the result of Lemma 6.4 does not hold, and we proceeded to do a separate complexity investigation for the different types of queries that we treat in this thesis. In fact, it turns out that all query types, except for full, arbitrary (U)CQs, the query answering problem remains co-NP-complete. However, the additional expressive power that a full, arbitrary CQ provides is enough to push the complexity of the (U)CQ-Answering problem one level up the polynomial hierarchy, and we find that the problem is in fact complete for the complexity class Π_2^p.

In conclusion, we can say that the impact of disjunction on weakly-acyclic sets of rules is as follows: in both the combined and the data complexity, we find a very intuitive jump from the respective deterministic complexity class in case of (non-disjunctive) TGDs, to the corresponding non-deterministic complexity class in case of DTGDs: in the combined complexity we have a jump from 2ExpTime-completeness to co-N2ExpTime-completeness, and the analogous result holds true for the data complexity: we find a jump from PTime-completeness to co-NP-completeness. We can thus say that the non-determinism added by allowing disjunctions in rule heads is reflected in the complexity of the query answering problem.

7 Applications to Description Logics

In the previous three chapters, we have discussed, and analyzed the complexity of query answering for guarded-based classes of DTGDs, sticky sets of DTGDs and weakly-acyclic sets of DTGDs. In this chapter, we explore the consequences of our previous results, when applied to existing formalisms in the area of description logics (DLs). We will investigate consequences of our results for DLs in Section 7.2, when dealing with the problem of answering conjunctive queries.

In order to express these formalisms with DTGDs, in Section 7.1, we will first extend DTGDs with two auxiliary features. Firstly, we introduce *negative constraints (NCs)* that allow us to express that some condition is forbidden and must not occur during construction of the chase. Secondly, we introduce the notion of *equality-generating dependencies (EGDs)* that allow for expressing equalities between variables.

7.1 Additional Features

In this section, we extend our previously introduced classes of DTGDs with two useful and relevant constructs, namely, negative constraints and equality-generating dependencies. None of the classes of DTGDs discussed so far are able to express natural statements like "no student is a professor," or "a student ID uniquely identifies a student's name." Assume a schema which has a unary relation $professor$ and a binary relation $student$, where the first attribute is the student ID, and the second attribute is the student's name. The first statement above could be expressed by the rule $student(X, Y), professor(X) \rightarrow \bot$, known as a negative constraint, where $\bot$ is the first-order logic constant *false*; the second statement above would be represented, for example, by the rule $student(X, Y), X, Y'(\rightarrow) Y = Y'$, known as an equality-generating dependency. The formal definitions of these two notions can be found in the next two sections, respectively.

7.1.1 Negative Constraints

As seen in the example given in the previous paragraph, a negative constraint is similar to a TGD, but differs in that the head is always the constant $\bot$, or, *false*. The formal definition is given below:

Definition 7.1. *A* negative constraint (NC) *over a schema $\mathcal{R}$ is a first-order logic formula of the form*

$$\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot,$$

where $\phi(\mathbf{X})$ is a conjunction of atoms over $\mathcal{R}$ called the body *with free variables* $\mathbf{X}$*, and $\bot$ is the Boolean constant* false*. The body of an NC ν is denoted $body(\nu)$.*

Note that the above definition does not place any kind of syntactic restriction on the conjunction of atoms in the body of such an NC ν. It is thus possible for ν to contain a full conjunctive query. For brevity, we will omit the universal quantifier in front a any NC, and implicitly assume that all the variables are universally quantified.

Similarly to (D)TGDs, an NC ν is satisfied by an instance I over $\mathcal{R}$, denoted $I \vDash \nu$, if and only if there is no homomorphism h, such that $h(body(\nu)) \subseteq I$. Conversely, we write $I \nvDash \nu$, if I does not satisfy ν. This notation extends naturally to sets of NCs. Note that we only consider finite sets of NCs.

Example 7.2. *Let $\mathcal{R}$ be a schema where a binary relation $dept$ contains departments and their heads, and a binary relation $projmgr$ contains projects and their managers. Then the fact that department heads cannot be project managers can be expressed by the NC*

$$dept(D, H), projmgr(P, H) \rightarrow \bot.$$

The fact that Peter cannot be the head of any department can be expressed using the NC

$$dept(D, peter) \rightarrow \bot.$$

Note that both constants and variables can be used in the body of an NC.

The models of a database w.r.t. a set of DTGDs and NCs is defined in exactly the same way as it is for DTGDs alone. Formally, the *models* of a database D w.r.t. a set $\Sigma = \Sigma_R \uplus \Sigma_\bot$, where Σ_R is a set of DTGDs and $\Sigma_\bot$ is a set of NCs, denoted $\vDash D\Sigma$, is the set of all finite or infinite instances I, such that $I \supseteq D$, $I \vDash \Sigma_R$ and $I \vDash \Sigma_\bot$, denoted $I \vDash D \cup \Sigma$. In [Calì et al., 2012] it is observed that, when reasoning under a set of TGDs, checking whether an NC is satisfied amounts to answering an (arbitrary) conjunctive query. It turns out that this observation also holds true for reasoning under sets of DTGDs, as the following theorem shows:

Theorem 7.3. *For any database D and UCQ Q, given a set of rules $\Sigma = \Sigma_R \uplus \Sigma_\bot$ over schema $\mathcal{R}$, where Σ_R is a set of DTGDs and $\Sigma_\bot$ is a set of NCs, we can construct a UCQ Q', such that $D \cup \Sigma \vDash Q$ if and only if $D \cup \Sigma_R \vDash Q'$.*

Proof. The reduction is rather simple: we only need to treat all the negative constraints as disjuncts of a UCQ. Thus, let

$$Q' = Q \vee \bigvee_{\nu \in \Sigma_\bot} body(\nu).$$

This completes the construction. In order to prove correctness, we need to establish that $D \cup \Sigma \vDash Q$ if and only if $D \cup \Sigma_R \vDash Q'$. The following vital observation will help to show this:

Note that $models(D, \Sigma_R)$ will necessarily be a subset of $models(D, \Sigma)$. Any model M of $D \cup \Sigma$ will surely be a model of $D \cup \Sigma_R$, as, by definition, $M \supseteq D$ and $M \vDash \Sigma_R$. For all such models M,

it is clear that $M \models Q$ and thus also $M \models Q'$, as $Q' \supseteq Q$. An instance I that is a model of $D \cup \Sigma_R$ but not a model of $D \cup \Sigma$ thus must necessarily violate at least one negative constraint in $\Sigma_\perp$.

Assume that $D \cup \Sigma \models Q$. For all $M \in models(D, \Sigma_R)$ it must thus hold that $M \models \Sigma_\perp$ and $M \models Q$ (as the query is by assumption true in all models), or otherwise that $M \not\models \Sigma_\perp$ (and M thus is not a model of $D \cup \Sigma$. For the first case, by the above observation, we have that $M \models Q'$, as desired. In the second case, we also have that $M \models Q'$, as for some $\nu \in \Sigma_\perp$ it holds that $M \not\models \nu$, that is, there is a homomorphism that maps $body(\nu)$ into M. But then, by construction, the corresponding disjunct of Q' is can be so mapped into M. Conversely, assume that $D \cup \Sigma_R \models Q'$. Then by a similar argument, we can show that a model M of $D \cup \Sigma_R$ either satisfies a disjunct in Q or a body of an NC. We thus have, as desired, that if Q is true in every model of $D \cup \Sigma$, then and only then is Q' true in every model of $D \cup \Sigma'$. This completes the proof. □

Note that the above proof does not rely on the chase procedure but deals with models directly. In order to utilize the chase when the theory contains NCs, we need to make a small addition to the definition of the chase.

The definition of the disjunctive chase tree remains the same as for the classical disjunctive chase; however, the definition of $chase(D, \Sigma)$ changes slightly: for a disjunctive chase tree T, let $chase(D, \Sigma)$ be the set $\{I \mid I \text{ is a leaf of } T \text{ and for all } \nu \in \Sigma_\perp, I \models \nu\}$. Again, by construction, each instance in $chase(D, \Sigma)$ is a model of D and Σ, and this definition preserves the universal model set property of the chase. For query answering purposes, we can thus still concentrate on the models in the universal model set as computed by the chase.

As an immediate consequence of the above discussion and Theorem 7.3, we obtain that the addition of negative constraints does not increase the complexity of answering arbitrary UCQs. By Lemmas 4.6, 5.3 and 6.4, we get immediately that this also holds for CQs. In fact, for (weakly-)frontier-guarded, sticky and weakly-acyclic sets of DTGDs, the addition of negative constraints do not even increase the complexity of the CQ_1-ANSWERING problem. Thus, in these cases, the complexity of query answering is the same, whether NCs are present in the theory or not.

