FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Static Analysis for Ontology-Mediated Querying

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Gerald Berger
Registration Number 0958520

to the Faculty of Informatics
at the TU Wien

Advisor: O.Univ.Prof. Dr. Georg Gottlob
Second advisor: Ao.Univ.Prof. Dr. Uwe Egly

The dissertation has been reviewed by:

Prof. Dr. Pablo Barceló	Prof. Dr. Marie-Laure Mugnier

Vienna, 7th May, 2019

Gerald Berger

Erklärung zur Verfassung der Arbeit

Gerald Berger
Abelegasse 26/2/11, 1160 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. Mai 2019

Gerald Berger

Acknowledgements

First and foremost, I would like to thank my advisor Georg Gottlob for his guidance and support. I thank him for providing a prime example of academic excellence to me, and I deeply appreciate his human qualities as a mentor. Georg is probably the busiest person I have ever met, yet he was able to provide valuable support and insights into topics that go far beyond the scope of this thesis. During the last four years, Georg has definitely become one of my personal role models.

Although Andreas Pieris is not my official advisor, a quick inspection of my publication list reveals that he has been a steady companion of mine during my studies. It is thus fair to say that Andreas is actually an advisor of mine. Without his support, successful completion of this thesis would not be imaginable. I thank him for the great work together and for his constant support. Andreas also helped me a lot in writing the proposal for the DOC fellowship of the Austrian Academy of Sciences, which secured funding for my PhD studies.

During my studies, I have received support from many people. In the initial phase of my studies, I worked as a senior lecturer in the group of Gerti Kappel, and I thank her for this opportunity. Being then a part-time university assistant, Stefan Woltran provided additional financial support during this early phase of my studies, and I thank Reinhard Pichler for helping me to obtain the position as a university assistant at our group in the first place. I am also grateful to Thomas Eiter, who, as the head of our institute, offered help and advice in all circumstances. Moreover, I am thankful that he provided office space to me when I finally transitioned from a university position to my DOC fellowship. I also thank Uwe Egly for co-supervising this thesis, and for the support I received from him as a tutor and teaching assistant during my undergraduate and master studies.

Occupying three different positions in four years incurs plenty of bureaucratic effort, and equally does traveling to conferences, etc. I thank Juliane Auerböck, Beatrix Buhl, and Eva Nedoma – the secretaries of our institute – for helping me with all these efforts.

Many thanks also to the doctoral college LogiCS (FWF-project no. W1255-N23) for funding most of my research trips during my studies. Moreover, I thank the numerous young colleagues that I met in this college, and I appreciate the time we spent together.

Special thanks to Pablo Barceló, who invited me for a research stay to Chile, and with whom Andreas Pieris and I have established a fruitful research collaboration during the last years. I also thank him and Marie-Laure Mugnier for agreeing to review this doctoral thesis.

My studies would have been impossible without the help and support of my family, and especially my parents. I also thank my girlfriend Miriam for her love and her constant support, and I apologize to her for spending sometimes too much time on a weird thing like a PhD thesis 🙃.

I have received numerous years high-quality university education and had to virtually pay no fees for that. In this context, I would like to acknowledge the taxpayer's contribution to that with the emphasis that such a contribution is not self-evident in most parts of the world.

Finally, I would like to compose a list of people (in no particular order) whom I additionally want to thank: Elisabeth, Thomas, Philipp, Franz, Lukas, Matthias, Stefan, Rainer, Benjamin, Marijana, Ilina, Jens, Zeynep, Shqiponja, Harald, Thomas, Martin, Adrian, Etienne, Alena, Anna.

Gerald Berger
7th May, 2019

Kurzfassung

Ontology-Based Data Access (OBDA) ist die Heirat von Datenbanktechnologie und ontologischen Theorien, welche eine einheitliche Sicht auf mehrere, möglicherweise heterogene, Datenquellen bieten. Ontologische Theorien sind eine Menge logischer Behauptungen, welche es ermöglichen, neues Wissen aus den gegebenen extensionalen Daten abzuleiten. Benutzeranfragen auf den Daten werden dadurch relativ zu einer logischen Theorie beantwortet. In OBDA ist es üblich, Benutzeranfragen in Verbindung mit einer ontologischen Theorie als ein zusammengesetztes Objekt zu betrachten, welches man Ontology-Mediated Query (OMQ) nennt.

Ontologische Theorien können prinzipiell in jeder verfügbaren logischen Sprache formuliert werden. Einen etablierten Formalismus, um diese Theorien auszudrücken, stellen Tuple-Generating Dependencies (TGDs) dar, da sich diese zur Formulierung vieler praktisch relevanter Szenarien eignen. Das Hauptaugenmerk der vorliegenden Dissertation liegt auf Ontology-Mediated Querys basierend auf TGDs. Während dem Evaluierungsproblem dieser viel Aufmerksamkeit in der Literatur gewidmet wurde, beschäftigt sich die vorliegende Arbeit mit anderen computationalen Problemen für diese, nämlich den sogenannten Problemen der statischen Analyse. Diese Probleme bezeichnen von den Daten unabhängige Entscheidungsprobleme, welche insbesondere Anwendungen in der Optimierung von OMQs finden.

Als ersten Beitrag studieren wir das Implikationsproblem für viele der bekannten Klassen von OMQs. Insbesondere betrachten wir das Implikationsproblem für OMQs, welche mit Hilfe von (frontier-)guarded, non-recursive, und sticky Mengen von TGDs formuliert werden. Dazu entwickeln wir spezielle Techniken, um für diese Klassen das jeweilige Implikationsproblem zu lösen, und wir beweisen exakte Komplexitätsresultate.

In Folge widmen wir uns dem sogenannten First-Order Rewritability Problem für OMQs, welches jenes Entscheidungsproblem bezeichnet, das nach der Existenz einer zur gegebenen OMQ äquivalenten Anfrage in der Prädikatenlogik erster Stufe fragt. Wir studieren dieses Problem für jene OMQs, welche auf (frontier-)guarded TGDs basieren, und wir beweisen exakte Komplexitätsresultate mit Hilfe von Cost-Automaten auf Bäumen.

Ein wiederkehrendes Phänomen bei der Untersuchung von Problemen der statischen Analyse ist die Feststellung, dass die entsprechenden Entscheidungsprobleme bereits für Datalog-Anfragen unentscheidbar sind. Dies trifft sowohl auf das Implikationsproblem, als auch auf das First-Order Rewritability Problem zu. In dem dritten Beitrag dieser Dissertation untersuchen wir den Zusammenhang zwischen Datalog-Anfragen und

OMQs basierend auf TGDs. Wir entwickeln ein starkes hinreichendes Kriterium für die Ausdrückbarkeit von OMQs als Datalog-Anfragen. Dabei zeigen wir auch, dass OMQs formuliert in der Sprache von VADALOG – ein kommerziell eingesetztes Reasoning System für Knowledge Graphen – immer als Datalog-Anfragen ausgedrückt werden können. Weiters identifizieren wir ein Fragment von VADALOG, welches hinsichtlich Speicherverbrauch effizienter ist, jedoch trotzdem viele in der Praxis relevante Abfragen ausdrücken kann.

Abstract

Ontology-based data access (OBDA) is the marriage of database technology with ontological theories that provide a unified conceptual view on multiple, possibly heterogeneous data sources. Ontological theories are presented as logical assertions that are capable of deriving new knowledge from extensional data. User queries are thus answered against a logical theory put on top of the data. In OBDA, it is common to view a user query together with an ontological theory as a composite object, which is coined ontology-mediated query (OMQ).

Ontological theories can, in principle, be formulated using any available logical formalism. However, it is widely acknowledged that tuple-generating dependencies (TGDs) are a convenient formalism for formulating many ontologies that occur in real-life scenarios. Ontology-mediated queries based on TGDs are the main objects of study in this thesis. While the evaluation problem for such queries has received much attention in the literature, this thesis is dedicated to the study of another important family of computational tasks for OMQs, namely so-called static analysis tasks. These tasks are data-independent computational problems that can be employed for query optimization.

As a first contribution, we study the query containment problem for many of the well-known ontology-mediated query languages based on TGDs. More specifically, we study containment for classes of OMQs formulated via (frontier-)guarded, non-recursive, and sticky sets of TGDs. We develop specifically tailored techniques for deciding the containment problems for each of them, allowing us to obtain sharp complexity bounds.

We then focus our attention on the first-order rewritability problem for OMQs, that is, the problem asking whether or not a given OMQ can be equivalently expressed as a plain first-order query. For this second contribution, we study first-order rewritability for OMQs based on (frontier-)guarded TGDs, and we pinpoint its exact complexity by using the sophisticated model of cost automata on trees.

A recurring theme for static analysis tasks is that many of them are already undecidable when one focuses on Datalog queries, and this holds true for containment and first-order rewritability. As a third contribution, we investigate the relationship between OMQs based on TGDs and the existence of rewritings into Datalog queries. We develop a strong sufficient criterion for OMQs to be expressible as Datalog queries, and we show that OMQs in the language of Vadalog – a commercially employed reasoning engine for knowledge graphs – are always Datalog rewritable. We furthermore identify a space-efficient fragment of the language of Vadalog that restricts the use of recursion, but still allows to express many queries occurring in real-life scenarios.

Contents

CHAPTER 1

Introduction

The novel application of knowledge representation techniques for handling incomplete and inherently heterogeneous data sources is giving rise to a new field of study, recently coined as *knowledge-enriched data management* [6]. A central point of interest within this emerging field is that of *ontology-based data access* (OBDA) [118], which amounts to the utilization of ontologies for providing a global and unified conceptual view on data sources. The ontologies serve as *mediators* between user queries and individual data sources. Users thus formulate their queries in the schema provided by the ontology, and they are answered against data sources by harnessing the knowledge that is encoded in the ontology. This is in stark contrast to the classical database setting, where the dominating assumption is that databases represent *closed-world* objects that explicitly represent the information of a user's interest.

In OBDA, one interprets the ontology $\mathcal{O}$ and the user query q as two different objects, where the former is usually specified via a logical formalism, while the latter belongs to a standard database query language, in our case (unions of) conjunctive queries (abbreviated (U)CQs). The marriage of $\mathcal{O}$ and q gives rise, together with a data schema $\mathbf{S}$ specifying the schema of legal input data, to a composite object $Q = (\mathbf{S}, \mathcal{O}, q)$ which is coined *ontology-mediated query* (OMQ) [37]. Ontology-mediated queries and their associated computational problems are central subjects of study in the field of OBDA.

While in this setting *description logics* (DLs) [12] are often used for modeling ontologies, it is widely accepted that, for handling relations of arbitrary arity in relational databases, it is convenient to resort to *tuple-generating dependencies* (TGDs), also known as *existential rules* or *Datalog*$^{\pm}$ rules [14, 25, 46]. TGDs are logical sentences of the form

$$\tau = \forall \bar{x}, \bar{y}\, (\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})),$$

where $\varphi(\bar{x}, \bar{y})$ and $\psi(\bar{x}, \bar{z})$ are conjunctions of relational atoms. When depicting TGDs, we usually omit the preceding universal quantifiers and separate the atoms of $\varphi(\bar{x}, \bar{y})$ by comma. We call $\varphi(\bar{x}, \bar{y})$ the *body atoms*, and $\psi(\bar{x}, \bar{z})$ the *head atoms* of τ. Thus, sets of TGDs generalize Datalog programs in the sense that, apart from recursion, TGDs also

allow the use of *value invention* which is realized by permitting existential quantifiers in heads of rules.

Example 1.1. Let $Q = (\mathbf{S}, \mathcal{O}, q(x))$, where $\mathbf{S}$ consists of a single unary predicate P, $q(x) = \exists y, z\,(P(x) \wedge F(y, x) \wedge F(z, y) \wedge P(z))$, and $\mathcal{O}$ consists of the rule

$$P(x) \rightarrow \exists y\,(F(y, x) \wedge P(y)).$$

When taking $P(x)$ to stand for the assertion "x is a person," and $F(x, y)$ to stand for "x is the father of y," then the meaning of the above rule is that every person has a father who is also a person. The query $q(x)$ asks for all persons x who have a person as a grandfather (this statement is, of course, true for any person). ⊣

From a semantic point of view, OMQs can be viewed as queries in the classical sense of databases: given an OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ and a database $\mathfrak{D}$ over $\mathbf{S}$, the *certain answers* of Q to $\mathfrak{D}$, denoted $Q(\mathfrak{D})$, is the set of all tuples of constants $\bar{a}$ such that $q(\bar{a})$ is logically entailed by the facts of $\mathfrak{D}$ in conjunction with $\mathcal{O}$. For example, the OMQ from Example 1.1 has as certain answers to an input database $\mathfrak{D}$ those constants a for which $P(a)$ holds in $\mathfrak{D}$. It is of no surprise that a central problem in the context of OBDA is the *(query) evaluation problem* (or *query answering problem*) for OMQs: given an OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$, an $\mathbf{S}$-database $\mathfrak{D}$, and a tuple of constants $\bar{a}$, does $\bar{a} \in Q(\mathfrak{D})$ hold? Beside its theoretical appeal, the practical relevance of this problem is witnessed by the fact that even traditional database providers have built ontological reasoning software on top of their existing products. For example, Oracle Inc. offers semantic technologies via their *Oracle Spatial and Graph* feature on top of their popular database management system.[1] Moreover, query answering under (special classes of) tuple-generating dependencies is the reasoning task implemented in Vadalog, a commercially employed reasoning system, which is also the main product of DeepReason.ai, a company spun off from Oxford University.[2]

Unfortunately, when $\mathcal{O}$ is taken to be an arbitrary set of TGDs, this problem is undecidable [14, 24, 44]. In fact, OMQs using TGDs are even capable of expressing any recursively enumerable database query that is closed under homomorphisms [124]. It is thus generally accepted that, for OBDA purposes, the unrestricted use of TGDs to formulate ontologies is inadequate. This led to a flurry of research activity to identify classes of sets of TGDs for which the evaluation problem becomes decidable [14, 44, 45, 47]. Roughly speaking, we can classify the known classes of TGDs that lead to decidable query evaluation into three categories [14]:

- *Finite expansion sets* These are sets of TGDs $\mathcal{O}$ for which the so-called *chase expansion* w.r.t. any database is finite. Roughly speaking, the chase expansion of $\mathcal{O}$ w.r.t. $\mathfrak{D}$ is an instance constructed by exhaustively applying the rules of $\mathcal{O}$ to the facts of $\mathfrak{D}$ such that all rules in $\mathcal{O}$ are satisfied eventually. To satisfy the rules with existential quantifiers in their heads, one introduces fresh elements that serve as witnesses for the

[1] https://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html
[2] http://www.deepreason.ai

existential demands posed by rule heads. Thus, chase expansions may be infinite as well. For OMQs based on sets of TGDs that have a finite chase expansion w.r.t. any database, the evaluation problem becomes decidable as it amounts to the evaluation of a (U)CQ over a finite database instance.