7.1.2 Equality-Generating Dependencies

As we have seen above, allowing NCs in addition to DTGDs does not cause any problems, and the complexity of the query answering problem is in general not affected. Unfortunately, the same cannot be said for the addition of equality-generating dependencies. These have been shown to lead to undecidability, even if one only considers the simple case of (non-disjunctive) inclusion dependencies and functional dependencies (see Chandra and Vardi [1985]), or inclusion dependencies and key dependencies (see, e.g. Calì et al. [2003a], which shows undecidability by using a similar machinery to the one employed by Johnson and Klug [1984] in their PSPACE-hardness proof for query containment under IDs). These undecidability results prevent us from simply introducing equality-generating dependencies into the decidable classes

of DTGDs discussed in the previous chapters. However, as we shall see, there are useful and sensible restrictions on these dependencies that can guarantee decidability. Before discussing these, let us give some more formal details.

Definition 7.4. *An* equality-generating dependency (EGD) *over a schema $\mathcal{R}$ is a first-order logic formula of the form*

$$\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow X_i = X_j,$$

where $\phi(\mathbf{X})$ is a conjunction of atoms over $\mathcal{R}$, called the body of the EGD, with free variables $\mathbf{X}$, and $X_i, X_j \in \mathbf{X}$. The body of an EGD ϵ is denoted $body(\epsilon)$.

As with DTGDs and NCs, we will, for brevity, omit the universal quantifiers, and implicitly assume that all unquantified variables are universally quantified. An instance I satisfies an EGD ϵ, denoted $I \models \epsilon$, if and only if there exists a homomorphism h, such that whenever $h(body(\epsilon)) \subseteq I$, then $h(X_i) = h(X_j)$. We write $I \not\models \epsilon$, if I does not satisfy ϵ. Again, these notions naturally extend to sets. Recall that while homomorphisms can map variables or null values to other values, constants are always preserved. Thus, the only way for an instance I to violate an EGD ϵ is if there is a homomorphism h that maps $body(\epsilon)$ to I, $h(X_i) = c_1$ and $h(X_j) = c_2$, where c_1 and c_2 are distinct constants from $\mathbf{C}$. As with NCs, we only deal with finite sets of EGDs in this work.

Example 7.5. *Let $\mathcal{R}$ be a schema containing a binary relation $dept$ which contains departments and their heads, a binary relation $runs$ which contains departments and the projects they run, and a binary relation $projmgr$ which contains projects and their managers. Then the fact that only department heads can be managers of projects run by their department can be expressed using an EGD as follows:*

$$\epsilon : dept(D, H), runs(D, P), projmgr(P, M) \rightarrow H = M.$$

Given the database

$$D = \{dept(it, emp1), runs(it, support), projmgr(support, emp1)\},$$

it is easy to verify that $D \models \epsilon$. However, if a fact $projmgr(support, emp2)$ to D, thereby forming D', then we can find a homomorphism h that maps variable H to $emp1$ and variable M to $emp2$, while mapping the body of ϵ to D'. Thus, $D' \not\models \epsilon$.

The models of a database w.r.t. a set of DTGDs and EGDs is defined in exactly the same way as it is for DTGDs alone. Formally, the *models* of a database D w.r.t. a set $\Sigma = \Sigma_R \uplus \Sigma_E$, where Σ_R is a set of DTGDs and Σ_E is a set of EGDs, denoted $\models D\Sigma$, is the set of all finite or infinite instances I, such that $I \supseteq D$, $I \models \Sigma_R$ and $I \models \Sigma_\perp$, denoted $I \models D \cup \Sigma$. Based on this, the query answering problem (under the certain answer semantics) can be naturally extended to sets of DTGDs and EGDs.

As was the case for sets of DTGDs, our main algorithmic tool for query answering under DTGDs and EGDs is the (disjunctive) chase. In order for the chase to be able to work correctly, we need a separate chase rule for EGDs that unifies symbols in the chase in order to "repair" the instance so that it satisfies violated EGDs.

Definition 7.6 (EGD Chase Rule)**.** *Consider an instance I, and an EGD ϵ of the form $\phi(\mathbf{X}) \rightarrow X_i = X_j$. ϵ is* applicable *to I if there exists a homomorphism h such that $h(\phi(\mathbf{X})) \subseteq I$ and $h(X_i) \neq h(X_j)$, and the result of applying σ to I with h is the set $\{I'\}$, where (a) if $h(X_i)$ and $h(X_j)$ are both constants, then $I' = \{\bot\}$, and otherwise (b) I' is obtained from I by replacing all occurrences of $h(X_i)$ with $h(X_j)$, if $h(X_i)$ precedes $h(X_j)$ in the lexicographical order, or vice-versa otherwise. For such an application, which defines a single* chase step*, we write $I\langle\epsilon, h\rangle\{I'\}$.*

With this definition in place, we can now proceed to redefine the chase. In fact, the chase is constructed by executing the following two steps iteratively, until a fixpoint is reached:

1. Apply the DTGD chase rule, as defined in Section 2.3, once; and then
2. apply the EGD chase rule as long as it is applicable (i.e. until a fixpoint is reached).

The chase is thus constructed by always applying EGDs as often as possible after each application of a DTGD, in order to catch inconsistencies as early as possible, and avoid the degenerate case where in an infinite chase expansion no EGD may ever be applied.

The definition of the disjunctive chase tree remains the same as for the classical disjunctive chase; however, the definition of $chase(D, \Sigma)$ changes slightly: for a disjunctive chase tree T, let $chase(D, \Sigma)$ be the set $\{I \mid I \text{ is a leaf of } T \text{ and } I \neq \{\bot\}\}$. Again, by construction, each instance in $chase(D, \Sigma)$ is a model of D and Σ, and this definition preserves the universal model set property of the chase. For query answering purposes, we can thus still concentrate on the models in the universal model set as computed by the chase. Note that if the chase is empty (i.e. all constructed instances violate an EGD), we also say that the chase *fails*. It is straightforward to see that EGDs and NCs can also be used together, by simply combining the relevant modifications of the chase.

As discussed before, allowing EGDs in addition to DTGDs (or even TGDs) can easily lead to undecidability. An abstract restriction that has been used in the literature to overcome this problem is the notion of *separability*. First proposed by Calì et al. [2003a] for inclusion dependencies and functional dependencies, it was later extended to cover arbitrary TGDs; cf. Calì et al. [2012]. While this was done for deterministic sets of rules, it is straightforward to extend the notion to also work for DTGDs:

Definition 7.7 (Separability)**.** *Let $\Sigma = \Sigma_R \uplus \Sigma_E \uplus \Sigma_\bot$, where Σ_R is a set of DTGDs and Σ_E is a set of EGDs, all over a schema $\mathcal{R}$. Σ is* separable*, if for every database D over $\mathcal{R}$ and every CQ q over $\mathcal{R}$, we have either that $chase(D, \Sigma) = \emptyset$ (i.e. the chase fails) or that $chase(D, \Sigma) \models q$ if and only if $chase(D, \Sigma_R) \models q$.*

In fact, the notion of separability guarantees that the problem of answering a CQ over a set of DTGDs and EGDs has the same complexity as answering CQs over DTGDs alone. This even holds true if NCs are allowed into the mix as well. While in [Pieris, 2011] this has only been shown for frontier-weakly-sticky-join sets of TGDs, it is implicit that this in fact holds for any class of TGDs. In fact, with the definition of failure of the chase being emptiness, that is, $chase(D, \Sigma) = \emptyset$, the same proof as the one given in [Pieris, 2011] also works to establish the complexity equivalence for arbitrary sets of DTGDs, implicitly showing the following statement:

Theorem 7.8. *Let q be a CQ over a schema $\mathcal{R}$, D a database over $\mathcal{R}$, a set $\Sigma = \Sigma_R \uplus \Sigma_E$ of rules over $\mathcal{R}$, where Σ_R is a set of DTGDs, Σ_E a set of EGDs and a set $\Sigma_\perp$ of NCs. Then, if Σ is separable, deciding whether $D \cup \Sigma \cup \Sigma_\perp \models q$ has the same complexity as deciding whether $D \cup \Sigma_R \cup \Sigma_\perp \models q$.*

The proof given in [Pieris, 2011] works by reducing the problem of deciding whether the chase under a set of DTGDs and EGDs fails to query answering under the set of DTGDs alone. This is done by rewriting the EGDs into a disjunction of CQs, with the help of a non-equivalence relation added to the database. Unfortunately however, it is in general undecidable whether a given set of DTGDs and EGDs is separable.

This concludes the present section about additional features that we can add to our discussed classes of DTGDs without impacting the complexity of the query answering problem. In the next section, we will see how we can use these features to encode several interesting reasoning formalisms using DTGDs, NCs and EGDs, and thus derive (partially novel) complexity bounds for them.

7.2 Querying Description Logic Knowledge Bases

Ontological reasoning is a fundamental task in the Semantic Web. In this field, description logics (DLs) (see, e.g. [Baader et al., 2003] for a comprehensive overview) have been playing a prominent role. Most DLs are decidable fragments of first-order logic, based on concepts (classes of objects) and roles (binary relations on concepts). Several variants of DLs have been proposed, where a central issue is the trade-off between the expressive power and the computational complexity of the reasoning services such as query answering. Our goal in this section is to show that our techniques and results on guarded DTGDs can be used as a generic tool for establishing results on query answering under several central DLs that can be found in the literature. We will first give a brief overview of the world of DLs, before we start our investigation.

Brief Overview. One of the most expressive DLs is the description logic $\mathcal{SROIQ}$, proposed by Horrocks et al. [2006], which forms the basis of the standardized specification of the second version of the Web Ontology Language (OWL 2). Reasoning in $\mathcal{SROIQ}$ turns out to be

very difficult and thus very computationally expensive, which lead to the invention of description logics that offer better computational properties. Another well-known DL that allows for non-determinism is the simple, yet expressive DL $\mathcal{ALC}$, or *Attributive Concept Language with Complements*, a key DL that forms the basis of many other DLs, be they more or less expressive.

The $\mathcal{EL}$ family of DLs [Baader, 2003; Baader et al., 2005] is based on the lightweight DL $\mathcal{EL}$, which is a DL designed to allow for the most common DL constructs, namely existential quantification and conjunction to be expressed, but to still retain favorable computational properties. In fact, reasoning under $\mathcal{EL}$ is in PTIME for most tasks. In our case we will deal with the DL $\mathcal{ELU}$, which is $\mathcal{EL}$ extended with union ($\mathcal{U}$), which thereby allows for non-determinism. We will also consider the addition of inverse roles ($\mathcal{I}$) and role hierarchies ($\mathcal{H}$).

The *DL-Lite* family (see, e.g. Calvanese et al. [2007]; Poggi et al. [2008]) is a group of DLs designed to exhibit favorable computational properties. In particular, query answering under most deterministic *DL-Lite* variants in in AC_0 in the data complexity, which follows from the fact that they are first-order rewritable. In [Artale et al., 2009], several extensions are proposed that include number restrictions, role hierarchies, and (more) boolean connectives. In this paper, the language of *DL-Lite*$_{bool}$ is proposed, which allows for disjunctive knowledge to be represented. Together with role hierarchies ($\mathcal{H}$), we get the language of *DL-Lite*$^{H}_{bool}$, which, among others, we will take a closer look at in this section.

With this brief overview of DL languages in mind, let us now formally investigate how the results obtained in previous chapters can be used to pinpoint the complexity of query answering under several description logics for which the answers to these questions were hitherto missing, or at least incomplete.

7.2.1 The Description Logic $\mathcal{ALCIH}$

The DL $\mathcal{ALC}$ is a central DL, introduced by Schmidt-Schauß and Smolka [1991], which is at the basis of many expressive DLs, and is the "smallest" one that is propositionally closed, i.e., that allows for all Boolean connectives. By enriching $\mathcal{ALC}$ with inverse roles ($\mathcal{I}$) and role hierarchies ($\mathcal{H}$) we obtain the expressive DL $\mathcal{ALCIH}$. Let $\mathbf{N_C}$ and $\mathbf{N_R}$ be disjoint countably infinite sets of concept and role names, respectively. The set of $\mathcal{ALCIH}$-concepts is the smallest set such that

1. every concept name $A \in \mathbf{N_C}$, as well as $\top$ and $\bot$, are $\mathcal{ALCIH}$-concepts; and
2. if C and D are $\mathcal{ALCIH}$-concepts and $R \in \mathbf{N_R}$, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\exists R^{-}.C$, $\forall R.C$ and $\forall R^{-}.C$ are $\mathcal{ALCIH}$-concepts.

A DL *knowledge base (KB)* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ represents the domain of interest in terms of two parts, a *terminological box (TBox)* $\mathcal{T}$, specifying the intensional knowledge, and an *assertional box (ABox)* $\mathcal{A}$, asserting the extensional knowledge. In an $\mathcal{ALCIH}$ KB the TBox is a finite set of concept inclusion assertions of the form $B \sqsubseteq C$, where B and C are $\mathcal{ALCIH}$-concepts, as

well as a finite set of role inclusion assertions of the form $R \sqsubseteq S$, where R and S are (possibly inverse) roles. The ABox consists of a finite set of assertions of the form $C(a)$ and $R(a,b)$, where a and b are individuals (or constants) of $\mathbf{C}$, C is an $\mathcal{ALCIH}$-concept and R is a role name.

We now recall the semantics of $\mathcal{ALCIH}$. An $\mathcal{ALCIH}$-interpretation $\mathcal{I}$ is a pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta_\mathcal{I}$ is a non-empty set called *domain*, and $\cdot^\mathcal{I}$ is an *interpretation function* that maps every concept name A to a subset $A^\mathcal{I}$ of $\Delta_\mathcal{I}$, and every role name R to a binary relation $R^\mathcal{I}$ of $\Delta_\mathcal{I}$. $\mathcal{I}$ is extended to complex concepts as follows:

$$
\begin{aligned}
(\top)^\mathcal{I} &:= \Delta_\mathcal{I} \\
(\bot)^\mathcal{I} &:= \emptyset \\
(\neg C)^\mathcal{I} &:= \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(B \sqcap C)^\mathcal{I} &:= B^\mathcal{I} \cap C^\mathcal{I} \\
(B \sqcup C)^\mathcal{I} &:= B^\mathcal{I} \cup C^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &:= \{x \mid \text{there exists } y \in \Delta_\mathcal{I} \text{ such that } (x,y) \in R^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \\
(\exists R^-.C)^\mathcal{I} &:= \{x \mid \text{there exists } y \in \Delta_\mathcal{I} \text{ such that } (y,x) \in R^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \\
(\forall R.C)^\mathcal{I} &:= \{x \mid \text{for all } y \in \Delta_\mathcal{I}, (x,y) \in R^\mathcal{I} \text{ implies } y \in C^\mathcal{I}\} \\
(\forall R^-.C)^\mathcal{I} &:= \{x \mid \text{for all } y \in \Delta_\mathcal{I}, (y,x) \in R^\mathcal{I} \text{ implies } y \in C^\mathcal{I}\}.
\end{aligned}
$$

An interpretation $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ if $B^\mathcal{I} \subseteq C^\mathcal{I}$ for all $B \sqsubseteq C \in \mathcal{T}$. An interpretation $\mathcal{I}$ can be extended to ABoxes by demanding that $\cdot^\mathcal{I}$ maps every individual a to an element $a^\mathcal{I} \in \Delta_\mathcal{I}$. $\mathcal{I}$ satisfies an assertion $C(a)$ if $a^\mathcal{I} \in C^\mathcal{I}$ and an assertion $R(a,b)$ if $(a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$. We say that $\mathcal{I}$ is a model of an ABox $\mathcal{A}$ if it satisfies all the assertions of $\mathcal{A}$. Given an $\mathcal{ALCIH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, $\mathcal{I}$ is a model of $\mathcal{K}$, written $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{A}$. It is now straightforward to define the problem of query answering under $\mathcal{ALCIH}$ KBs. A CQ q over an $\mathcal{ALCIH}$ KB $\mathcal{K}$ is a conjunction of atoms of the form $A(X)$ or $R(X,Y)$, where $A \in \mathbf{N_C}$ and $R \in \mathbf{N_R}$. The answer to q is positive, written $\mathcal{K} \models q$, if $\mathcal{I} \models q$, for every model $\mathcal{I}$ of $\mathcal{K}$. The UCQ, ACQ and CQ_1 answering problems are defined analogously.

7.2.2 Generic Complexity Results

By exploiting our techniques and results, we can establish the following result about (U)CQ answering; $\mathcal{L}_1 \subseteq \mathcal{L}_2$ means that $\mathcal{L}_1$ is a subformalism of $\mathcal{L}_2$:

Theorem 7.9. *Consider a DL $\mathcal{L}$ such that $\mathcal{L} \subseteq \mathcal{ALCIH}$ and $\mathcal{L}$ is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^-.\top \sqsubseteq A \qquad R \sqsubseteq S \qquad R^- \sqsubseteq S,$$

where A, A_1, A_2, A_3 are concept names and R, S are role names. Then, (U)CQ answering under $\mathcal{L}$ is 2ExpTime-complete in combined complexity.

An analogous result for UACQ answering can be also established:

Theorem 7.10. *Consider a DL $\mathcal{L}$ such that $\mathcal{L} \subseteq \mathcal{ALCIH}$ and $\mathcal{L}$ is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^{-}.\top \sqsubseteq A,$$

where A, A_1, A_2, A_3 are concept names. Then, UACQ answering under $\mathcal{L}$ is ExpTime-complete in combined complexity, even when the TBox is fixed.

The above generic results provide alternative proofs for already known complexity results, but also close some interesting open problems in a uniform way. The rest of this section is devoted to establish the above two theorems.