- *First-order rewritable sets* These are sets of TGDs $\mathcal{O}$ for which any OMQ $Q = (\mathbf{S}, \mathcal{O}, q)$ can actually be equivalently rewritten into a finite first-order query φ_Q. In fact, as we shall see, if an OMQ Q is first-order rewritable, it is also *UCQ-rewritable*, i.e., expressible as a union of conjunctive queries. Since φ_Q is a first-order query – thus essentially a plain SQL-query – the evaluation problem for Q can be solved by evaluating φ_Q over the given input database using standard database technology. Notice thus that the data complexity of the evaluation problem for OMQs based on first-order rewritable sets is in AC_0, since evaluating first-order queries has that data complexity.
- *Finite tree-width sets* These are sets of TGDs for which the chase expansion w.r.t. any database has finite tree-width. Decidability of the evaluation problem for OMQs based on finite tree-width sets can be shown using a celebrated result of Courcelle [65].

The three categories above represent *abstract* criteria for decidability of the evaluation problem. Many of the popular concrete *syntactic* classes of sets of TGDs fall into at least one of these classes, and therefore these abstract criteria provide explanations for the decidability of many of the known syntactic classes of sets of TGDs. Let us briefly state the main concrete syntactic classes of sets of TGDs that we are going to encounter in this thesis (formal definitions are provided later):

- *Linear sets of TGDs* These are sets of TGDs in which every rule has at most one body atom. For example, the rule $R(x, y) \rightarrow \exists z\, S(y, z)$ is linear. Linear TGDs extend *inclusion dependencies* [74] which are, roughly speaking, rules that state that values of a certain column should be contained in the values of another column. Linear sets of TGDs are first-order rewritable sets and finite tree-width sets [45].
- *Non-recursive sets of TGDs* They are also known as *acyclic* sets of TGDs. As their name suggests, these sets of TGDs use no recursion, where recursion is defined as in the case of Datalog. For example, the set of TGDs consisting of the rules

$$R(x, y), T(y, v) \rightarrow \exists z\, S(y, z), \quad S(y, z), A(y) \rightarrow G(y),$$

  is non-recursive. Non-recursive sets of TGDs are first-order rewritable sets, finite expansion sets, and thus also finite tree-width sets.
- *(Frontier-)guarded sets of TGDs* We say that a TGD is *guarded* if it has a body atom that contains all variables of all body atoms as arguments [44], and we say that it is *frontier-guarded* if it has an atom in its body that has a body atom which has all the *frontier variables* as arguments [15], where a frontier variable is a variable that is propagated from the body atoms to the head atoms. Accordingly, (Frontier-)guarded sets of TGDs are sets of TGDs in which each rule is (frontier-)guarded. For example,

the rule $R(x, y), A(y) \rightarrow A(x)$ is guarded, while $R(x, y), R(y, z) \rightarrow A(y)$ is frontier-guarded but not guarded. Notice in particular that every guarded TGD is trivially frontier-guarded. (Frontier-)guarded sets of TGDs are finite tree-width sets [15, 44].

- *Sticky sets of TGDs* These are sets of TGDs that restrict the shape of joins in rule bodies in order to obtain sets of TGDs which are able to express cross products and unguarded joins, but which also remain first-order rewritable [47, 117].
- *Warded sets of TGDs* These sets are a subclass of the so-called *weakly frontier-guarded* sets of TGDs [15]. Roughly speaking, weakly frontier-guarded sets of TGDs are sets of TGDs that extend frontier-guarded sets of TGDs by admitting rules that are not frontier-guarded in order to provide full Datalog recursion on the input database. Unfortunately, reasoning under weakly-frontier guarded sets of TGDs becomes intractable in data complexity, in fact ExpTime-complete [15, 44]. Warded sets of TGDs restrict weakly frontier-guarded sets of TGDs so that query answering under warded sets of TGDs becomes tractable in data complexity [84]. Furthermore, warded sets of TGDs are still able to express a plethora of practically relevant ontologies and therefore aim to provide a balance between expressiveness and computational complexity [26, 27, 84]. Since weakly-frontier guarded sets of TGDs are finite tree-width sets [15], so are warded sets.

Static analysis If we restrict OMQs to be formulated using TGDs from a concrete class $\mathcal{C}$, then this gives rise to an *ontology-mediated query language*, and due to its prominence and importance in the setting of OBDA, it is not surprising that the evaluation problem for OMQ languages based on different classes of TGDs has received great attention in the literature [14, 44, 45, 47]. Since the evaluation problem receives an OMQ Q and a database $\mathfrak{D}$ as input, it decides, from a conceptual point of view, a property of Q *relative to* $\mathfrak{D}$. In this thesis, we focus on computational tasks of a different kind, namely so-called *static analysis* tasks for OMQs. Unlike query evaluation, these tasks are data-independent and aim at deciding semantic properties of OMQs that can be harnessed for query optimization. More specifically, we study the following static analysis problems for OMQ languages based on TGDs:

- *Containment and equivalence* Given two OMQs Q_1 and Q_2 that have the same input data schema $\mathbf{S}$, we say that Q_1 is *contained* in Q_2, if $Q_1(\mathfrak{D}) \subseteq Q_2(\mathfrak{D})$ for every database $\mathfrak{D}$ over $\mathbf{S}$. We call Q_1 and Q_2 *equivalent*, if Q_1 is contained in Q_2 and vice versa. The *containment problem* (respectively, *equivalence problem*) for two OMQs Q_1 and Q_2 is the problem of deciding whether Q_1 is contained in (respectively, equivalent to) Q_2. Notice that equivalence of two OMQs can easily be solved by two containment checks. Therefore, we will focus on studying the containment problem. Deciding containment and equivalence for two given OMQs has applications in query optimization. Indeed, if Q_1 is known to be equivalent to Q_2, then instead of evaluating Q_1 over a given database $\mathfrak{D}$, we may instead compute $Q_1(\mathfrak{D})$ by evaluating Q_2 over $\mathfrak{D}$. This is useful when Q_2 belongs to an OMQ language for which more efficient query answering algorithms are known.

- *First-order rewritability* Given an OMQ Q, we say that Q is *first-order rewritable*, if Q can be equivalently expressed as a first-order query φ_Q. The problem of *first-order rewritability* is the task of deciding whether a given OMQ is indeed first-order rewritable. This problem has again applications in query optimization, as a first-order rewritable OMQ Q can be evaluated using standard database technology, provided φ_Q can be effectively obtained. For an OMQ $Q = (\mathbf{S}, \mathcal{O}, q)$ where $\mathcal{O}$ is a first-order rewritable set, the task of deciding first-order rewritability of Q is clearly trivial. However, as we shall see, (frontier-)guarded TGDs are a prominent class of TGDs for which the according first-order rewritability problem is decidable but not trivial.

Despite their prominence in the context of traditional database query languages, an in-depth study of these tasks has not been carried out so far for OMQs based on TGDs. It is a main contribution of this thesis to perform such an in-depth study.

As said above, unlike query evaluation, containment and first-order rewritability are properties of OMQs, and these properties are data-independent. Moreover, query evaluation is an instance of the classical inference problem of first-order logic, while containment and first-order rewritability are of a higher-order nature. Indeed, for the case of containment, one universally quantifies over all possible input databases in its problem definition, while first-order rewritability demands to explore the space of all first-order queries in the pursuit of finding an equivalent one to the given input OMQ. It is thus of no surprise that containment and first-order rewritability are undecidable when one considers these problems in their full generality, i.e., when one imposes no restrictions on the sets of TGDs allowed in the input OMQs. Hence, we focus on classes of TGDs that have a decidable query evaluation problem, and we establish a number of decidability and complexity results for containment and first-order rewritability for OMQs based on these classes. We study containment in Chapter 4, and first-order rewritability in Chapter 5.

Datalog and OMQs As mentioned above, TGDs extend, from a syntactic point of view, plain Datalog rules by allowing the use of existential quantifiers in rule heads. Answering Datalog queries is well-known to be ExpTime-complete, and tractable in data complexity, more precisely, PTime-complete [66, 130]. In contrast, containment and equivalence for Datalog queries are undecidable [126]. This is a recurring theme in the investigation of static analysis tasks for OMQs – namely, that the according tasks already become undecidable when one focuses on plain Datalog queries. As said above, undecidability of containment for Datalog queries is shown in [126], while deciding first-order rewritability of Datalog queries amounts to deciding the so-called *boundedness problem* [3], – that is, the problem whether the least fixed point of a given Datalog query can be, for all input databases, uniformly reached by a number independent of the input data. This problem is well-known to be undecidable for Datalog queries [78]. Thus, containment and first-order rewritability are undecidable for OMQ languages that capture all Datalog queries. On the other hand, many of the known OMQ languages are actually rewritable to Datalog programs. In Chapter 6, we are interested in the relationship between OMQ languages based on TGDs and the existence of Datalog queries equivalent to them.

More specifically, we provide strong sufficient conditions for OMQs to be equivalent to Datalog queries. Although deciding whether a given OMQ is equivalent to a Datalog program is in general undecidable, the investigation of rewritability into Datalog queries yields interesting insights into the relationship between OMQs based on TGDs and Datalog queries. We exploit these insights in Chapter 6 to study the class of warded sets of TGDs. As said above, OMQs based on warded sets of TGDs aim to provide a balance between expressiveness and computational complexity, by (i) capturing all Datalog queries, and (ii) restricting the use of existential quantifiers in rule heads in order to obtain a query evaluation problem that is ExpTime-complete in combined, and, more importantly, PTime-complete in data complexity. Warded sets of TGDs constitute the underlying language of the commercial knowledge graph management system Vadalog [26, 27]. In the context of knowledge graphs, it is important to have low data complexity due to the tremendous amounts of data present in practical knowledge graphs. We contribute to the theory of Vadalog by (i) showing that OMQs based on warded sets of TGDs are always expressible as Datalog queries, and (ii) identifying a space-efficient fragment of warded sets of TGDs that admits an NLogSpace-complete query evaluation problem in data complexity. As NLogSpace consists of parallelizable problems, we hope that this fragment will lead to efficiency gains in future implementations.

1.1 Research Challenges

Let us now provide more details on the individual contributions:

Containment and equivalence As mentioned above, containment and equivalence of OMQs is undecidable when one does not restrict the sets of TGDs used in the OMQs. It turns out that the two main reasons that render containment for OMQs undecidable are (i) undecidability of query evaluation, and (ii) undecidability of containment for Datalog queries. Hence, we are going to focus on OMQ languages that have a decidable query evaluation problem and that are not able to express all Datalog queries. We divide our study of the containment problem as follows:

Linear, non-recursive, and sticky sets of TGDs All these classes share the property of UCQ-rewritability. Transitioning from OMQs to their UCQ-rewritings allows us to solve the containment problem by checking containment of the respective UCQ-rewritings. In fact, if Q_1 and Q_2 belong to a UCQ-rewritable OMQ language $\mathcal{L}$, then the fact that Q_1 is *not* contained in Q_2 is witnessed via a database whose size is bounded by an integer $k \geq 0$, where k is the maximum size of a disjunct in a UCQ-rewriting of Q_1. This *small witness property* allows us to devise a simple non-deterministic algorithm, which makes use of query evaluation as a subroutine, for checking non-containment of Q_1 in Q_2: guess an input database $\mathfrak{D}$ of size at most k, and then check if there is a certain answer in $Q_1(\mathfrak{D})$ that is not present in $Q_2(\mathfrak{D})$. This algorithm allows us to establish optimal complexity bounds for OMQs based on linear and sticky sets of TGDs. The exact complexity for OMQs based on non-recursive sets of TGDs remains open, but we provide "nearly" matching upper and lower bounds.

(Frontier-)guarded sets of TGDs The OMQ language based on guarded TGDs is not UCQ-rewritable, which forces us to develop different tools to study its containment problem. Let us remark that guarded OMQs can be rewritten as guarded Datalog queries but the known rewritings are very large [85], and hence the reduction of containment for guarded OMQs to containment for guarded Datalog does not yield optimal upper bounds. To solve the containment problem for guarded OMQs, we devise procedures based on two-way alternating tree automata from first principles. Containment for OMQs based on frontier-guarded sets of TGDs is then solved by reducing the problem to the guarded case. To this end, we translate frontier-guarded sets of TGDs to guarded sets of TGDs using the technique of *treeification* [18, 20]. This translation is inspired by a similar translation of sentences of guarded negation least fixed point logic (GNFP) to sentences of guarded least fixed point logic (GFP) [20]. Although this reduction is exponential, it still provides optimal upper bounds for containment.