Upper Bounds

The upper bounds for both theorems are obtained by reducing query answering under the DL $\mathcal{ALCIH}$ to query answering under guarded DTGDs. First, we discuss how complex concepts can be eliminated from the given ABox by pushing them in the TBox. Such a reformulation of the given KB will allow us to consider the ABox as a relational database. Consider an $\mathcal{ALCIH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. We construct an ABox $\mathcal{A}'$ from $\mathcal{A}$ by replacing each assertion $C(a)$, where C is a complex concept, by $C^{\star}(a)$, where $C^{\star}$ is an auxiliary concept name not occurring in $\mathcal{A}$. Then, the TBox $\mathcal{T}'$ is constructed by adding to $\mathcal{T}$ an assertion of the form $C^{\star} \sqsubseteq C$, for each complex concept C occurring in $\mathcal{A}$. It is easy to see that $(\mathcal{T}, \mathcal{A})$ and $(\mathcal{T}', \mathcal{A}')$ are equivalent w.r.t. UCQ answering. Let us now explain how an $\mathcal{ALCIH}$ TBox can be normalized in such a way that can be seen as a set of guarded DTGDs. In fact, we exploit the normalization procedure proposed by Simancik et al. [2011a] for $\mathcal{ALCH}$, that is, $\mathcal{ALCIH}$ without inverse roles. An $\mathcal{ALCIH}$ TBox is in normal form if it only contains axioms of the form given on the left column of Table 7.1. It is implicit in [Simancik et al., 2011a] that every $\mathcal{ALCH}$ TBox $\mathcal{T}$ can be transformed in polynomial time into an $\mathcal{ALCH}$ TBox in normal form which is equivalent to $\mathcal{T}$ w.r.t. UCQ answering. This result can be straightforwardly extended to $\mathcal{ALCIH}$. From the above discussion, we immediately get the following auxiliary result:

Lemma 7.11. *Consider an $\mathcal{ALCIH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. We can construct in polynomial time an $\mathcal{ALCIH}$ KB $\mathcal{K}' = (\mathcal{T}', \mathcal{A}')$ such that $\mathcal{A}'$ contains only concept and role names, and $\mathcal{K} \models Q$ if and only if $\mathcal{K}' \models Q$, for every UCQ Q.*

As shown in Table 7.1, given an $\mathcal{ALCIH}$ TBox $\mathcal{T}$ in normal form, every axiom of $\mathcal{T}$ can be equivalently rewritten as a first-order sentence which is either a guarded DTGD or a constraint of the form $\forall X(A_1(X) \wedge \ldots \wedge A_n(X) \rightarrow \bot)$, where $\bot$ denotes the truth constant *false*, which may lead to an inconsistency. Such inconsistencies can be encoded in the query by considering the left-hand side of the constraints as disjuncts of the given UCQ. Moreover, observe that these disjuncts will be ACQs, and thus, if the given UCQ is acyclic, then the obtained UCQ remains

$\mathcal{ALCIH}$ Axiom	First-Order Representation
$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \bot$	$\forall X(A_1(X) \wedge \ldots \wedge A_n(X) \rightarrow \bot)$
$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B_1 \sqcup \ldots \sqcup B_m$	$\forall X(A_1(X) \wedge \ldots \wedge A_n(X) \rightarrow B_1(X) \vee \ldots \vee B_m(X))$
$A \sqsubseteq \exists R.B$	$\forall X(A(X) \rightarrow \exists Y R(X,Y) \wedge B(Y))$
$A \sqsubseteq \exists R^-.B$	$\forall X(A(X) \rightarrow \exists Y R(Y,X) \wedge B(Y))$
$\exists R.A \sqsubseteq B$	$\forall X(R(X,Y) \wedge A(Y) \rightarrow B(X))$
$\exists R^-.A \sqsubseteq B$	$\forall X(R(Y,X) \wedge A(Y) \rightarrow B(X))$
$A \sqsubseteq \forall R.B$	$\forall X(A(X) \wedge R(X,Y) \rightarrow B(Y))$
$A \sqsubseteq \forall R^-.B$	$\forall X(A(X) \wedge R(Y,X) \rightarrow B(Y))$
$R \sqsubseteq S$	$\forall X \forall Y(R(X,Y) \rightarrow S(X,Y))$
$R^- \sqsubseteq S$	$\forall X \forall Y(R(Y,X) \rightarrow S(X,Y))$
$R \sqsubseteq S^-$	$\forall X \forall Y(R(X,Y) \rightarrow S(Y,X))$
$R^- \sqsubseteq S^-$	$\forall X \forall Y(R(Y,X) \rightarrow S(Y,X))$

Table 7.1: Normal form and first-order representation of $\mathcal{ALCIH}$.

acyclic. Now, an ABox which contains only concept and role names (and not complex concepts) can be naturally seen as a relational database. Therefore, from Lemma 7.11 and the above discussion, we conclude the following: given an $\mathcal{ALCIH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and an U(A)CQ Q, we can construct in polynomial time a database $D_{\mathcal{A}}$, a set $\Sigma_{\mathcal{T}}$ of guarded DTGDs, and an (A)UCQ $Q_{\mathcal{T}}$ such that $\mathcal{K} \models Q$ if and only if $D_{\mathcal{A}} \cup \Sigma_{\mathcal{T}} \models Q_{\mathcal{T}}$, and the next result follows:

Proposition 7.12. *U(A)CQ answering under $\mathcal{ALCIH}$ KBs can be polynomially reduced to answering U(A)CQs under guarded DTGDs.*

Clearly, the above proposition, combined with our results on answering U(A)CQs under guarded DTGDs, implies the upper bounds stated in Theorems 7.9 and 7.10.

Lower Bounds

The desired lower bounds are inherited from Theorems 4.16 and 4.25. In fact, the sets of DIDs employed in the proofs of the above results can be equivalently rewritten as DL axioms.

We first focus on the proof of Theorem 4.16. The DIDs employed in this proof have one of the following forms which corresponds to a DL axiom:

$$\begin{aligned} A(X) \rightarrow B_1(X) \vee \ldots \vee B_n(X) &\equiv A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n \\ A(X) \rightarrow \exists Y\, R(X,Y) &\equiv A \sqsubseteq \exists R.\top \\ R(X,Y) \rightarrow A(Y) &\equiv \exists R^-.\top \sqsubseteq A \\ R(X,Y) \rightarrow S(X,Y) &\equiv R \sqsubseteq S \\ R(X,Y) \rightarrow S(Y,X) &\equiv R^- \sqsubseteq S. \end{aligned}$$

Clearly, an axiom of the form $A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n$ can be rewritten as follows:

$$\begin{aligned} A &\sqsubseteq B_1 \sqcup B_{2..n} \\ B_{2..n} &\sqsubseteq B_2 \sqcup B_{3..n} \\ &\vdots \\ B_{(n-2)..n} &\sqsubseteq B_{n-2} \sqcup B_{(n-1)n} \\ B_{(n-1)n} &\sqsubseteq B_{n-1} \sqcup B_n, \end{aligned}$$

and the next result follows.

Proposition 7.13. *Consider a DL $\mathcal{L}$ which is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^-.\top \sqsubseteq A \qquad R \sqsubseteq S \qquad R^- \sqsubseteq S,$$

where A, A_1, A_2, A_3 are concept names and R, S are role names. Then, (U)CQ answering under $\mathcal{L}$ is 2ExpTime-hard in combined complexity.

Let us now focus on the proof of Theorem 4.25. The DIDs employed in this proof have one of the following forms which corresponds to DL axioms (possibly more than one):

$$\begin{aligned} A(X) \rightarrow \exists Y\, R(X,Y) &\equiv A \sqsubseteq \exists R.\top \\ R(X,Y) \rightarrow A(Y) &\equiv \exists R^-.\top \sqsubseteq A \\ R(X,Y) \rightarrow \exists Z\, S(Y,Z) \vee A(Y) &\equiv \exists R^-.\top \sqsubseteq B_1 \quad B_1 \sqsubseteq B_2 \sqcup A \quad B_2 \sqsubseteq \exists S.\top \\ R(X,Y) \rightarrow A_1(Y) \vee A_2(Y) \vee A_3(Y) &\equiv \exists R^-.\top \sqsubseteq B \quad B \sqsubseteq A_1 \sqcup A_{23} \quad A_{23} \sqsubseteq A_2 \sqcup A_3, \end{aligned}$$

and the next result follows.

Proposition 7.14. *Consider a DL $\mathcal{L}$ which is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^-.\top \sqsubseteq A,$$

where A, A_1, A_2, A_3 are concept names. Then, AUCQ answering under $\mathcal{L}$ is ExpTime-hard in combined complexity, even when the TBox is fixed.