First-order rewritability As mentioned above, the problem of first-order rewritability is trivial for OMQs based on classes of TGDs that are first-order rewritable. The situation looks quite different for OMQs based on (frontier-)guarded sets of TGDs. In fact, there are OMQs based on (frontier-)guarded TGDs that are inherently recursive, and thus not expressible as a first-order query. We address the question whether we can decide first-order rewritability for OMQs based on (frontier-)guarded OMQs and, if yes, pinpoint the exact complexity of this problem. Let us mention that first-order rewritability has been studied in [34] for OMQ languages based on Horn description logics, including those from the $\mathcal{EL}$-family, which are essentially special cases of OMQs based on guarded TGDs that only use unary and binary predicates. In [34], the authors characterize first-order rewritability by a locality property on tree-like databases, which allows them to optimally solve first-order rewritability using automata-based procedures. As we shall see, the approach of [34] does not work in our setting since we allow the use of predicates of arity greater than two. Thus, we have to develop specifically tailored methods to solve first-order rewritability for our OMQ languages of interest.

To solve first-order rewritability for OMQs based on (frontier-)guarded sets of TGDs, we first show how to solve first-order rewritability for OMQs whose set of TGDs is guarded and whose query consists of a single atom (a so-called *atomic query*). For this OMQ language, we provide a semantic characterization of its first-order rewritable OMQs, and then use this characterization to reduce the problem of first-order rewritability to the question whether the language of a certain tree automaton is finite. The resulting procedure runs in 3ExpTime and, unfortunately, turns out to be not optimal. We therefore employ the more sophisticated model of *cost automata on trees* [31, 58, 60] to arrive at an optimal solution, and we adapt the semantic characterization of first-order rewritability to make it amenable to cost automata techniques. Intuitively, cost automata extend traditional automata by allowing to modify counter values when transitioning from one state to another. Input trees t are assigned values from $\mathbb{N} \cup \{\infty\}$, called the *cost* of t, and this value indicates that the minimum cost of an accepting run for t equals this value. A central decision problem for cost automata is *boundedness*, that is, the question whether the cost of all accepted trees is uniformly bounded by a natural number.

We reduce the question of first-order rewritability to the boundedness problem for cost automata on trees. This allows us to show that first-order rewritability for OMQs having a guarded set of TGDs and an atomic query is in 2ExpTime.

Having a solution for this more restricted OMQ language in place, we then solve first-order rewritability for the frontier-guarded case by resorting to the technique of treeification. That is, as in the case of containment, we translate frontier-guarded sets of TGDs to guarded sets of TGDs. Again, this translation is exponential, but we still are able to obtain optimal complexity upper bounds (i.e., 2ExpTime).

Datalog rewritings and wardedness Consider a Datalog query $Q = (\Pi, G(\bar{x}))$, where Π is a set of *Datalog rules* – essentially TGDs without existential quantifiers in rule heads –, and $G(\bar{x})$ the *goal predicate* of Q. It is well-known that Q is equivalent to a possibly infinite union $q = \bigvee_{i \geq 1} q_k$ of conjunctive queries [55]. Roughly speaking, each q_k can be obtained from the rules of Q by, starting with the goal predicate, "unfolding" them backwards. Each of these "unfoldings" can be represented as a tree whose nodes are labeled with atomic formulas such that (i) leaf nodes are labeled by extensional atoms, i.e., atoms over the input schema of Q, and (ii) if an internal node is labeled by an atom α, and has children respectively labeled by $\beta_1, \ldots, \beta_k$, then the Datalog rule $\beta_1, \ldots, \beta_k \rightarrow \alpha$ arises from a rule of Π just by renaming variables. Let us call such a tree a *proof tree*. Each of the CQs q_k is obtained by taking the conjunction of all leaf nodes of some proof tree. Intuitively, each proof tree describes a witnessing computation of the Datalog query Q for some input database(s). It turns out that proof trees are convenient structures for solving several decision problems for Datalog queries. For example, in [55] proof trees are employed to study the problem asking if a Datalog query is contained in a UCQ. We are interested in extending the notion of proof tree to capture also OMQs based on sets of TGDs, and we aim to solve important questions concerning OMQs using proof trees. We divide our contributions as follows:

Proof trees for OMQs While proof trees for Datalog queries are trees whose nodes are labeled by atomic formulas, proof trees for OMQs will be trees whose nodes are labeled by conjunctive queries. As in the case of Datalog queries, proof trees for OMQs represent witnessing computations of an OMQ over a given input database. Thus, they also serve as an elegant generic tool for evaluating OMQs over a given input database. Proof trees are closely related to resolution procedures for query answering as presented in [82]. The guiding principle of proof trees is to provide a generic tree structure, where each node label contains all and only the information necessary to prove that node label (which is a CQ) using resolution. Another way to view proof trees is to consider them as computation trees of a generic alternating query answering algorithm – the CQs labeling the nodes of a proof tree can be viewed as memory snapshots of this algorithm when its computation progresses to the according node.

Notice that proof trees bear an important measure: since their nodes are labeled with CQs, the mentioned alternating query answering algorithm is space-bounded only if we can bound the maximum size of a CQ occurring in a proof tree. We show that this is true for OMQs based on warded sets of rules, and thereby re-establish a result by Gottlob

and Pieris [84] stating that query evaluation for such OMQs is ExpTime-complete in combined, and PTime-complete in data complexity.

Interestingly, size bounds on CQs in proof trees has another interpretation. It turns out that bounds on the size of the CQs occurring in proof trees can be used as an elegant sufficient criterion to ensure Datalog rewritability. We use this result to show that OMQs based on warded sets of TGDs can be expressed as Datalog queries.

A space-efficient fragment of warded rules As said above, Vadalog [26, 27] is a system for performing complex reasoning tasks such as those required for advanced knowledge graphs. Its main language is essentially an implementation of warded sets of rules, and thereby achieves tractability in data complexity. Since knowledge graphs tend to be rather huge in practice, the question arises whether one can identify a fragment of Vadalog that has an even more attractive data complexity than the full language. We contribute to the theory of Vadalog by identifying such a fragment.

The definition of this fragment is based on the observation that, for many practical cases, full recursion is not necessary, but can rather be replaced by *linear* forms of recursion [111, 113] which is well-known in the context of plain Datalog. Formally, a Datalog query is *linear* if each of its rules has at most one occurrence of an intensional predicate in its body, where an *intensional predicate* is a predicate that appears in the head of some rule (other predicates are called *extensional*). For example, the rules

$$E(x,y) \rightarrow T(x,y), \quad T(x,y), T(y,z) \rightarrow T(x,z),$$

which compute the transitive closure of the extensional binary predicate E using non-linear recursion, can be rewritten as the set

$$E(x,y) \rightarrow T(x,y), \quad E(x,y), T(y,z) \rightarrow T(x,z),$$

which uses only linear recursion. More specifically, benchmark results for Vadalog showed that a large number of recursion employed in practice can be expressed by warded sets of rules that have only one body atom that is *mutually recursive* with a head atom. This type of recursion is also known in the Datalog literature, namely as *piecewise linear* recursion [2].

Based on this key observation, the following key question has immediately emerged: *does piecewise linear recursion provide an adequate restriction of warded sets of rules in order to achieve space-efficiency?* We answer this question positively, by showing that the query evaluation problem for OMQs based on piecewise linear warded sets of TGDs is PSpace-complete in combined, and, more importantly, NLogSpace-complete in data complexity. Moreover, we show that the wardedness condition cannot be omitted in this context, that is, we show that query evaluation for OMQs based on piecewise linear sets of TGDs is undecidable. As for the case of OMQs based on warded sets, we study the expressive power of the newly defined language, and we establish that OMQs based on piecewise linear warded sets of TGDs can, in fact, be always expressed as (piecewise) linear Datalog queries. For all these results, we strongly rely on the machinery of proof trees described above.

1.2 Road Map

This thesis is organized as follows:

▶ After this introductory chapter, we provide technical background information in Chapter 2. Besides introducing elementary notation, we provide technical definitions and notation for concepts from logic, databases, and automata that we are going to use throughout this thesis.

▶ We introduce the terminology concerning ontology-mediated querying in Chapter 3. In particular, we introduce the well-known chase procedure, and we formally introduce the most popular classes of TGDs for which query answering becomes decidable.

▶ Chapter 4 is dedicated to study the containment problem for OMQs. This chapter is divided into three main parts: in the first one, we study containment for OMQs that are UCQ-rewritable; in the second one, we study containment for OMQs based on (frontier-)guarded rules; and in the third one, we consider the containment problem when the left-hand side OMQ and the right-hand side OMQ fall into different languages.

▶ First-order rewritability for OMQs based on (frontier-)guarded rules is studied in Chapter 5. As mentioned above, we divide the study of this problem into two approaches, one based on classical automata techniques, and one based on cost automata. The former approach is non-optimal, but yields useful insights into the problem at hand, while the latter yields optimal decision procedures.

▶ In Chapter 6, we study proof trees for OMQs and use them to (i) re-establish the complexity of OMQs based on warded sets of rules, (ii) identify a space-efficient fragment of warded sets of rules, namely those warded sets that are piecewise linear, (iii) provide a strong sufficient criterion for OMQs to be expressible as Datalog queries, and (iv) show that OMQs based on (piecewise linear) warded sets of TGDs are expressible as (linear) Datalog queries.

▶ We conclude the thesis in Chapter 7. Some of the technical proofs are deferred to Appendix B.

Publications We want to mention that this thesis is mainly based on three publications which were published or accepted in the proceedings of conference venues:

[22] Pablo Barceló, Gerald Berger, and Andreas Pieris. Containment for Rule-Based Ontology-Mediated Queries. In *Proc. of the PODS*, pages 267–279, 2018.

[21] Pablo Barceló, Gerald Berger, Carsten Lutz, and Andreas Pieris. First-Order Rewritability of Frontier-Guarded Ontology-Mediated Queries. In *Proc. of the IJCAI*, pages 1707–1713, 2018.

[32] Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The Space-Efficient Core of Vadalog. To appear in PODS '19.

Chapter 4 is based on [22], Chapter 5 is based on [21], and Chapter 6 is based on [32]. The exposition of the contents of these publications is significantly more detailed than in the works cited, and the author of this thesis has a significant contribution to the contents of these papers.

CHAPTER 2

Background

In this chapter, we set up basic notions that will be used throughout this thesis. We assume that the reader is familiar with the basic notions of logic, database theory, and complexity theory. In Section 2.1 we agree upon basic mathematical notation, and in Section 2.2 we discuss the essential notions from logic that are used throughout this thesis. We continue in Section 2.3 to introduce the most common database query languages that we are going to encounter in this thesis. In Section 2.4 we introduce automata models that we are going to harness in later chapters.

2.1 Basic Notation

Before proceeding to give background on notions from logic and databases, let us fix some basic mathematical notation here which we are going to employ throughout the thesis:

▶ We usually denote sequences of objects $a_1, \ldots, a_n$ simply as a list as depicted. However, it will often be convenient to identify the sequence $a_1, \ldots, a_n$ with the tuple $(a_1, \ldots, a_n)$, and we make no formal distinction between these two kinds of objects. To indicate that an object is a sequence or a tuple, we employ the convention that we (usually) place a bar above an identifier, e.g., $\bar{a}$, $\bar{b}$, etc. For a sequence $\bar{a}$, we write $[\bar{a}]$ to denote the set containing precisely the elements present in $\bar{a}$, and we write $|\bar{a}|$ to denote the length of the sequence $\bar{a}$.

▶ If $f\colon A \to B$ is a function then, for $A_0 \subseteq A$, we denote by $f \restriction A_0$ the restriction of f to A_0. For a partial function f, we denote by $\mathsf{dom}(f)$ its domain.

▶ A *substitution* is just a mapping from one set to another set. A substitution over a (mostly implicit) domain U is often specified via an expression of the form $\{x_i \mapsto t_i\}_{1 \leq i \leq n}$, which stands for the substitution ρ such that $\rho(x_i) = t_i$ and that is the identity on all elements of $U \setminus \{x_1, \ldots, x_n\}$. Given an expression e (e.g., a formula, a fact, etc.) and a substitution ρ, we denote by $\rho(e)$ the result of simultaneously replacing each occurrence

of an object t in e uniformly by $\rho(t)$, provided t occurs in the domain of ρ. If $t_1, \ldots, t_n$ are arbitrary objects are indicated, then we denote by $x_1/t_1, \ldots, x_n/t_n$ the substitution $\{x_i \mapsto t_i\}_{1 \leq i \leq n}$. If $\bar{x} = x_1, \ldots, x_n$ and $\bar{t} = t_1, \ldots, t_n$ are given in sequence-notation, we may express $x_1/t_1, \ldots, x_n/t_n$ simply by $\bar{x}/\bar{t}$. If $e \coloneqq e(x_1, \ldots, x_n)$ is an expression where occurrences of subexpressions $x_1, \ldots, x_n$ are explicitly indicated, then we denote by $e(x_1/t_1, \ldots, x_n/t_n)$ the result of applying the substitution $\{x_i \mapsto t_i\}_{1 \leq i \leq n}$ to e. If the $x_1, \ldots, x_n$ are clear from context, then we may omit them and denote $e(x_1/t_1, \ldots, x_n/t_n)$ simply by $e(t_1, \ldots, t_n)$.