7.2.3 A Brief Case Study

We conclude this section by briefly discussing which existing DLs benefit from our generic Theorems 7.9 and 7.10. Clearly, $\mathcal{ALCIH}$ itself is one of those DLs since is powerful enough to express all the axioms listed in our theorems. Other DLs are *DL-Lite*$^{H}_{bool}$ [Artale et al., 2009], one of the most expressive languages of the DL-Lite family, $\mathcal{ELU}$ (cf. Baader et al. [2005]), that is, the extension of the well-known DL $\mathcal{EL}$ with union of concepts, and $\mathcal{ELUIH}$, that is, the extension of $\mathcal{ELU}$ with inverse roles and role hierarchies. We thus have the following:

Corollary 7.15. (U)CQ-ANSWERING *under $\mathcal{ALCIH}$ TBoxes, DL-Lite$^{H}_{bool}$ TBoxes and $\mathcal{ELUIH}$ TBoxes is 2EXPTIME-complete in the combined complexity.*

The 2EXPTIME-completeness of (U)CQ answering under $\mathcal{ALCIH}$ is actually known; cf. Lutz [2008]. However, the EXPTIME-completeness of UACQ answering under $\mathcal{ALCIH}$ (even in the case of a fixed TBox), as well as all the inherited results regarding 2EXPTIME-completeness for *DL-Lite*$^{H}_{bool}$ and $\mathcal{ELUIH}$, are novel. In fact, our 2EXPTIME-hardness proof provides an interesting alternative to the proof in [Lutz, 2008].

Regarding the problem of answering acyclic queries under DL knowledge bases, it can be seen rather straightforwardly that in the combined complexity we get a reduction to EXPTIME-completeness. In the DL world this result is well-known (see, e.g. Meier and Schneider [2013]), as an acyclic query can be seen as a (complex, non-atomic) concept, and the problem can thus be interpreted as concept (un-)satisfiability. This is also reflected in Theorem 7.10, which represents the case of answering queries under sets of DTGDs with fixed arity. However, we can also infer some new results. In fact, our theorem holds even for the case when the TBox is fixed. In particular, as stated in our theorem, the only constructs needed are a fixed set of disjunctions, limited existential quantifications and inverse roles. These constructs can of course be found in $\mathcal{ALCIH}$, but the result also holds for $\mathcal{ALCI}$, as role hierarchies are not required for our proof. The same holds true for *DL-Lite*$^{H}_{bool}$ and *DL-Lite*$_{bool}$, as well as $\mathcal{ELU}$ and its extension $\mathcal{ELUIH}$. We thus have the following:

Corollary 7.16. UACQ-ANSWERING *under $\mathcal{ALCIH}$, $\mathcal{ALCI}$, DL-Lite$^{H}_{bool}$, DL-Lite$_{bool}$, $\mathcal{ELU}$ and $\mathcal{ELUIH}$ TBoxes is EXPTIME-complete, even in case where the TBox is fixed.*

Finally, when also allowing for the presence of EGDs, we can also treat certain DLs with functionality assertions ($\mathcal{F}$) of the form funct$R$, which state that R (which may also be an inverse role) is functional. Such axiom can be translated into EGDs of the form $p_R(X, Y), p_R(X, Z) \rightarrow Y = Z$. With this translation, and by Theorem 7.8, Corollary 7.16 can be extended to treat DLs that are a sub-language of $\mathcal{ALCIHF}$, for which it is known that the set of DTGDs and EGDs Σ obtained from such a translation is separable (cf. Definition 7.7). One such DL is *DL-Lite*$^{F}_{bool}$. In fact this is already implicit in [Pieris, 2011, Lemma 8.1], where the same is shown for *DL-Lite*$^{F}_{horn}$. We thus have the following:

Corollary 7.17. UACQ-ANSWERING *under DL-Lite*$^{F}_{bool}$ *EXPTIME-complete, even in case where the TBox is fixed.*

This concludes the section about query answering under description logic knowledge bases.

8 Conclusions

In this chapter, we give a brief overview and summary of the results in this thesis and discuss meaningful conclusions we can draw from them. In addition, we suggest general directions for future research regarding ontological query answering under sets of DTGDs.

8.1 Overview

In this thesis, we have investigated the impact of allowing disjunctions in rule heads of existential rules, or TGDs, when considering the query answering problem. To this end, we concentrated on three established, expressive classes of TGDs that guarantee decidability, namely guarded-based TGDs, sticky sets of TGDs and weakly-acyclic TGDs, and defined analogous classes for disjunctive TGDs (DTGDs). For each of the resulting languages, a thorough decidability and complexity investigation of the query answering problem was performed.

For the guarded-based classes of DTGDs, we have shown a strong 2ExpTime lower bound, even a fixed set of DTGDs of the weakest language considered, which is the language of disjunctive inclusion dependencies (DIDs). Together with an 2ExpTime upper bound for our most expressive class, weakly-frontier-guarded sets of DTGDs, and some easily obtainable co-NP-completeness results in the data complexity, we answered the entire complexity question for all guarded-based languages when answering arbitrary conjunctive queries. We then proceeded to show that the same complexity bounds also hold for queries of bounded (hyper-)treewidth. When restricted to acyclic queries, we have shown that the complexity of the query answering problem in most cases drops to ExpTime in case where the maximum predicate arity or the entire set of DTGDs is fixed. When further restricted to atomic queries, we exhibited two tractability results for the weakest formalisms, namely linear DTGDs and DIDs. In the rest of the cases, the complexity remains the same as for acyclic queries.

In the case of sticky sets of DTGDs, we established an undecidability result for the query answering problem, even when restricted to atomic queries. In fact, for general queries, this result holds even when only unary and binary predicates are allowed in the schema. However, we then extended this undecidability result even to the case of data complexity, where the set of DTGDs and the query is considered fixed. This is indeed a strong indicator that the principle of stickiness is not enough to ensure decidability vis-a-vis non-determinism.

Lastly, we investigated the class of weakly-acyclic sets of DTGDs. For the query answering problem there, we have shown a co-N2ExpTime lower bound and corresponding upper bound by reduction from the square tiling problem with a square of double-exponential size

in some input number n. Both of these results hold for arbitrary queries as well as atomic queries, and even when the predicate arity of the schema is bounded. In case of a fixed set of DTGDs, we showed Π_2^{P}-completeness for the general query answering problem, and co-NP-completeness for all restricted query languages considered (i.e. bounded (hyper-)treewidth, acyclic and atomic queries). In the data complexity, the problem complexity was shown to be reduced to co-NP-completeness in all cases.

Additionally, we look at applications and consequences of our results to description logics. After examining the construction of the proofs constructed in previous chapters, we were able to state two generic complexity results for description logics that are sub-formalisms of the expressive description logic $\mathcal{ALCIH}$. Firstly, for any such description logic powerful enough to express limited existential quantification, concept union, the role inverse and role hierarchies, the query answering problem immediately becomes 2ExpTime-hard in general. Secondly, for any description logic that is a sub-formalism of $\mathcal{ALCIH}$ and powerful enough to express limited existential quantification, concept union and the rule inverse, answering acyclic queries immediately becomes ExpTime-hard, even when the set of inclusions (i.e. the TBox) is fixed. We then also proceeded to show that corresponding upper bounds, although partially known, can be obtained from our results, by giving a reduction from query answering under $\mathcal{ALCIH}$ knowledge bases to query answering under guarded DTGDs, enriched with negative constraints.

8.2 Brief Discussion of Results

In order to draw some meaningful conclusions from the bare complexity results given in the previous chapters, we will in this section give a brief discussion and interpretation of our results.

When considering guarded-based classes of TGDs, the impact of allowing disjunction can be described as follows: for arbitrary queries (as well as bounded (hyper-)treewidth queries), the "cost" of non-determinism is profound when considering less expressive formalisms. As an example, for DIDs or linear DTGDs we have an increase in complexity from PSpace-complete to 2ExpTime-complete. An even starker impact can be observed, when a fixed set of rules is considered. For (non-disjunctive) IDs, the query answering problem is known to be co-NP-complete. However, when considering DIDs, we have a jump to 2ExpTime. On the other hand, for expressive classes of TGDs, like weakly-guarded or frontier-guarded sets of TGDs, we have no change in complexity in general. This means that for these classes, disjunction virtually comes for free.

Turning to the language of sticky sets of TGDs, we observe that the addition of disjunction is a strong cause of undecidability in this case, even when the query is as simple as a single atom. In fact this undecidability does not stem from the fact that the set of rules is sticky, but from the fact that stickiness allows for Cartesian products to be expressed. We exploit

precisely this combination of cross product rules and disjunction to guess the contents of each cell of a grid that is infinite in both dimensions. This allows us to "guess" a computation of a general, deterministic Turing machine, which can then be checked via a query. Thus, the root cause of undecidability is the fact that, using cross products, two values can be associated with each other, and then, using a disjunctive rule, a non-deterministic choice can be made for each such association. We conjecture that if non-deterministic choices on such associations are forbidden, the query answering problem becomes decidable.

Finally, for the language of weakly-acyclic sets of rules, we observe a correspondence between the cases of deterministic and non-deterministic sets of rules, and the relevant deterministic and non-deterministic complexity classes. For example, for TGDs, the query answering problem is known to be 2ExpTime-complete, while for DTGDs, it becomes co-N2ExpTime-complete. The same thing can be observed in the data complexity, where we have PTime-completeness and co-NP-completeness, respectively for these two cases. In fact, this correspondence, although not trivial to prove, is not surprising. The fact that the chase expansion under a weakly-acyclic set of TGDs is finite and can only be of double-exponential (resp. polynomial) size in the general (resp. data complexity) case, and the fact that the disjunctive chase can be seen as non-deterministically producing such chase expansions, already points towards this dichotomy.

8.3 Directions for Future Research

With the investigation in this thesis, we have laid the groundwork for non-deterministic reasoning with existential rules. However, even without disjunctions, ontological reasoning over databases is a comparatively young research topic, and many interesting and challenging research problems remain open. The following paragraphs describe issues we are planning to tackle in the future:

Other Expressive Fragments

While in this thesis we have concentrated on three established, decidable languages of TGDs (guarded-based, sticky and weakly-acyclic), there are other decidable classes that can be found in the literature, where it would be interesting to investigate what the impact of allowing disjunction is on them w.r.t. query answering. Some of these languages have been surveyed briefly at the end of Chapter 3.

One such notion is the class of tame sets of TGDs, which was introduced by Gottlob et al. [2013b], and combines the paradigms underlying guardedness and stickiness. The possibility of combining the notions of guardedness and weak acyclicity to obtain more expressive classes of TGDs was studied by Krötzsch and Rudolph [2011]. The class of domain-restricted TGDs, which guarantees first-order rewritability, is investigated in [Baget et al., 2009]. For none of

these classes it is known up to now, what the impact of allowing disjunctions in rule heads would be.

Another area where disjunction has not been fully studied is negation. In [Calì et al., 2012], guarded TGDs are enriched with *stratified negation*, a simple non-monotonic form of negation in rules, classically used in the context of Datalog. Negation for guarded TGDs has also been studied under the well-founded semantics [Gottlob et al., 2013a], and stable model semantics [Gottlob et al., 2014]. In each of these cases, we would like to investigate how we can add disjunction into the mix, and what the impact on the complexity of reasoning might be. Furthermore, apart from enriching languages with disjunction, studying the addition of negation to sticky and weakly-acyclic languages would be interesting in its own right. Thus there are two open questions that arise: how to add stable or well-founded negation to sticky sets or weakly-acyclic sets of TGDs, and the same for sets of DTGDs.

For sticky sets of DTGDs, it would be interesting to find sufficient conditions to guarantee decidability. As we have seen in Chapter 5, the combination of cross products and disjunctions lead to undecidability. However, upon closer inspection of the proofs, it seems that they only work if a non-deterministic choice can be made for a pair of values that have been associated with each other via the application of a cross product rule. One possibility to obtain decidability could be to disallow precisely such non-deterministic guesses, and limit them to single values not associated with such a pair. If considering only $\times$-DIDs, then this would isolate the non-determinism to the tree-like part of the chase, thus possibly providing enough guarantees for decidability. Thus, as a conjecture, a possible way of restoring decidability could be to require the following abstract property: a descendant atom of an atom introduced by a TGD containing a join (even if only a cross-join) is not allowed to cause the application of a disjunctive TGD in the chase.

Lastly, the question of EGDs that are non-separable is still open. In the world of description logics, there are certain formalisms that allow for EGDs to be violated during the construction of the chase (see, e.g. $DL\text{-}Lite_{bool}^{HF}$ [Artale et al., 2009], or $\mathcal{ALCHF}$), but the main reasoning tasks remain decidable. In order to express more powerful functionality and cardinality restrictions, it would be helpful to establish syntactic conditions for EGDs that are expressive enough to violate separability, yet still guarantee decidability. Small steps in this direction have already been undertaken (see, e.g. [Calì et al., 2011, 2012a]), but so far none achieve the level of expressiveness of functional constraints, for example in $DL\text{-}Lite_{bool}^{HF}$. Again, this issue is initially separate from the issue of disjunction, and non-determinism can be added to the mix later on.

Finite Controllability and Finite Model Reasoning

In this work so far we have only considered reasoning under arbitrary models, and the models may thus be infinite. However, in the database world, much interest is usually paid to the case of finite databases (i.e. finite instances). For a class $\mathcal{C}$ of dependencies, an interesting question

is whether it exhibits the property of *finite controllability*, that is, given a CQ q, a set of (D)TGDs Σ and a database D, is it true that $D \cup \Sigma \models q$ if and only if $D \cup \Sigma \models_{fin} q$? Here, $\models_{fin}$ denotes entailment under finite models, that is, the query is true if it is true in all finite models.

For our classes, finite controllability was first shown for a restricted form of IDs by Johnson and Klug [1984], and extended to arbitrary IDs by Rosati [2011]. Bárány et al. [2014] recently extended this further, and showed decidability for the entire guarded fragment of first-order logic w.r.t. query answering. This implies that guarded DTGDs are finitely controllable, and this can easily be extended to also cover the weakly- and frontier-guarded classes of DTGDs. Recently also sticky sets of TGDs where shown, by Gogacz and Marcinkowski [2013], to have the finite controllability property. The question arises whether whether this also holds for DTGDs.

Especially when taking into account EGDs, there are cases where classes of dependencies are not finitely controllable (see, e.g. the class of non-conflicting conceptual dependencies [Calì et al., 2011], which are a restricted form of IDs in combination with functionality constraints that can be expressed as EGDs). In such cases, we would like to explicitly investigate the complexity of query answering under such constraints. In the world of description logics, such investigations have been performed, for example in [Calvanese, 1996; Lutz et al., 2005; Rosati, 2008]. An overview over finite model theory can be found in the book on the topic by Libkin [2004].

Parameterized Complexity

In traditional complexity theory, the single property that is examined about a problem is its input size: for example, for a problem in NP, a problem instance of size n can be decided by a non-deterministic Turing machine in an amount of time that is polynomial in n. Therefore the only "parameter" that classical complexity theory knows is the problem size. In practice there are many computationally hard (i.e. intractable) problems that have to be solved. It turns out however, that, even though the problem is known to be intractable in general, there are a number of instances for which solving the problem is easy. But still classical complexity theory tells us that the problem is hard. Parameterized complexity tries to deal with this seemingly contradictory phenomenon. The interested reader can find good disquisitions on the matter in relevant books by Downey and Fellows [1999]; Flum and Grohe [2006]; Niedermeier [2006].

Unless PTIME = NP(or more precisely, unless the exponential time hypothesis is false), in order to solve NP-hard problems, algorithms need exponential running time. The idea of parameterized complexity is to find problem-specific parameters of problems, such that the running time of a suitable algorithm can be expressed (in big-O notation) as a function of the parameter and the input size, with the underlying idea that the exponentiality of the running time can be restricted to the parameter and only the parameter. For relatively "small" values of the parameter, solving the problem can then be very efficiently done. If such a parameterization

exists the problem is said to be *fixed-parameter tractable*, as bounding the parameter by some constant yields a polynomial-time algorithm.

Fixed-parameter tractability (or FPT, for short) is a complexity class defined as follows:

Definition 8.1 (Fixed-Parameter Tractable (FPT) Complexity)**.**

$$FPT = \{\mathcal{L} \mid \exists \textit{ deterministic Turing machine deciding } \mathcal{L} \textit{ in } f(k) \cdot n^{O(1)} \textit{ time}\}$$

where n is the input size, $\mathcal{L}$ is a language (or problem), k is a problem-specific parameter and $f(k)$ is some computable function depending only on k and not on n.

The main idea of this definition is to exclude problems with runtime n^k, for a parameter k: the runtime in terms of the input size n only depends on the input size and not on the parameter.

Regarding query evaluation over databases, a good introduction to parameterized complexity in this field can be found in [Grohe, 2001], where different complexity classes and model checking problems are surveyed for their complexity parameterized by query size (i.e. the number of symbols needed to encode the query). In [Downey et al., 1996], a new characterization of the non-FPT complexity class $W[1]$ has been established, based on the parameterization of monotone conjunctive queries by its size. However, to the best of our knowledge, parameterized complexity for query answering under TGDs or DTGDs has not yet been investigated.

However, first results have been established by Simancik et al. [2011b] for description logics, where a tree decomposition-based resolution calculus is presented to establish a fixed-parameter result for reasoning under certain description logics. We would like to investigate the problem of query answering under (D)TGDs w.r.t. its parameterized complexity, using different parameterizations, in the style done for query answering without constraints by Papadimitriou and Yannakakis [1999].

Bibliography

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

Acciarri, A., Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., Palmieri, M., and Rosati, R. (2005). QuOnto: Querying Ontologies. In *Proceedings of the 20th National Conference on Artificial Intelligence and the 27th Innovative Applications of Artificial Intelligence Conference, AAAI*, pages 1670–1671. AAAI Press / The MIT Press.

Alviano, M., Faber, W., Leone, N., and Manna, M. (2012). Disjunctive Datalog with Existential Quantifiers: Semantics, Decidability, and Complexity Issues. *Theory and Practice of Logic Programming*, 12(4-5):701–718.

Andréka, H., Németi, I., and van Benthem, J. (1998). Modal Languages and Bounded Fragments of Predicate Logic. *Journal of Philosophical Logic*, 27(3):217–274.