▶ A *(directed) tree* is a graph $\mathcal{T} = (T, E)$, where T is a set of *nodes* and E a set $E \subseteq T \times T$ of *edges* such that $\mathcal{T}$ is connected and has no cycles. We usually denote trees by calligraphic letters, and denote by a Latin version of that letter its set of nodes (e.g., T denotes the set of nodes of $\mathcal{T}$). We call $\mathcal{T}$ *rooted* if there is a distinguished element $\varepsilon_{\mathcal{T}} \in T$, called the *root* of $\mathcal{T}$. We usually assume that $\mathcal{T}$ is rooted, and we always denote by $\varepsilon_{\mathcal{T}}$ its root and omit the subscript "$\mathcal{T}$" whenever clear from context. Given nodes $v, w \in T$, we write $v \prec_{\mathcal{T}} w$ if v is a proper ancestor of w, and we write $v \preceq_{\mathcal{T}} w$ if $v \prec_{\mathcal{T}} w$ or $v = w$. Again, we omit the subscript "$\mathcal{T}$" whenever clear from context. The *child nodes* (or *children*) of a node $v \in T$ is the set of all w such that vEw. A node is a *leaf node* if it has no children. An *(undirected) path* in $\mathcal{T}$ is a sequence $\pi = v_1, \ldots, v_n$ of nodes such that every node in the sequence is succeeded by a neighboring node, i.e., a parent or a child node. The *length* of π is $n - 1$. A *branch* of $\mathcal{T}$ is a path $v_1, \ldots, v_n$, where $v_1 = \varepsilon_{\mathcal{T}}$, $v_i E v_{i+1}$ $(i = 1, \ldots, n-1)$, and v_n is a leaf node.

▶ We assume familiarity with the basic concepts of complexity theory, and refer the reader to [8, 115] for comprehensive introductions. Given complexity classes $\mathcal{C}$ and $\mathcal{O}$, we denote by $\mathcal{C}^{\mathcal{O}}$ the class of all problems that can be solved in $\mathcal{C}$ when having unrestricted access to an oracle in $\mathcal{O}$.

▶ All the finitary objects that can be passed as inputs to Turing machines are, unless stated otherwise, assumed to be reasonably encoded by $\{0,1\}$-strings in an "efficient" way (see, e.g., [71, 94]). For example, numbers are encoded in binary representation. Given any such object o, we denote by $\|o\|$ the size of its encoding, i.e., the number of symbols used to encode o. If o is a finite set, this is to be distinguished from the cardinality of o, i.e., its number of elements, which we denote by $|o|$.

▶ Given a natural number $n \geq 0$, we write $[n]$ for the set $\{1, \ldots, n\}$.

2.2 Background from Logic and Databases

We assume basic knowledge of the syntax and semantics of first-order logic, a comprehensive introduction can be found in, e.g., [119]. For a more database-oriented treatment of logical concepts, we refer the reader to the book [102].

2.2.1 Relational Schemas

All the formulas and logical objects we are going to consider in this thesis are formulated over signatures that do not have function symbols. Moreover, signatures will not explicitly contain constants, but instead we assume a globally fixed countably infinite set of

constants const at our disposal. Throughout the thesis, we treat the elements of const as distinguished ones, i.e., unless stated otherwise, no other named mathematical object is equal to an element of const. The first-order signatures we henceforth consider carry neither function nor constant symbols and they are thus purely relational. Moreover, we always consider them to be finite. In database parlance, we shall call such signatures *(relational) schemas* and we will denote them by boldface letters. Formally, a relational schema is thus a finite set $\mathbf{S} = \{R_1, \ldots, R_k\}$ of *relation names* (or simply *predicates*) R_i $(i = 1, \ldots, k)$, each having an associated non-negative *arity*. We shall often write R/n to indicate that the relation name R has arity $n \geq 0$. The *width* of $\mathbf{S}$, denoted $\mathsf{wd}(\mathbf{S})$, is the maximum arity of all relation symbols contained in it.

2.2.2 First-Order Formulas

All the logical objects that we are going to study in this thesis are, from a syntactic point of view, built up using first-order formulas. It is therefore useful to fix some notation concerning first-order logic in the following.

For all the syntactic objects that are formulated using first-order logic, we assume that we are given a countably infinite set of *variables* vars at our disposal, which we assume to be disjoint from const. A *term* is a constant from const or a variable. Since all our signatures are purely relational, the only *atomic formulas* (or simply *atoms*) available are either *relational atoms* of the form $R(t_1, \ldots, t_n)$, where each t_i is a term, or *equality atoms* of the form $t_1 = t_2$, where t_1 and t_2 are terms. We respectively call $t_1, \ldots, t_n$ and t_1, t_2 the *arguments* of $R(t_1, \ldots, t_n)$ and $t_1 = t_2$. Unless stated otherwise, we henceforth explicitly allow the use of equality atoms in formulas. First-order formulas over a schema $\mathbf{S}$ are built up in the usual manner: we make use of the standard connectives such as negation $(\neg)$, logical and $(\wedge)$, logical or $(\vee)$, implication $(\rightarrow)$, equivalence $(\leftrightarrow)$, universal quantification $(\forall)$, and existential quantification $(\exists)$. We are often going to abbreviate a sequence of quantifiers $\mathsf{Q}x_1 \cdots \mathsf{Q}x_n$ $(\mathsf{Q} \in \{\forall, \exists\})$ by $\mathsf{Q}x_1, \ldots, x_n$. The notions of *bound* and *free* occurrences of variables are defined in the usual manner. A *sentence* is a formula that has no free occurrence of a variable. We shall use $\top$ as an abbreviation for the formula $\exists x\, x = x$, and $\bot$ as an abbreviation for $\neg\top$. We denote the class of all first-order formulas by FO.

Given a formula φ, we shall denote φ by $\varphi(x_1, \ldots, x_n)$ to indicate that the free variables of φ are among $x_1, \ldots, x_n$. We say that φ *has the variables* $x_1, \ldots, x_n$ *as indicated* if the free variables of φ are exactly given by $x_1, \ldots, x_n$. Moreover, we write $\mathsf{free}(\varphi)$ to denote the set of all free variables of φ.

Remark 2.1. We want to recall that $\varphi(x_1/t_1, \ldots, x_n/t_n)$ denotes the object that results from φ by uniformly substituting each x_i by t_i. Hence, if the t_i are terms, then $\varphi(x_1/t_1, \ldots, x_n/t_n)$ will be a formula again. We do not impose any restrictions on the substitutions used for formulas, since we will only use them in a very basic fashion, and thus problems that may arise in using arbitrary substitutions do not concern us (for example, substituting y for x in the formula $\varphi(x) = \exists y\, y \neq x$ would yield the unsatisfiable formula $\exists y\, y \neq y$).

2.2.3 Structures and Databases

Let $\mathbf{S}$ be a relational schema and U be a set of objects. For arbitrary objects $t_1, \ldots, t_n \in U$ and $R \in \mathbf{S}$, we call an expression α of the form $R(t_1, \ldots, t_n)$ an $\mathbf{S}$-*fact* or a *fact over* $\mathbf{S}$, and we call $t_1, \ldots, t_n$ the *arguments* of α. For a set of $\mathbf{S}$-facts F, the *active domain* of F, denoted $\mathsf{adom}(F)$, is the set of all objects that occur as arguments in facts.

Definition 2.2. An $\mathbf{S}$-*structure* is a tuple $\mathfrak{A} = (A, \cdot^{\mathfrak{A}})$, where A is a non-empty set (the *domain of* $\mathfrak{A}$) and $\cdot^{\mathfrak{A}}$ a function that assigns an n-ary relation $R^{\mathfrak{A}} \subseteq A^n$ to every relation name R/n of $\mathbf{S}$.

Facts $R(a_1, \ldots, a_n)$ over $\mathbf{S}$ such that $(a_1, \ldots, a_n) \in R^{\mathfrak{A}}$ are called *facts of* $\mathfrak{A}$. The *active domain* of $\mathfrak{A}$, denoted $\mathsf{adom}(\mathfrak{A})$, is the active domain of its set of facts. We call a structure *finite* if its domain is finite. For a finite $\mathfrak{A}$, we write $|\mathfrak{A}|$ for the number of its facts.

Notation. If the schema $\mathbf{S}$ is not explicitly given, we write $\mathsf{sig}(\mathfrak{A})$ for $\mathbf{S}$. Moreover, if the domain is not explicitly named, we will denote it by $\mathsf{dom}(\mathfrak{A})$.

Assignments and modelhood Notice that $\mathbf{S}$-structures do *not* interpret constants via the function $\cdot^{\mathfrak{A}}$. Instead, their meaning is implicitly given by the constant names themselves. Let us formalize this more carefully.

A *(variable) assignment over* $\mathfrak{A}$ is a function π that assigns values from A to variables. We will often consider variable assignments to be partial functions, which are defined only on the relevant variables for the specific purpose. For an $\mathbf{S}$-structure $\mathfrak{A}$, a term t (i.e., a variable or a constant from const), we define a function $[\![\cdot]\!]_\pi$ as follows:

$$[\![\cdot]\!]_\pi \colon t \longmapsto \begin{cases} \pi(t), & \text{if } t \in \mathsf{vars} \text{ is a variable,} \\ t, & \text{if } t \in \mathsf{const} \text{ is a constant.} \end{cases}$$

Thus, we employ the *unique name assumption* on elements from const, i.e., different constants denote different elements – more precisely, they are always interpreted as themselves.

Given an atom $R(t_1, \ldots, t_n)$, we specify that

$$\mathfrak{A}, \pi \models R(t_1, \ldots, t_n) \iff_{df} ([\![t_1]\!]_\pi, \ldots, [\![t_n]\!]_\pi) \in R^{\mathfrak{A}}.$$

Likewise, for an equality atom $t_1 = t_2$, we write $\mathfrak{A}, \pi \models t_1 = t_2$ iff $[\![t_1]\!]_\pi = [\![t_2]\!]_\pi$ holds. The evaluation of a first-order formula φ on $\mathfrak{A}$ and π is then defined recursively in the usual manner. We write $\mathfrak{A}, \pi \models \varphi$ to indicate that φ is satisfied in $\mathfrak{A}$ under π. We call pairs of the form $(\mathfrak{A}, \pi)$ $\mathbf{S}$-*interpretations*, and if $\mathfrak{A}, \pi \models \varphi$, we call the interpretation $(\mathfrak{A}, \pi)$ a *model* of φ. If φ is a sentence, then we omit the assignment π and simply call $\mathfrak{A}$ a model of φ. If $\varphi = \varphi(x_1, \ldots, x_n)$ has the free variables $x_1, \ldots, x_n$ as indicated, it will often be convenient to treat assignments as anonymous objects. Henceforth, for $a_1, \ldots, a_n \in \mathsf{dom}(\mathfrak{A})$, we may thus write $\mathfrak{A} \models \varphi(a_1, \ldots, a_n)$ in case $\mathfrak{A}, \pi \models \varphi(x_1, \ldots, x_n)$ holds, where $\pi \colon x_i \mapsto a_i$. We shall also call the tuple $(\mathfrak{A}, a_1, \ldots, a_n)$ a *model* of $\varphi(x_1, \ldots, x_n)$. We remark that the notation $\mathfrak{A} \models \varphi(a_1, \ldots, a_n)$ may cause some ambiguity: if $a_1, \ldots, a_n \in \mathsf{const}$, then

$\varphi(a_1, \ldots, a_n)$ may stand for the sentence $\varphi(x_1/a_1, \ldots, x_n/a_n)$. However, in this case, $\mathfrak{A}, \pi \models \varphi(x_1, \ldots, x_n)$, where $\pi\colon x_i \mapsto a_i$, holds iff $\mathfrak{A} \models \varphi(x_1/a_1, \ldots, x_n/a_n)$. Hence, we can safely apply the notion of anonymous assignments in this case.

Databases vs. structures In the database context, it is usually the case that one defines databases simply as sets of facts. We adopt a more logical notion:

Definition 2.3. An $\mathbf{S}$-*database* is an $\mathbf{S}$-structure whose domain is finite and consists solely of elements from const.

The database-oriented reader may ask why we did not simply define databases as set of facts. The reason is a formal one. Sets of facts do not explicitly name a non-empty domain of discourse, but their domains are rather implicitly given through the active domain of the facts they name. However, some notions defined in this thesis require an explicitly given non-empty domain of discourse, since the omission of such would render these notions ill-defined. Hence, we shall always formally consider databases to be proper structures. However, to simplify presentation, we follow the convention that we are allowed to declare structures (and thus databases) as sets of facts:

Notation. If we specify an $\mathbf{S}$-structure $\mathfrak{A}$ as a set of facts F over a schema $\mathbf{S}$, then we mean by $\mathfrak{A}$ the following structure:

- The domain of $\mathfrak{A}$ equals $\mathsf{adom}(F)$ in case $\mathsf{adom}(F) \neq \varnothing$. Otherwise, we specify that $\mathsf{dom}(\mathfrak{A}) = \{d\}$, where d is a globally fixed "dummy" element.
- For all $R/n \in \mathbf{S}$ and all $a_1, \ldots, a_n \in \mathsf{adom}(F)$ we have that

$$(a_1, \ldots, a_n) \in R^{\mathfrak{A}} \iff R_i(a_1, \ldots, a_n) \in F.$$

This convention ensures that whenever we specify $\mathfrak{A}$ as a set of facts, its domain will never be empty. The structure specified is called the *structure corresponding to* F. If $\mathfrak{D}$ is an $\mathbf{S}$-database, then we also say that a structure $\mathfrak{B}$ is a *model* of $\mathfrak{D}$ in case every fact of $\mathfrak{D}$ is also a fact of $\mathfrak{B}$.