Artale, A., Calvanese, D., Kontchakov, R., and Zakharyaschev, M. (2009). The *DL-Lite* Family and Relations. *Journal of Artificial Intelligence Research*, 36:1–69.

Baader, F. (2003). Least Common Subsumers and Most Specific Concepts in a Description Logic with Existential Restrictions and Terminological Cycles. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI*, pages 319–324. Morgan Kaufmann.

Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the $\mathcal{EL}$ Envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI*, pages 364–369. Professional Book Center.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baget, J.-F., Leclère, M., Mugnier, M.-L., and Salvat, E. (2009). Extending Decidable Cases for Rules with Existential Variables. In *Proceedings of the 21nd International Joint Conference on Artificial Intelligence, IJCAI*, pages 677–682. IJCAI/AAAI.

Baget, J.-F., Leclère, M., Mugnier, M.-L., and Salvat, E. (2011a). On Rules with Existential Variables: Walking the Decidability Line. *Artificial Intelligence*, 175(9-10):1620–1654.

Baget, J.-F. and Mugnier, M.-L. (2002). Extensions of Simple Conceptual Graphs: The Complexity of Rules and Constraints. *Journal of Artificial Intelligence Research*, 16:425–465.

Baget, J.-F., Mugnier, M.-L., Rudolph, S., and Thomazo, M. (2011b). Walking the Complexity Lines for Generalized Guarded Existential Rules. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, pages 712–717. IJCAI/AAAI.

Bárány, V., Gottlob, G., and Otto, M. (2014). Querying the Guarded Fragment. *Logical Methods in Computer Science*, 10(2).

Bárány, V., ten Cate, B., and Otto, M. (2012). Queries with Guarded Negation. *Proceedings of the VLDB Endowment*, 5(11):1328–1339.

Bárány, V., ten Cate, B., and Segoufin, L. (2011). Guarded Negation. In Aceto, L., Henzinger, M., and Sgall, J., editors, *Proceedings of the 38th Colloquium on Automata, Languages and Programming, ICALP*, volume 6756 of *Lecture Notes in Computer Science*, pages 356–367. Springer.

Beeri, C. and Vardi, M. Y. (1981). The Implication Problem for Data Dependencies. In *Proceedings of the 8th Colloquium on Automata, Languages and Programming, ICALP*, volume 115 of *Lecture Notes in Computer Science*, pages 73–85. Springer.

Björklund, H., Martens, W., and Schwentick, T. (2008). Optimizing Conjunctive Queries over Trees Using Schema Information. In Ochmanski, E. and Tyszkiewicz, J., editors, *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 5162 of *Lecture Notes in Computer Science*, pages 132–143. Springer.

Bourhis, P., Morak, M., and Pieris, A. (2013a). The Impact of Disjunction on Query Answering Under Guarded-Based Existential Rules. In Rossi, F., editor, *Proceedings of the 23nd International Joint Conference on Artificial Intelligence, IJCAI*. IJCAI/AAAI.

Bourhis, P., Morak, M., and Pieris, A. (2013b). The Impact of Disjunction on Query Answering Under Guarded-Based Existential Rules. In Eiter, T., Glimm, B., Kazakov, Y., and Krötzsch, M., editors, *Proceedings of the 26th International Workshop on Description Logics, DL*, volume 1014 of *CEUR Workshop Proceedings*, pages 539–551. CEUR-WS.org.

Bourhis, P., Morak, M., and Pieris, A. (2014a). Acyclic Query Answering under Guarded Disjunctive Existential Rules and Consequences to DLs. In Bienvenu, M., Ortiz, M., Rosati, R., and Simkus, M., editors, *Proceedings of the 27th International Workshop on Description Logics, DL*, volume 1193 of *CEUR Workshop Proceedings*, pages 100–111. CEUR-WS.org.

Bourhis, P., Morak, M., and Pieris, A. (2014b). Towards Efficient Reasoning Under Guarded-Based Disjunctive Existential Rules. In Csuhaj-Varjú, E., Dietzfelbinger, M., and Ésik, Z., editors, *Proceedings of the 39rd International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 8634 of *Lecture Notes in Computer Science*, pages 99–110. Springer.

Cabibbo, L. (1998). The Expressive Power of Stratified Logic Programs with Value Invention. *Information and Computation*, 147(1):22–56.

Calì, A., Gottlob, G., and Kifer, M. (2013). Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *Journal of Artificial Intelligence Research*, 48:115–174.

Calì, A., Gottlob, G., and Lukasiewicz, T. (2012). A General Datalog-based Framework for Tractable Query Answering over Ontologies. *Journal of Web Semantics*, 14:57–83.

Calì, A., Gottlob, G., Orsi, G., and Pieris, A. (2012a). On the Interaction of Existential Rules and Equality Constraints in Ontology Querying. In Erdem, E., Lee, J., Lierler, Y., and Pearce, D., editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 117–133. Springer.

Calì, A., Gottlob, G., and Pieris, A. (2011). New Expressive Languages for Ontological Query Answering. In *Proceedings of the 25th Conference on Artificial Intelligence, AAAI*. AAAI Press.

Calì, A., Gottlob, G., and Pieris, A. (2011). Querying Conceptual Schemata with Expressive Equality Constraints. In Jeusfeld, M. A., Delcambre, L. M. L., and Ling, T. W., editors, *Proceedings of the 30th International Conference on Conceptual Modelling, ER*, volume 6998 of *Lecture Notes in Computer Science*, pages 161–174. Springer.

Calì, A., Gottlob, G., and Pieris, A. (2012b). Towards More Expressive Ontology Languages: The Query Answering Problem. *Artificial Intelligence*, 193:87–128.

Calì, A., Lembo, D., and Rosati, R. (2003a). On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proceedings of the 22th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 260–271. ACM.

Calì, A., Lembo, D., and Rosati, R. (2003b). Query Rewriting and Answering under Constraints in Data Integration Systems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI*, pages 16–21. Morgan Kaufmann.

Calvanese, D. (1996). Finite Model Reasoning in Description Logics. In *Proceedings of the 1996 International Workshop on Description Logics, DL*, volume WS-96-05 of *AAAI Technical Report*, pages 25–36. AAAI Press.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2011). The MASTRO System for Ontology-based Data Access. *Semantic Web*, 2(1):43–53.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2013). Data Complexity of Query Answering in Description Logics. *Artificial Intelligence*, 195:335–360.

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., and Rosati, R. (2007). Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *Journal of Automated Reasoning*, 39(3):385–429.

Casanova, M. A., Fagin, R., and Papadimitriou, C. H. (1984). Inclusion Dependencies and Their Interaction with Functional Dependencies. *Journal of Computer and System Science*, 28(1):29–59.

Ceri, S., Gottlob, G., and Tanca, L. (1990). *Logic Programming and Databases*. Springer.

Chandra, A. K., Kozen, D., and Stockmeyer, L. J. (1981). Alternation. *Journal of the ACM*, 28(1):114–133.

Chandra, A. K. and Merlin, P. M. (1977). Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, STOC*, pages 77–90. ACM.

Chandra, A. K. and Vardi, M. Y. (1985). The Implication Problem for Functional and Inclusion Dependencies is Undecidable. *SIAM Journal on Computation*, 14(3):671–677.

Chekuri, C. and Rajaraman, A. (2000). Conjunctive Query Containment Revisited. *Theoretical Computer Science*, 239(2):211–229.

Courcelle, B. (1990). The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425.

Deutsch, A., Nash, A., and Remmel, J. B. (2008). The Chase Revisited. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 149–158. ACM.

Deutsch, A. and Tannen, V. (2003). Reformulation of XML Queries and Constraints. In Calvanese, D., Lenzerini, M., and Motwani, R., editors, *Proceedings of the 9th International Conference on Database Theory, ICDT*, volume 2572 of *Lecture Notes in Computer Science*, pages 225–241. Springer.

Deutsch, A. and Tannen, V. (2005). XML Queries and Constraints, Containment and Reformulation. *Theoretical Computer Science*, 336(1):57–87.

Downey, R. G. and Fellows, M. R. (1999). *Parameterized Complexity*. Monographs in Computer Science. Springer.

Downey, R. G., Fellows, M. R., and Taylor, U. (1996). The Parameterized Complexity of Relational Database Queries and an Improved Characterization of *W*[1]. In *Proceedings of the 1st International Conference on Discrete Mathematics and Theoretical Computer Science, DMTCS*, Discrete Mathematics and Theoretical Computer Science, pages 194–213. Springer Verlag.

Eiter, T., Gottlob, G., and Mannila, H. (1997). Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418.

Eiter, T., Ianni, G., and Krennwallner, T. (2009). Answer Set Programming: A Primer. In Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M., and Schmidt, R. A., editors, *Tutorial Lectures of the 5th International Summer School on Reasoning Web*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer.

Eiter, T. and Simkus, M. (2010). FDNC: Decidable Nonmonotonic Disjunctive Logic Programs with Function Symbols. *ACM Transactions on Computational Logic*, 11(2).

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

Faltings, B. and Macho-Gonzalez, S. (2005). Open Constraint Programming. *Artificial Intelligence*, 161(1-2):181–208.

Flum, J. and Grohe, M. (2006). *Computational Complexity*. Springer.

Gogacz, T. and Marcinkowski, J. (2013). Converging to the Chase—A Tool for Finite Controllability. In *Proceedings of the 28th Annual IEEE Symposium on Logic in Computer Science, LICS*, pages 540–549. IEEE Computer Society.

Gonçalves, R. S., Bail, S., Jiménez-Ruiz, E., Matentzoglu, N., Parsia, B., Glimm, B., and Kazakov, Y. (2013). OWL Reasoner Evaluation (ORE) Workshop 2013 Results: Short Report. In Bail, S., Glimm, B., Gonçalves, R. S., Jiménez-Ruiz, E., Kazakov, Y., Matentzoglu, N., and Parsia, B., editors, *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation, ORE*, volume 1015 of *CEUR Workshop Proceedings*, pages 1–18. CEUR-WS.org.

Gottlob, G., Hernich, A., Kupke, C., and Lukasiewicz, T. (2013a). Well-Founded Semantics for Extended Datalog and Ontological Reasoning. In Hull, R. and Fan, W., editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 225–236. ACM.

Gottlob, G., Hernich, A., Kupke, C., and Lukasiewicz, T. (2014). Stable Model Semantics for Guarded Existential Rules and Description Logics. In Baral, C., Giacomo, G. D., and Eiter, T., editors, *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR*. AAAI Press.

Gottlob, G., Leone, N., and Scarcello, F. (2001). The Complexity of Acyclic Conjunctive Queries. *Journal of the ACM*, 48(3):431–498.

Gottlob, G., Leone, N., and Scarcello, F. (2002). Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Science*, 64(3):579–627.

Gottlob, G., Leone, N., and Scarcello, F. (2003). Robbers, Marshals, and Guards: Game Theoretic and Logical Characterizations of Hypertree Width. *Journal of Computer and System Science*, 66(4):775–808.

Gottlob, G., Manna, M., Morak, M., and Pieris, A. (2012). On the Complexity of Ontological Reasoning under Disjunctive Existential Rules. In Rovan, B., Sassone, V., and Widmayer, P., editors, *Proceedings of the 37rd International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 1–18. Springer.

Gottlob, G., Manna, M., and Pieris, A. (2013b). Combining Decidability Paradigms for Existential Rules. *Theory and Practice of Logic Programming*, 13(4-5):877–892.

Gottlob, G., Orsi, G., and Pieris, A. (2011). Ontological Queries: Rewriting and Optimization. In *Proceedings of the 27th International Conference on Data Engineering, ICDE*, pages 2–13. IEEE Computer Society.

Grädel, E. (1999). On The Restraining Power of Guards. *Journal of Symbolic Logic*, 64(4):1719–1742.

Grohe, M. (2001). The Parameterized Complexity of Database Queries. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 82–92. ACM.

Horrocks, I., Kutz, O., and Sattler, U. (2006). The Even More Irresistible $\mathcal{SROIQ}$. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning, KR*, pages 57–67. AAAI Press.

Immerman, N. (1988). Nondeterministic Space is Closed under Complementation. *SIAM Journal on Computation*, 17(5):935–938.

Johnson, D. S. and Klug, A. C. (1984). Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *Journal of Computer and System Science*, 28(1):167–189.

Kolaitis, P. G. and Vardi, M. Y. (2000). Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Science*, 61(2):302–332.

Krötzsch, M. and Rudolph, S. (2011). Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, pages 963–968. IJCAI/AAAI.

Larson, J. T. (2006). *Complexity in Databases, Games and Logics*. PhD thesis, University of California, Santa Cruz, California, USA.

Leone, N., Manna, M., Terracina, G., and Veltri, P. (2012). Efficiently Computable Datalog$^{\exists}$ Programs. In Brewka, G., Eiter, T., and McIlraith, S. A., editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, KR*. AAAI Press.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562.

Libkin, L. (2004). *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer-Verlag.

Lutz, C. (2008). The Complexity of Conjunctive Query Answering in Expressive Description Logics. In Armando, A., Baumgartner, P., and Dowek, G., editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning, IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 179–193. Springer.

Lutz, C., Sattler, U., and Tendera, L. (2005). The Complexity of Finite Model Reasoning in Description Logics. *Information and Computation*, 199(1-2):132–171.

Mailharro, D. (1998). A Classification and Constraint-based Framework for Configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):383–397.

Marnette, B. (2009). Generalized Schema-Mappings: From Termination to Tractability. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 13–22. ACM.

Meier, A. and Schneider, T. (2013). Generalized Satisfiability for the Description Logic $\mathcal{ALC}$. *Theoretical Computer Science*, 505:55–73.

Meier, M., Schmidt, M., and Lausen, G. (2009). On Chase Termination Beyond Stratification. *Proceedings of the VLDB Endowment*, 2(1):970–981.

Niedermeier, R. (2006). *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press.

Orsi, G. and Pieris, A. (2011). Optimizing Query Answering under Ontological Constraints. *Proceedings of the VLDB Endowment*, 4(11):1004–1015.

Papadimitriou, C. H. and Yannakakis, M. (1999). On the Complexity of Database Queries. *Journal of Computer and System Science*, 58(3):407–427.

Papadimitriou, C. M. (1994). *Computational Complexity*. Addison-Wesley, Reading, Massachusetts.

Patel-Schneider, P. F. and Horrocks, I. (2007). A Comparison of Two Modelling Paradigms in the Semantic Web. *Journal of Web Semantics*, 5(4):240–250.

Pieris, A. (2011). *Ontological Query Answering: New Languages, Algorithms and Complexity*. PhD thesis, University of Oxford, Oxford, Oxfordshire, UK.

Poggi, A., Lembo, D., Calvanese, D., Giacomo, G. D., Lenzerini, M., and Rosati, R. (2008). Linking Data to Ontologies. *Journal of Data Semantics*, 10:133–173.

Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., and Leone, N. (2009). OntoDLV: An ASP-based System for Enterprise Ontologies. *Journal of Logic and Computation*, 19(4):643–670.

Robertson, N. and Seymour, P. D. (1984). Graph minors. III. Planar Tree-Width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64.

Rosati, R. (2008). Finite Model Reasoning in DL-Lite. In Bechhofer, S., Hauswirth, M., Hoffmann, J., and Koubarakis, M., editors, *Proceedings of the 5th European Semantic Web Conference, ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 215–229. Springer.

Rosati, R. (2011). On the Finite Controllability of Conjunctive Query Answering in Databases under Open-World Assumption. *Journal of Computer and System Science*, 77(3):572–594.

Savitch, W. J. (1970). Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Science*, 4(2):177–192.

Schmidt-Schauß, M. and Smolka, G. (1991). Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26.

Simancik, F., Kazakov, Y., and Horrocks, I. (2011a). Consequence-Based Reasoning beyond Horn Ontologies. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, pages 1093–1098. IJCAI/AAAI.

Simancik, F., Motik, B., and Krötzsch, M. (2011b). Fixed Parameter Tractable Reasoning in DLs via Decomposition. In *Proceedings of the 24th International Workshop on Description Logics, DL*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Spezzano, F. and Greco, S. (2010). Chase Termination: A Constraints Rewriting Approach. *Proceedings of the VLDB Endowment*, 3(1):93–104.

Szelepcsényi, R. (1988). The Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica*, 26(3):279–284.

Thomazo, M., Baget, J., Mugnier, M., and Rudolph, S. (2012). A Generic Querying Algorithm for Greedy Sets of Existential Rules. In Brewka, G., Eiter, T., and McIlraith, S. A., editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, KR*. AAAI Press.

Tsarkov, D. and Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. In Furbach, U. and Shankar, N., editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning, IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 292–297. Springer.

Vardi, M. Y. (1982). The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC*, pages 137–146. ACM.

Vardi, M. Y. (1995). On the Complexity of Bounded-Variable Queries. In *Proceedings of the 14th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 266–276. ACM Press.

Yannakakis, M. (1981). Algorithms for Acyclic Database Schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases, VLDB*, pages 82–94. IEEE Computer Society.

Nomenclature

ABox Assertional Box

ACQ Acyclic Query

BHTWQ Bounded Hyper-Treewidth Query

BTWQ Bound Treewidth Query

CQ Conjunctive Query

CQ_1 Atomic Query

DID Disjunctive Inclusion Dependency

DL Description Logic

DTGD Disjunctive Tuple Generating Dependency

EGD Equality-Generating Dependency

GFO Guarded Fragment of First-Order Logic

GNFO Guarded Negation First-Order Logic

ID Inclusion Dependency

KB Knowledge Base

LGFO Loosely-Guarded Fragment of First-Order Logic

NC Negative Constraint

ODBMS Ontological Database Management System

TBox Terminological Box

TGD Tuple Generating Dependency

UCQ Union of Conjunctive Queries