Operations on structures If no ambiguity arises, we will often abuse notation and treat structures and databases as sets of facts. Given two structure $\mathfrak{A}$ and $\mathfrak{B}$, we will often use set-based notations to state properties of the sets of facts of $\mathfrak{A}$ and $\mathfrak{B}$. We write $\mathfrak{A} \subseteq \mathfrak{B}$ to indicate that every fact of $\mathfrak{A}$ is also a fact of $\mathfrak{B}$. Likewise, we denote by $\mathfrak{A} \cap \mathfrak{B}$ the structure corresponding to the intersection of the set of facts of $\mathfrak{A}$ and the set of facts of $\mathfrak{B}$. According notation is employed for other operations on sets.

Definition 2.4. Let $\mathfrak{B}$ be an $\mathbf{S}$-structure. We say that the $\mathbf{S}$-structure $\mathfrak{A}$ is the *substructure of* $\mathfrak{B}$ *induced by* $X \subseteq \mathsf{dom}(\mathfrak{B})$ if (i) $X = \mathsf{dom}(\mathfrak{A}) \subseteq \mathsf{dom}(\mathfrak{B})$, and (ii) $R^{\mathfrak{A}} = X^n \cap R^{\mathfrak{B}}$ for every $R/n \in \mathbf{S}$. We write $\mathfrak{B} \restriction X$ for the substructure of $\mathfrak{B}$ induced by X.

If X is empty, then we let $\mathfrak{B} \restriction X$ be the structure whose domain consists of the globally fixed element d, and that has all relations empty.

Definition 2.5. Let $\mathfrak{A}$ be a $\mathbf{T}$-structure and $\mathbf{S} \subseteq \mathbf{T}$. The $\mathbf{S}$-*retract* of $\mathfrak{A}$ is the $\mathbf{S}$-structure $\mathfrak{A} \restriction \mathbf{S}$ such that $\mathsf{dom}(\mathfrak{A} \restriction \mathbf{S}) := \mathsf{dom}(\mathfrak{A})$ and $R^{\mathfrak{A}\restriction\mathbf{S}} = R^{\mathfrak{A}}$ for all $R \in \mathbf{S}$. We also say that $\mathfrak{A} \restriction \mathbf{S}$ is the *result of restricting* $\mathfrak{A}$ *to* $\mathbf{S}$.

A $\mathbf{T}$-structure $\mathfrak{B}$ is an *expansion* of an $\mathbf{S}$-structure $\mathfrak{A}$, if $\mathbf{S} \subseteq \mathbf{T}$ and $\mathfrak{B} \restriction \mathbf{S} = \mathfrak{A}$.

Definition 2.6. Let $\mathfrak{A}$ and $\mathfrak{B}$ be two structures. A *weak homomorphism from* $\mathfrak{A}$ *to* $\mathfrak{B}$ is a function $h\colon \mathsf{dom}(\mathfrak{A}) \to \mathsf{dom}(\mathfrak{B})$ such that

$$(a_1, \ldots, a_k) \in R^{\mathfrak{A}} \implies (h(a_1), \ldots, h(a_k)) \in R^{\mathfrak{B}},$$

for all $a_1, \ldots, a_k \in \mathsf{dom}(\mathfrak{A})$ and all $R \in \mathsf{sig}(\mathfrak{A})$. We call h a *homomorphism* if it is the identity on const.

The image $h(\mathfrak{A})$ of $\mathfrak{A}$ under a (weak) homomorphism h is the structure whose domain consists of $h(\mathsf{dom}(\mathfrak{A}))$ and that has $R^{h(\mathfrak{A})} := R^{\mathfrak{B}} \cap \mathsf{dom}(h(\mathfrak{A}))^n$ for all $R/n \in \mathsf{sig}(\mathfrak{A})$.

Definition 2.7. We say that two structures $\mathfrak{A}$ and $\mathfrak{B}$ are *homomorphically equivalent*, if there exists a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ and a homomorphism from $\mathfrak{B}$ to $\mathfrak{A}$.

Definition 2.8. We say that two structures are *(weakly) isomorphic*, if there exists a bijective function $f\colon \mathsf{dom}(\mathfrak{A}) \to \mathsf{dom}(\mathfrak{B})$ such that h is a (weak) homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ and f^{-1} a (weak) homomorphism from $\mathfrak{B}$ to $\mathfrak{A}$. Such a function f is called a *(weak) isomorphism* between $\mathfrak{A}$ and $\mathfrak{B}$.

Logical entailment Given two formulas $\varphi(x_1, \ldots, x_n)$ and $\psi(y_1, \ldots, y_m)$, we write $\varphi \models \psi$ in case the first-order sentence

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_m(\varphi \to \psi)$$

is true in every model. We write $\varphi \equiv \psi$ if $\varphi \models \psi$ and $\psi \models \varphi$ both hold. For a set of sentences Γ, we write $\Gamma \models \varphi$ if every model of Γ is also a model of φ. For a structure $\mathfrak{A}$ and an assignment π over $\mathfrak{A}$, we write $(\mathfrak{A}, \Gamma), \pi \models \varphi(x_1, \ldots, x_n)$, if for every interpretation $(\mathfrak{B}, \pi)$ such that $\mathfrak{B} \supseteq \mathfrak{A}$ it holds that, if $\mathfrak{B}$ is a model of Γ, then $(\mathfrak{B}, \pi)$ is also a model of $\varphi(x_1, \ldots, x_n)$. In this context we also apply the conventions for anonymous assignments as in the case of standard modelhood, i.e., if $\pi\colon x_i \mapsto a_i$ $(i = 1, \ldots, n)$, then we write $(\mathfrak{A}, \Gamma) \models \varphi(a_1, \ldots, a_n)$ for $(\mathfrak{A}, \Gamma), \pi \models \varphi(x_1, \ldots, x_n)$.

2.3 Database Query Languages

In this section, we are going to review the most common logic-based database query languages that we are going to encounter in this thesis. For more extensive presentations, we refer the interested reader to [1, 102].

Definition 2.9. Let $\mathbf{S}$ be a relational schema and let $k \geq 0$. A *k-ary $\mathbf{S}$-query* Q is a function that maps $\mathbf{S}$-databases to k-tuples over const. We say that Q is *Boolean* in case $k = 0$. The set of tuples $Q(\mathfrak{D})$ is called the *evaluation of Q over $\mathfrak{D}$* and tuples contained in $Q(\mathfrak{D})$ are called *answer tuples of Q to $\mathfrak{D}$*.

Notation. We also write $\mathfrak{D} \models Q(a_1, \ldots, a_k)$ to indicate that $(a_1, \ldots, a_k) \in Q(\mathfrak{D})$. In case Q is Boolean, we simply write $\mathfrak{D} \models Q$ to indicate that the empty tuple is an answer tuple of Q to $\mathfrak{D}$.

Notice that the notion of **S**-query is a genuinely semantic one and is *a priori* not linked to any syntactic formalism at all. In the following we are going to briefly introduce the most relevant *query languages* that we will encounter in this thesis. Formally, a query language $\mathcal{L}$ is simply a formal language such that for every $\varphi \in \mathcal{L}$, there exists a unique **S**-query Q_φ, where **S** depends on φ. We say that φ *defines* Q_φ or that Q_φ is the *semantics* of φ. The mapping Q_φ can be viewed as the semantics of φ in the database context, as it specifies how answers to a database $\mathfrak{D}$ are obtained from the specification φ.

Remark 2.10. We want to mention at this point that we will henceforth not make a pedantic distinction between the semantic notion of **S**-query and the syntactic object that defines it. In particular, when a syntactic object φ defines Q_φ, we shall overload notation and use φ both for the syntactic object, and for the query Q_φ. Thus, whenever unambiguous, $\varphi(\mathfrak{D})$ will in this case also denote the set of answer tuples of Q_φ to $\mathfrak{D}$.

One of the most relevant computational problems concerning a query language $\mathcal{L}$ is arguably the problem which asks whether a tuple is contained in the evaluation of a query over some given database. We define the problem $\mathsf{Eval}(\mathcal{L})$ as follows:

PROBLEM:	$\mathsf{Eval}(\mathcal{L})$
INPUT:	A query $\varphi \in \mathcal{L}$ such that Q_φ is a k-ary **S**-query, an **S**-database $\mathfrak{D}$, and a tuple $(a_1, \ldots, a_k) \in \mathsf{adom}(\mathfrak{D})$.
QUESTION:	Is it the case that $(a_1, \ldots, a_k) \in Q_\varphi(\mathfrak{D})$?

$\mathsf{Eval}(\mathcal{L})$ is called the *query answering problem for* $\mathcal{L}$ or the *(query) evaluation problem for* $\mathcal{L}$. We remark that $\mathsf{Eval}(\mathcal{L})$ does not ask for the enumeration of *all* the answer tuples of $Q_\varphi(\mathfrak{D})$, and hence $\mathsf{Eval}(\mathcal{L})$ is sometimes called the *query-of-tuple problem*. When $\mathsf{Eval}(\mathcal{L})$ permits all inputs as stated above, then we speak about the *combined complexity* of the query answering problem for $\mathcal{L}$. The complexity of the problem which considers the query $\varphi \in \mathcal{L}$ as fixed is the *data complexity* of the query answering problem for $\mathcal{L}$ (see [130]).

In the following we are going to revisit the most important standard database query languages that we will encounter in this thesis. Before presenting them, let us introduce one additional natural notion:

DEFINITION 2.11. We say that two **S**-queries Q_1 and Q_2 are *equivalent*, written $Q_1 \equiv Q_2$, if $Q_1(\mathfrak{D}) = Q_2(\mathfrak{D})$ for all **S**-databases $\mathfrak{D}$.

2.3.1 First-Order Queries

Given a first-order formula $\varphi(x_1, \ldots, x_n)$ over a schema **S**, the query Q_φ defined by φ is given by

$$Q_\varphi \colon \mathfrak{D} \longmapsto \{(a_1, \ldots, a_n) \mid \mathfrak{D} \models \varphi(a_1, \ldots, a_n), \text{ where } a_1, \ldots, a_n \in \mathsf{adom}(\mathfrak{D})\}.$$

In the context of queries, we call the sequence of variables $x_1, \ldots, x_n$ the *answer variables* of φ. The class of all such queries is the class of *first-order queries*.

Remark 2.12. Notice that the definition of Q_φ as given above is strictly speaking not unique, since it depends on the order of the answer variables $x_1, \ldots, x_n$. However, in the context of queries we will implicitly always assume a canonical lexicographic order on all possible variables to avoid ambiguity. When writing down first-order queries or queries that fall into the class of first-order queries such as CQs and UCQs (see below), we always assume that all the free variables are indicated, provided we mention a list of free variables at all.

It is well-known that, roughly speaking, first-order queries correspond to relational algebra queries (see, e.g., [1]). Moreover, first-order queries find a practical realization in standard relational database management systems though the arguably most popular query language SQL.

The following theorem summarizes the rather well-known complexity results of answering first-order queries [93, 130]:[1]

Theorem 2.13. *Answering first-order queries is* PSpace*-complete in combined complexity, and in* AC_0 *(and thus in* LogSpace*) in data complexity.*

2.3.2 (Unions of) Conjunctive Queries

A *conjunctive query* (CQ) over a schema $\mathbf{S}$ is a first-order formula $q(\bar{x})$ that takes the form $\exists \bar{y}\,(\alpha_1 \wedge \cdots \wedge \alpha_n)$, where each α_i is an atomic formula, i.e., either a relational atom or an equality atom. Given a set of atoms A, we write $\mathsf{var}(A)$ for the set of all variables mentioned in A. We write $\mathsf{var}(q)$ for $\mathsf{var}(\{\alpha_1, \ldots, \alpha_n\})$. We call the atoms $\alpha_1, \ldots, \alpha_n$ the *body atoms* of q, and we write $\mathsf{body}(q)$ for the set of its body atoms. It will, however, often be convenient to abuse notation and consider CQs as the sets of its constituent body atoms, and we often write $|q|$ for the number of body atoms of q. The query defined by $q(\bar{x})$ is defined exactly as in the case of first-order queries. Again, we call the sequence of variables $\bar{x}$ the *answer variables* of $q(\bar{x})$, and we say that $q(\bar{x})$ is *Boolean* (for short, a BCQ) if it is a sentence, i.e., if its sequence of answer variables is empty. Recall that the formula $\top$ (which equals $\exists x\, x = x$) is a CQ according to this definition – we shall call $\top$ the *empty* CQ.[2] We call q *atomic* if $\bar{y}$ is empty and q consists of a single body atom only. Moreover, we call q *constant-free* if it does not have any occurrence of a constant.

Notation. We denote by CQ the class of all conjunctive queries, and by BCQ the class of all Boolean conjunctive queries. We write AQ for the class of all atomic queries, and BAQ for the class of all Boolean atomic queries – notice that elements from BAQ consist of a single 0-ary relation symbol only.

[1] AC_0 consists of all functions computable by families of circuits of constant depth and polynomial size that have unlimited fan-in AND- and OR-gates, possibly with NOT-gates that are placed at the input gates (see, e.g., [9, 94]).

[2] We use the term "empty" because $\top$ can be viewed as a conjunction over the empty set of atoms.

Remark 2.14. Let us make a few more remarks on the use of equality atoms in CQs. It is not hard to see that, from a purely logical perspective, equality atoms can always be eliminated from CQs by identifying variables accordingly. That is, for any CQ q that is not logically equivalent to $\top$ and which uses equality atoms, there is a CQ q' that results from q by identifying variables such that q and q' are equivalent in the sense of first-order logic (the assumption that q is not logically equivalent to $\top$ is needed because $\top$ uses equality by its definition).

However, equality atoms are useful to duplicate answer variables and are required to maintain equivalence on the level of queries (cf. Definition 2.11). For example, the CQ $q(x, y) \coloneqq \exists z\,(R(x, z) \wedge x = y)$ has as sequence of answer variables the sequence (x, y), and all answer tuples of q to any database must be of the form (a, a). The use of equality allows now us to genuinely express queries *via* first-order formulas – indeed, eliminating the equality atom $x = y$ from q yields the CQ $q' \coloneqq \exists z\, R(x, z)$, which has a single answer variable x. However, by our definition of CQs as plain first-order formulas, there is no way for us to demand that the query *defined by* q' has two answer variables such that the first matches the second. Thus, although q and q' are equivalent in the sense of first-order logic – the sentence $\forall x, y\,(q(x, y) \leftrightarrow q'(x))$ is clearly true in all models – the queries these two CQs define are not equivalent in the sense of Definition 2.11.

Homomorphisms and modelhood Let $\mathfrak{A}$ be an $\mathbf{S}$-structure. Given a set of atoms B, a *homomorphism from B to $\mathfrak{A}$* is a function $h\colon \mathsf{var}(B) \to \mathsf{dom}(\mathfrak{A})$ such that, when h is considered as an assignment, $\mathfrak{A}, h \models \alpha$ for all $\alpha \in B$. For a CQ $q(\bar{x})$, a *homomorphism from $q(\bar{x})$ to $\mathfrak{A}$* is a homomorphism from $\mathsf{body}(q(\bar{x}))$ to $\mathfrak{A}$. That is, a homomorphism from $q(\bar{x})$ to $\mathfrak{A}$ is simply a variable assignment that covers all variables occurring in $q(\bar{x})$. It is immediate by the semantics of first-order logic that $\mathfrak{A} \models q(\bar{a})$ iff there is a homomorphism h from q to $\mathfrak{A}$ such that $h(\bar{x}) = \bar{a}$. Homomorphisms provide a convenient alternative for defining the evaluation $q(\mathfrak{D})$ of $q(\bar{x})$ over $\mathfrak{D}$: for all tuples $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$,

$$\bar{a} \in q(\mathfrak{D}) \iff \text{there is a homomorphism } h \text{ from } q(\bar{x}) \text{ to } \mathfrak{D} \text{ such that } h(\bar{x}) = \bar{a}.$$

Notice that if α_i is a body atom of $q(\bar{x})$ that contains only constants as arguments, then there exists a homomorphism from $q(\bar{x})$ to $\mathfrak{D}$ only if $\alpha_i \in \mathfrak{D}$. Given a homomorphism h from $q(\bar{x})$ to $\mathfrak{A}$, we write $h(\alpha_i)$ for the fact that emerges from α_i by replacing each of its arguments by their images under h (i.e., we may treat h as a substitution), and a similar convention is applied for sets of atoms. Accordingly, when writing $h(q)$, we mean the set of facts $\{h(\alpha_1), \ldots, h(\alpha_n)\}$, where $\alpha_1, \ldots, \alpha_n$ is an enumeration of the body atoms of $q(\bar{x})$ (a similar convention is employed for sets of atoms). In this context, we may often abuse notation and write expressions like $h(q) \subseteq \mathfrak{A}$ to indicate that, for $i = 1, \ldots, n$, $\mathfrak{A}, h \models \alpha_i$ holds.

CQs as structures It will often be convenient to view CQs as finite structures and vice versa. While a finite structure $\mathfrak{A}$ has an obvious correspondence to a CQ simply by taking the sets of facts of $\mathfrak{A}$ as body atoms – and considering domain elements of $\mathfrak{A}$ that are not in const as variables –, more care must be taken when passing from a CQ

q to a structure due to the use of equality atoms in q. We will formally describe this correspondence in the following.

Let $q(x_1, \ldots, x_n)$ be a conjunctive query over $\mathbf{S}$. For terms s and t occurring in q, i.e., variables or constants, we define $s \sim t$ if there is an equality atom $s = t$ in the body of q. Moreover, assume that $\sim$ is reflexively, symmetrically, and transitively closed, i.e., (i) $s \sim s$ for all terms s occurring in q, (ii) $s \sim t$ implies $t \sim s$, and (iii) $s \sim t$ and $t \sim u$ implies $s \sim u$. Obviously, $\sim$ is an equivalence relation on the terms occurring in q, and we denote by $[s]_q$ the equivalence class of s according to $\sim$. Due to the unique name assumption on constants, notice that $q(x_1, \ldots, x_n)$ does not have a model iff there is a term s occurring in q such that $[s]_q$ contains two distinct constants. Notice also that a constant-free CQ obviously always possesses models.

Suppose now that $q(x_1, \ldots, x_n)$ has a model. We write $\mathfrak{A}_q$ for the $\mathbf{S}$-structure whose domain consists of the set of equivalence classes $[x]_q$, where $x \in \mathsf{var}(q)$, plus the set of all constants occurring in q. For $R/m \in \mathbf{S}$, we define $\mathfrak{A}_q \models R(\alpha_1, \ldots, \alpha_m)$ iff there are terms $s_1, \ldots, s_m$ such that $R(s_1, \ldots, s_m)$ is a body atom of q, and, for $i = 1, \ldots, m$, one of the following is true:

(i) $\alpha_i = [x]_q$, for some $x \in \mathsf{var}(q)$, and $s_i \in [x]_q$.

(ii) α_i is a constant and $s_i = \alpha_i$.

It is not hard to check that $\mathfrak{A}_q$ is well-defined. Moreover, notice that the domain of $\mathfrak{A}_q$ is not empty, even in case q equals the empty CQ $\top$, which equals $\exists x\, x = x$ by definition. We call $\mathfrak{A}_q$ the *structure corresponding to* q, and it can be easily shown that, for any structure $\mathfrak{B}$,

$$\mathfrak{B} \models q(a_1, \ldots, a_n) \iff \begin{array}{l}\text{there is a homomorphism } h \text{ from } \mathfrak{A}_q \text{ to } \mathfrak{B} \\ \text{such that } h([x_i]_q) = a_i, \text{ for all } i \in [n].\end{array}$$

Moreover, for two CQs $q(x_1, \ldots, x_n)$ and $p(x_1, \ldots, x_n)$, it holds that $q(x_1, \ldots, x_n) \models p(x_1, \ldots, x_n)$ iff there exists a homomorphism h from $\mathfrak{A}_p$ to $\mathfrak{A}_q$ such that $h([x_i]_p) = [x_i]_q$ for all $i \in [n]$.

Unions of conjunctive queries (UCQs) A *union of conjunctive queries* (UCQ) over a schema $\mathbf{S}$ is a finite disjunction $q(\bar{x}) = \bigvee_{i=1}^{n} q_i(\bar{x})$ of CQs $q_1, \ldots, q_n$ over $\mathbf{S}$ that all have the same sequence of answer variables. We say that $q(\bar{x})$ is a *union of Boolean conjunctive queries* (UBCQ) if each q_i is Boolean. The class of all UCQs is denoted by UCQ. The *query defined by* $q(\bar{x})$ is again defined exactly as in the case of first-order queries. It is easy to check that

$$q(\mathfrak{D}) = \bigcup_{i=1}^{n} q_i(\mathfrak{D}).$$

Hence, $\bar{a} \in q(\mathfrak{D})$ iff there is an $i \in [n]$ and a homomorphism h from q_i to $\mathfrak{D}$ such that $h_i(\bar{x}) = \bar{a}$. We call $\bar{x}$ the *answer variables* of q and we say that q is *constant-free* iff every q_i is constant-free.

Notation. We write UCQ for the class of all UCQs, and UBCQ for the class of all unions of BCQs.

Complexity of answering (U)CQs Since answering CQs amounts to the check for the existence of a homomorphism, and since verifying whether a given function is indeed a homomorphism is feasible in polynomial time, the fact that answering CQs is feasible in NP follows – this is a celebrated result by Chandra and Merlin [54]. Likewise, answering UCQs amounts to guessing one CQ that is a disjunct of the UCQ at hand, and then verifying the existence of a homomorphism from that disjunct to the given database. Establishing a matching NP lower bound is folklore, whence:

THEOREM 2.15. *The query answering problem for (unions of) conjunctive queries is* NP*-complete in combined complexity.*

Naturally, as (U)CQs are also first-order queries, the data complexity of answering (U)CQs is in AC_0 as well.

2.3.3 DATALOG QUERIES

A *Datalog rule* τ is an expression of the form

$$\alpha_1, \ldots, \alpha_n \to \beta,$$

where $\alpha_1, \ldots, \alpha_n$ are atoms and β is a relational atom[3] such that every variable from β also appears in one of the α_i. We call β the *head (atom)* of τ and $\alpha_1, \ldots, \alpha_n$ the *body (atoms)* of τ. Let $\varphi \coloneqq \alpha_1 \wedge \cdots \wedge \alpha_n$, and suppose $\beta = \beta(\bar{y})$ and $\varphi = \varphi(\bar{x}, \bar{y})$ with all formulas having the free variables as indicated. The *CQ corresponding to the body of* τ is the CQ $\exists \bar{y}\, \varphi(\bar{x}, \bar{y})$. Whenever unambiguous, we identify the body of τ with the CQ corresponding to it.

A *Datalog program* is a set of Datalog rules. A *Datalog query* is a tuple $Q = (\Pi, G)$, where Π is a Datalog program and G a relation name, called the *goal predicate* of Q. We say that a relation name R occurring in Π is an *intensional predicate* of Q if R appears as a relation name of at least one head of a rule of Π. Otherwise, we call R an *extensional predicate* of Q.

Example 2.16. Let $Q = (\Pi, G)$, where Π consists of the two Datalog rules

$$E(x, y) \to G(x, y),$$
$$E(x, y), G(y, z) \to G(x, z).$$

Then Q is a Datalog query with goal predicate G. The CQ corresponding to the first rule above is simply $E(x, y)$, while the CQ corresponding to the second rule is $\exists y\, (E(x, y) \wedge G(y, z))$. Notice that Q has only one extensional predicate (namely E), and only one intensional predicate (namely G). ⊣

[3] Notice thus that β is not permitted to be an equality atom, while the α_i may also be equality atoms.

Fixed point semantics We are now going to define the semantics of Datalog queries. Let $Q = (\Pi, G/k)$ be a Datalog query, and consider an $\mathbf{S}$-database $\mathfrak{D}$, where $\mathbf{S}$ consists of the set of all extensional predicates of Q. Let $\mathbf{T} \supseteq \mathbf{S}$ be the set of all predicates occurring in Π.

Let T_Q be the *one-step operator for* Q defined by

$$T_Q \colon \mathfrak{F} \longmapsto \mathfrak{F} \cup \{R(\bar{a}) \mid \text{there is a rule } \tau \in \Pi \text{ with head } R(\bar{x}) \text{ and a body corresponding to } \exists \bar{y}\, \varphi(\bar{x}, \bar{y}) \text{ such that } \mathfrak{F} \models \exists \bar{y}\, \varphi(\bar{a}, \bar{y}), \text{ where } \bar{a} \in \mathsf{adom}(\mathfrak{F})^{|\bar{a}|}\}.$$

Notice that T_Q is defined for all $\mathbf{T}$-databases and that T_Q is *monotone* in the sense that $\mathfrak{F}_1 \subseteq \mathfrak{F}_2$ implies that $T_Q(\mathfrak{F}_1) \subseteq T_Q(\mathfrak{F}_2)$. Given $\mathfrak{D}$, we set

$$\begin{aligned} T_Q^0(\mathfrak{D}) &\coloneqq \mathfrak{D}, \\ T_Q^{n+1}(\mathfrak{D}) &\coloneqq T_Q(T_Q^n(\mathfrak{D})), \quad \text{for } n \geq 0. \end{aligned}$$

Since T_Q is monotone and $\mathfrak{D}$ is finite, there is an $\ell \geq 0$ such that $T_Q^\ell(\mathfrak{D}) = T_Q^m(\mathfrak{D})$ for all $m \geq \ell$. Thus, $T_Q^\ell(\mathfrak{D})$ is a fixed point of T_Q and, moreover, it is actually the least fixed point of T_Q that contains $\mathfrak{D}$ (see, e.g., [1, 102]). We define that

$$Q \colon \mathfrak{D} \longmapsto \{\bar{a} \mid T_Q^\ell(\mathfrak{D}) \models G(\bar{a})\}.$$

Example 2.17. Reconsider the Datalog query $Q = (\Pi, T)$ from Example 2.16. Let

$$\mathfrak{D} \coloneqq \{E(a,b), E(b,c), E(c,d)\}.$$

Then,

$$\begin{aligned} T_Q^1(\mathfrak{D}) &= \mathfrak{D} \cup \{G(a,b), G(b,c), G(c,d)\}, \\ T_Q^2(\mathfrak{D}) &= T_Q^2(\mathfrak{D}) \cup \{G(a,c), G(b,d)\}, \\ T_Q^3(\mathfrak{D}) &= T_Q^2(\mathfrak{D}) \cup \{G(a,d)\}. \end{aligned}$$

Moreover, $T_Q^m(\mathfrak{D}) = T_Q^3(\mathfrak{D})$ for all $m \geq 3$. Hence,

$$Q(\mathfrak{D}) = \{(a,b), (b,c), (c,d), (a,c), (b,d), (a,d)\},$$

and so Q computes the transitive closure of the input relation E. ⊣

Logical semantics The semantics for Datalog queries we presented is operational in nature. Alternatively, we can define the semantics of a Datalog query $Q = (\Pi, G/k)$ in a purely logical fashion. To this end, we view the rules of Π as universally closed sentences: the rule

$$\tau \colon \quad \alpha_1, \ldots, \alpha_n \to \beta,$$

becomes the sentence

$$\forall x_1, \ldots, x_n\, (\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta),$$

where $x_1, \ldots, x_n$ is an enumeration of all the variables that appear in the body of τ. Let φ be the conjunction of all such sentences that emerge from a rule of Π. Consider an $\mathbf{S}$-database $\mathfrak{D}$, where $\mathbf{S}$ is the schema that comprises exactly the extensional predicates of Q. Now we set

$$Q\colon \mathfrak{D} \longmapsto \{(a_1, \ldots, a_k) \mid (\mathfrak{D}, \{\varphi\}) \models G(a_1, \ldots, a_k), \text{ where } a_1, \ldots, a_k \in \mathsf{const}\}.$$

It turns out that this logical semantics of Datalog queries is equivalent to the operational one we presented above (see [1, 50, 51]).

We close this section by recalling the complexity of query answering for Datalog queries (see [66, 130]):

Theorem 2.18. *The query answering problem for Datalog queries is* ExpTime*-complete in combined complexity, and* PTime*-complete in data complexity.*

2.4 Automata Techniques

The goal of this section is to introduce the classical automata models that we are going to use throughout this thesis. By "classical" we mean automata models whose principal objects of interests are words and trees which they discriminate via the notions of acceptance or rejection. In Chapter 5 we are going to encounter another type of automata, so-called *cost automata* [31, 58, 60], that do not fall into this category. The principle objects of interest for cost automata are *cost functions* which extend words and trees by a quantitative measure. We choose to delay the discussion of this type of automata until Chapter 5 when we employ them in the context of first-order rewritability (however, the material presented here will recur in different themes in Chapter 5).

Concerning classical automata, the family of automata of central interest to us is that of *tree automata* [61, 70] which comprises automata operating on trees in which each node has a label over a given finite, unranked alphabet. The family of tree automata consists of a plethora of concrete automata models which differ according to several dimensions. Among others, tree automata models on unranked trees differ in questions as to whether (i) they move in any particular direction in the tree or are constrained to move in one direction, (ii) they work on finite or infinite trees, (iii) they work on trees of arbitrary branching degree or not, (iv) they employ concepts of non-determinism or alternation [52]. Our focus in this thesis will arguably be on *alternating* automata [62, 128, 131, 133, 134] that work on *finite* trees (with an *arbitrary* or *bounded* branching degree) and that allow for *two-way* movements while processing the input tree (i.e., the automaton is allowed to move up and down in the tree). This type of automata model will be used quite extensively throughout this thesis, and therefore the main focus of this section is to get the reader acquainted with this model and its properties. Apart from two-way alternating automata, we will also introduce the most basic model of *non-deterministic automata on finite trees* [61, 62, 70, 131]. One can show that two-way alternating automata on finite trees can be translated to this simpler model by an exponential translation. This translation is achieved by mildly modifying a similar one given by Vardi in [133] for

tree automata on infinite trees. We will use this translation later to facilitate results on non-deterministic automata on finite trees.

We stress that we focus exclusively on tree automata work on finite trees. However, although none of our automata requires us to facilitate infinite trees at all, we favor the use of two-way alternating tree automata that employ the parity condition as their acceptance condition. The expert reader may recognize that automata using parity acceptance conditions are mostly used when working with infinite words or trees, which seems inconsistent given the fact that we work with finite trees only. Notice though, if an automaton uses two-way movements, it is possible that the automaton has infinite runs (see below) rather than only finite ones, and the parity condition is a simple and elegant way to discriminate accepting infinite runs from rejecting ones. Although two-way alternating automata have been used in the literature without employing the parity condition (see, e.g., [49, 62, 131]), the use of the parity condition in our automata seems more natural for our use cases.

2.4.1 Words and Trees

Given a set S, we write S^* for the set of all finite sequences that can be formed using elements from S, the (finite) *words over* S, and in this context we write ε to denote the empty word. We write $v \cdot w$ for the concatenation of two words.

An *alphabet* is a finite set of symbols Γ. A Γ-*labeled tree* is a partial function $t\colon (\mathbb{N} \setminus \{0\})^* \to \Gamma$ such that the following conditions are satisfied:

(i) The domain of t, denoted $\mathsf{dom}(t)$, is non-empty and is prefix-closed, i.e., if $x \cdot k \in \mathsf{dom}(t)$, where $x \in (\mathbb{N} \setminus \{0\})^*$ and $k \in \mathbb{N} \setminus \{0\}$, then $x \in \mathsf{dom}(t)$.

(ii) If $x \cdot k \in \mathsf{dom}(t)$, where $x \in (\mathbb{N} \setminus \{0\})^*$ and $k > 1$, then $x \cdot (k-1) \in \mathsf{dom}(t)$.

We write $\mathcal{T}(\Gamma)$ for the set of all Γ-labeled trees.

An element $x \in \mathsf{dom}(t)$ is called a *node* of t and $t(x)$ is the *label* of x. Since the domain of t is prefix-closed and non-empty, the empty sequence ε is a node of t, and we call it the *root node* of t. We say that t is *finite* iff $\mathsf{dom}(t)$ is finite, otherwise t is *infinite*. A node of the form $x \cdot k$, where $k \in \mathbb{N} \setminus \{0\}$, is a *child* of x, and x is the *parent* of $x \cdot k$. Obviously, every node distinct from the root has a unique parent. The number of children of a node is its *branching degree*, and the branching degree of t is the maximum branching degree of all nodes – notice that the branching degree may not exist if t is infinite. We say that t is *m-ary* if its branching degree is at most m.[4] For convenience, we extend the concatenation operator to also capture the number 0 by setting $x \cdot 0 \coloneqq x$ for all $x \in (\mathbb{N} \setminus \{0\})^*$. Moreover, we let $x \cdot k \cdot -1 \coloneqq x$ for all $x \in (\mathbb{N} \setminus \{0\})^*$ and $k \geq 1$. Thus, $x \cdot -1$ denotes the parent node of x if it exists at all – notice that $\varepsilon \cdot -1$ is thus not defined. The notions of *leaf node* of t, *path* in t, and *branch* in t are defined as in the case of ordinary directed trees.

[4] Notice that in some references, m-ary trees denote trees whose every node has branching degree m. Since we are going to work on finite trees only, we rather adapt a different terminology here.

Projections Suppose t is a Γ-labeled tree and s a Λ-labeled tree for some alphabets Γ and Λ. We write (t, s) for the $(\Gamma \times \Lambda)$-labeled tree defined by $(t, s)\colon v \mapsto (t(v), s(v))$. We call t the Γ-*projection* of (t, s) and s its Λ-*projection*. Given a $(\Gamma \times \Lambda)$-labeled tree t, we write $\pi_\Gamma(t)$ (respectively, $\pi_\Lambda(t)$) for its Γ-projection (respectively, Λ-projection). Given a set of $(\Gamma \times \Lambda)$-labeled trees $\mathcal{L}$, we write $\pi_\Gamma(\mathcal{L})$ (respectively, $\pi_\Lambda(\mathcal{L})$)for the set of all Γ-projections (respectively, Λ-projections) of elements contained in $\mathcal{L}$.

2.4.2 Non-deterministic Finite (Top-Down) Automata on Finite Trees

Before diving into the realm of tree automata, let us give a brief recap on the operation of standard non-deterministic automata on (finite) words. We can view such an automaton $\mathcal{A}$ as a finite device whose mode of operation is dictated by a finite set of states. Suppose $\mathcal{A}$ is presented a finite input word $w = a_1 a_2 \cdots a_n$. A snapshot of the computation of $\mathcal{A}$ can be captured by a pair (i, s), where $i \in [n]$ and s is a state of $\mathcal{A}$. We call such a pair a *position*. Intuitively, (i, s) describes the situation that $\mathcal{A}$ is reading the i-th symbol of w while being in state s. Now the transition function δ of $\mathcal{A}$ specifies the possible next moves $\mathcal{A}$ has. Given s and a_i, $\delta(s, a_i)$ specifies the set of possible states $\mathcal{A}$ can transition to. $\mathcal{A}$ selects one of these states $s' \in \delta(s, a_i)$ and launches a copy of itself in position (a_{i+1}, s') provided $i < n$ (if $\delta(s, a_i) = \varnothing$, it *rejects*). We say that (a copy of) $\mathcal{A}$ *accepts at* (i, s) if either $i = n$ and s is a final state, or if the copy that $\mathcal{A}$ launches accepts at its position. In its operation, $\mathcal{A}$ thus launches a sequence of copies of itself that operate at according positions. The word w is *accepted by* $\mathcal{A}$ iff (i) either $w = \varepsilon$ and the initial state s_0 of $\mathcal{A}$ is final, or (ii) $\mathcal{A}$ accepts at $(1, s_0)$.

Let us now consider trees instead of words. Each tree may consist of multiple branches, and we can view each branch as a plain word. Thus, an automaton working on trees has to launch, being at a node x in state s, not only one successor copy of itself, but it has to launch a successor copy for each of its children. The machinery necessary to achieve this is specified by the following definition:

Definition 2.19 ([62, 131]). A *(one-way) non-deterministic (top-down) automaton on finite trees* (1NTA) is a tuple $\mathcal{A} = (S, \Gamma, s_0, \delta, F)$, where

- S is a finite set of *states*,
- Γ is the *input alphabet*,
- $s_0 \in S$ is the *initial state*,
- $\delta\colon S \times \Gamma \to 2^{S^*}$ is the *transition function*, where $\delta(s, a)$ is finite for all $s \in S$ and all $a \in \Gamma$, and
- $F \subseteq S$ is a set of *final states*.

We say that $\mathcal{A}$ *runs on* Γ-labeled trees. We call $\mathcal{A}$ a *deterministic (top-down) automaton on finite trees* (1DTA) if $|\delta(s, a)| = 1$ for all $s \in S$ and all $a \in \Gamma$.

Given a finite Γ-labeled input tree t and a 1NTA $\mathcal{A}$, a *run* of $\mathcal{A}$ over t is a labeling function $\rho\colon \mathsf{dom}(t) \to S$ such that (i) the root of t is labeled with s_0, (i.e., $\rho(\varepsilon) = s_0$), and (ii) if

v is not a leaf node and has $k \geq 0$ children, then $\rho(v{\cdot}1){\cdot}\rho(v{\cdot}2)\cdots\rho(v{\cdot}k) \in \delta(\rho(v), t(v))$. If for every leaf node v of t, there is a word $s_1 s_2 \cdots s_k \in \delta(\rho(v), t(x))$ with $\{s_1, s_2, \ldots, s_k\} \subseteq F$, then we call ρ *accepting*. We say that $\mathcal{A}$ *accepts* t if there is an accepting run of $\mathcal{A}$ over t. The *language of* $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of all Γ-labeled trees t it accepts.

Let us now describe the operation of a 1NTA $\mathcal{A}$ on a given input tree t as we did before for standard non-deterministic automata on words. We call a pair (v, s) with $v \in \mathsf{dom}(t)$ and $s \in S$ again a *position*. Suppose $\mathcal{A}$ is at position (v, s) and suppose the children of v are $v_1, \ldots, v_k$. The 1NTA $\mathcal{A}$ reads the symbol $t(v)$ and chooses a word $s_1 s_2 \cdots s_k \in \delta(s_0, t(\varepsilon))$ of length k over S – if it finds no such $s_1 s_2 \cdots s_k$, it *rejects*. For $i = 1, \ldots, k$, it launches a copy of itself at position (v_i, s_i), i.e., the i-th copy of $\mathcal{A}$ proceeds to read the label of v_i in state s_i. We say that (a copy of) $\mathcal{A}$ *accepts at* (v, s) if either v is a leaf node and $s \in F$, or all the k copies launched by $\mathcal{A}$ accept at their positions. $\mathcal{A}$ *accepts* t iff it accepts at (ε, s_0).

We added the term "one-way" in the definition of 1NTA in order to emphasize that 1NTAs can only move in one direction within the tree, namely downward.

Properties of 1NTAs Let us now provide some results which we are going to use at later points in this thesis. The first one is the folklore result stating that 1NTAs are closed under intersections [64]:

Proposition 2.20. *Given 1NTAs $\mathcal{A}$ and $\mathcal{B}$ running over Γ-labeled trees, there is a 1NTA $\mathcal{C}$ whose size is the product of the sizes of $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$. Moreover, $\mathcal{C}$ can be constructed in polynomial time.*

1NTAs are also closed under complements, but this incurs an exponential blowup in the size of its state set:

Proposition 2.21. *Given a 1NTA $\mathcal{A}$ running on Γ-labeled trees, there is a 1NTA $\overline{\mathcal{A}}$, whose state set is of exponential size in the size of the state set of $\mathcal{A}$, such that $\mathcal{L}(\overline{\mathcal{A}}) = \mathcal{T}(\Gamma) \setminus \mathcal{L}(\mathcal{A})$.*

Proposition 2.21 can be shown by using the fact that there is an exponential translation from alternating tree automata to 1NTAs [62, 128] and the fact that complementing alternating tree automata is feasible in polynomial time (see below).

Besides the standard Boolean operations, 1NTAs are also closed under projections:

Proposition 2.22. *Let $\mathcal{A}$ be a 1NTA running on $(\Gamma \times \Lambda)$-labeled trees. Then there exists a 1NTA $\mathcal{A}_\Gamma$ running on Γ-labeled trees such that $\mathcal{L}(\mathcal{A}_\Gamma) = \pi_\Gamma(\mathcal{L}(\mathcal{A}))$. The state set of $\mathcal{A}_\Gamma$ is just the state set of $\mathcal{A}$, and $\mathcal{A}_\Gamma$ can be constructed from $\mathcal{A}$ in polynomial time.*

Proof. Let $\mathcal{A} = (S, \Gamma \times \Lambda, s_0, \delta, F)$. We let $\mathcal{A}_\Gamma \coloneqq (S, \Gamma, s_0, \delta_\Gamma, F)$, where

$$\delta_\Gamma \colon (s, a) \longmapsto \bigcup_{b \in \Lambda} \delta(s, (a, b)), \quad \text{for } s \in S \text{ and } a \in \Gamma.$$

It is not hard to show that $\mathcal{L}(\mathcal{A}_\Gamma) = \pi_\Gamma(\mathcal{L}(\mathcal{A}))$. □

The *(language) emptiness problem* for 1NTA is, given a 1NTA $\mathcal{A}$, the task to decide whether $\mathcal{L}(\mathcal{A}) = \varnothing$. It turns out that this problem can be solved in polynomial time [130]:

THEOREM 2.23. *The emptiness problem for 1NTAs is in* PTIME.

2.4.3 (Two-Way) Alternating Parity Tree Automata

The way a 1NTA operates is constrained in the sense that it only permits limited movements along the tree from its top to its bottom. Moreover, a 1NTA provides no ways to implement alternation [52, 128] in its computational process. We are going to introduce a popular automata model on trees that overcomes these limitations by providing two-way movements, i.e., the automaton can also move up in the tree, and alternation mechanisms, i.e., apart from existential branching, the automaton is also able to perform universal branching. The automata model we are going to introduce will be the model which we mostly use throughout this thesis. Therefore, we aim here at a fully self-contained description of it. Material in this section is mainly taken from [31, 89, 133, 134]. Before introducing the model, we need some additional auxiliary notions.

A *positive Boolean formula* is a propositional formula that is built up solely using conjunctions, disjunctions, propositional variables, and the truth constants true and false. Given a set X, we write $\mathbb{B}^+(X)$ to denote the set of all positive Boolean formulas that can be constructed using propositional variables from X.

A *direction* is a function that maps nodes of a labeled tree to sets of nodes of that tree. Intuitively, we use directions to specify the possible movements the automaton is able to perform within a tree. We shall denote directions in the usual way by introducing abbreviations for them. More precisely, we define the following functions:

$$k\colon v \longmapsto \begin{cases} \{v \cdot k\}, & \text{if } v \cdot k \text{ is defined,} \\ \varnothing, & \text{otherwise,} \end{cases} \qquad \text{for } k \in \{-1, 0, 1, 2, \ldots\}.$$

Notice that $v \cdot k$ is not defined only if $v = \varepsilon$ and $k = -1$, and hence $-1(\varepsilon) = \varnothing$. In all other cases, $k(v)$ consists of exactly one node, namely the k-th successor of v in case $k \geq 1$, the node v itself in case $k = 0$, and the parent of v in case $k = -1$. We use $\updownarrow$ to denote the direction that yields all possible neighboring nodes of a given node, including the parent. Formally, we thus define:

$$\updownarrow\colon v \longmapsto \bigcup_{k \in \{-1,0,1,\ldots\}} k(v).$$

Given a Γ-labeled tree t, a node $v \in \mathsf{dom}(t)$, and a direction d, by the *neighbors of v in direction d*, denoted $d_t(v)$, we mean the set of nodes $d(v) \cap \mathsf{dom}(t)$.

We are now ready to introduce the automata model of interest:

DEFINITION 2.24. An *alternating parity tree automaton* is a tuple

$$\mathcal{A} = (S, \Gamma, \mathsf{Dir}, \delta, \Omega, s_0),$$

where

- S is a finite set of *states*;
- Γ is an alphabet, called the *input alphabet*;
- Dir is a set of *directions*;
- $\delta\colon S \times \Gamma \to \mathbb{B}^+(\mathsf{tran}(\mathcal{A}))$ is the *transition function*, where

$$\mathsf{tran}(\mathcal{A}) \coloneqq \{\langle d\rangle s, [d]s \mid s \in S, d \in \mathsf{Dir}\};$$

- $s_0 \in S$ is the *initial state*;
- $\Omega\colon S \to \mathbb{N}$ is the *priority function* which assigns to each state a *priority*.

We say that $\mathcal{A}$ *runs on* Γ-labeled trees. If $\mathsf{Dir} = \{0, \updownarrow\}$ or $\mathsf{Dir} = \{-1, 0, 1, \ldots, m\}$ for some $m \geq 1$, then we say that $\mathcal{A}$ is *two-way* (for short, a 2APTA). In the former case (i.e., $\mathsf{Dir} = \{0, \updownarrow\}$), we say that $\mathcal{A}$ *runs on amorphous* Γ*-labeled trees* (for short, a $\updownarrow$-2APTA), and in the latter case (i.e., $\mathsf{Dir} = \{-1, 0, 1, \ldots, m\}$), we say that $\mathcal{A}$ *runs on* m*-ary* Γ*-labeled trees* (for short, an m-2APTA).

If neither $\updownarrow \in \mathsf{Dir}$ nor $-1 \in \mathsf{Dir}$ then we say that $\mathcal{A}$ is *one-way* (for short, an APTA).

If $\updownarrow \in \mathsf{Dir}$, then by the *input trees* of $\mathcal{A}$ we mean the set of all Γ-labeled trees. Otherwise, we mean by the *input trees* of $\mathcal{A}$ the set of all m-ary Γ-labeled trees where $m \coloneqq \max\{n \mid n \in \mathbb{N} \cap \mathsf{Dir}\}$.

Remark 2.25. We remind the reader here of a remark we have already made in the introductory part of this section. Unlike in the case of 1NTA, we do not require that the input trees to 2APTA are finite. Indeed, the semantics defined below for 2APTA based on parity games makes equally sense for the case of infinite trees. Employing the parity condition in automata is, however, mostly done when one considers infinite trees. As stressed above, we only deal with finite input trees and the reader might thus consider the use of the parity condition as an "overkill" in our case. However, we remind the reader that our automata model may make use of two-way movements which may give rise to infinite runs. Although two-way alternating automata were also defined for a sole use on finite trees (see, e.g., [49, 62, 131]), we choose to work with parity automata as they seem to be more convenient for our purposes.

Before providing the formal semantics of 2APTA, let us give a more intuitive explanation of its operation on a Γ-labeled input tree t. We can view a 2APTA $\mathcal{A}$ as a finitely specified device that traverses t according to the rules prescribed by the transition function $\delta(\cdot, \cdot)$. Hence, at each instant in time, we can imagine that $\mathcal{A}$ is in a given state $s \in S$ and at a node $v \in \mathsf{dom}(t)$ of the input tree – let us call the pair (v, s) a *position*. Once $\mathcal{A}$ reads the symbol $a \coloneqq t(v)$, $\mathcal{A}$ proceeds by inspecting $\varphi \coloneqq \delta(s, a)$. We can view φ as being part of the state of $\mathcal{A}$ since the acceptance behavior of $\mathcal{A}$ in state s at position v also depends on the formula φ – thus, let us augment the positional information of $\mathcal{A}$ by φ and say in the following that $\mathcal{A}$ is at position (v, s, φ).

If $\varphi = \mathsf{true}$ then $\mathcal{A}$ *accepts*, and if $\varphi = \mathsf{false}$ then it *rejects*. Otherwise, $\mathcal{A}$ creates copies of itself according to the shape of φ. If $\varphi = \varphi_1 \vee \varphi_2$, then $\mathcal{A}$, for some $i \in \{1, 2\}$, creates a copy of itself at position (v, s, φ_i), and $\mathcal{A}$ accepts iff this copy accepts. If $\varphi = \varphi_1 \wedge \varphi_2$,

then $\mathcal{A}$ creates two copies of itself at the respective positions (v, s, φ_1) and (v, s, φ_2), and $\mathfrak{A}$ accepts iff these two copies accept. Now if $\varphi = \langle d \rangle s'$, then $\mathcal{A}$ selects some $w \in d_t(v)$ (if such a w does not exist, it immediately rejects) and creates a copy of itself that proceeds at position $(w, s', \delta(s', t(w)))$. In this case $\mathcal{A}$ accepts at position (v, s, φ) iff that copy accepts at position $(w, s', \delta(s', t(w)))$. Likewise, if $\varphi = [d]s'$, then for every $w \in d_t(v)$, $\mathcal{A}$ creates a copy of itself at position $(w, s', \delta(s', t(w)))$. $\mathcal{A}$ accepts at position (v, s, φ) iff all these copies accept at position $(w, s', \delta(s', t(w)))$ – notice that if there is no $w \in d_t(v)$, then $\mathcal{A}$ vacuously accepts from (v, s, φ).

The input tree t is *accepted* by $\mathcal{A}$ iff $\mathcal{A}$ accepts at position $(\varepsilon, s_0, \delta(s_0, t(\varepsilon))$. Now it is possible that the process of creating copies of $\mathcal{A}$ and sending the copies to different nodes of t is never ending in the sense that these operations are carried out indefinitely, giving rise to an infinite sequence of automata copies. Such a sequence can be visualized as a branch in an exhaustive computation tree that emerges from the choices that $\mathcal{A}$ has. To specify acceptance for this case, we employ the *parity condition* – roughly speaking, we inspect the priorities of the states of the infinite sequence of invoked automata instances, and we judge such a sequence as *accepting* iff the least priority that occurs infinitely often in this sequence is even.

In the following, we are going to make this intuitive semantics more precise. The description of the operation of 2APTA above suggests that the way an alternating automaton (one-way or two-way) operates on an input tree can be viewed as an infinite game between two players [110, 134]. In fact, we are going to define the notion of acceptance via *parity games*.

Semantics of 2APTAs

Let us now provide a semantics to the 2APTA defined in Definition 2.24 by employing parity games. Given a 2APTA $\mathcal{A} = (S, \Gamma, \mathsf{Dir}, \delta, \Omega, s_0)$ as above and a Γ-labeled input tree t, the notion of acceptance of t by $\mathcal{A}$ is defined via a game played by two players, *Eve* and *Adam*. The goal of Eve is to satisfy the parity condition and prove that t is accepted by $\mathcal{A}$, while to goal of Adam is to disprove this. We shall make this more precise in the following.

Let $\chi \in \mathbb{B}^+(\mathsf{tran}(\mathcal{A}))$ be a positive formula. We assign χ to an *owner* according to its form:

- If $\chi = \mathsf{true}$ (resp., $\chi = \mathsf{false}$) then χ is owned by Adam (resp., Eve).
- If $\chi = \chi_1 \wedge \chi_2$ (resp., $\chi = \chi_1 \vee \chi_2$) then χ is owned by Adam (resp., Eve).
- If $\chi = [d]s$ (resp., $\langle d \rangle s$) then χ is owned by Adam (resp., Eve).

Moreover, we set

$$\Omega(\chi) \coloneqq \begin{cases} \Omega(s), & \text{if } \chi = \langle d \rangle s \text{ or } \chi = [d]s, \\ \max \Omega(S), & \text{otherwise.} \end{cases}$$

The *acceptance (parity) game* $\mathcal{G}(\mathcal{A}, t)$ *for* $\mathcal{A}$ *and* t is played in the *arena*

$$\mathbb{B}^+(\mathsf{tran}(\mathcal{A})) \times \mathsf{dom}(t).$$

For each position (χ, v) of the arena, we define the set of *possible choices*:

- If $\chi = \mathsf{true}$ or $\chi = \mathsf{false}$, then the possibly choices are $\varnothing$.
- If $\chi = \chi_1 \wedge \chi_2$ or $\chi = \chi_1 \vee \chi_2$, then the possible choices are $\{(\chi_1, v), (\chi_2, v)\}$.
- If $\chi = [d]s$ or $\chi = \langle d \rangle s$, then the possible choices are $\{(\delta(s, t(w)), w) \mid w \in d_t(v)\}$.

Let $\chi_0 \coloneqq \delta(s_0, t(\varepsilon))$. The initial position of the game $\mathcal{G}(\mathcal{A}, t)$ is (χ_0, ε) and, from any position (χ, v), the game proceeds as follows:

(i) The player who owns χ selects a (χ', w) among the possible choices for (χ, v), provided there is one, and if so then

(ii) the game continues from position (χ', w).

The transition from (χ, v) to (χ', w) is called a *move*. By a *play* in $\mathcal{G}(\mathcal{A}, t)$ we mean a (finite or infinite) sequence of positions $(\chi_0, \varepsilon), (\chi_1, v_1), (\chi_2, v_2), \ldots$ that arise from successive moves. We say that a play π is *winning for Eve*, if either (i) π is finite and the last position (χ, v) is such that χ is owned by Adam, or (ii) π is infinite and *satisfies* the *parity (acceptance) condition*, that is, the least priority among $\Omega(\chi_0), \Omega(\chi_1), \Omega(\chi_2), \ldots$ that occurs infinitely often is even. We say that π is *winning for Adam* if it is not winning for Eve.

A *strategy* for one of the players is a function that returns the next choice for that player given the history of the play. A strategy is *memoryless* if the strategy only depends on the current position but not on the entire history of the play. Fixing a strategy for both players thus uniquely determines a play in $\mathcal{G}(\mathcal{A}, t)$. A play π is *consistent* with a strategy ξ if there is a strategy ξ' for the other player such that ξ and ξ' yield π.

We say that a strategy for a player is *winning in* $\mathcal{G}(\mathcal{A}, t)$ if every play consistent with it is winning for that player. We remark that parity games as $\mathcal{G}(\mathcal{A}, t)$ are *determined*, i.e., either Eve or Adam has a winning strategy in $\mathcal{G}(\mathcal{A}, t)$, and they are *memoryless*, i.e., if a player has a winning strategy, then that player also has a memoryless winning strategy [73, 106, 109].

Definition 2.26. The *language of* $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of all Γ-labeled trees t such that Eve has a winning strategy in $\mathcal{G}(\mathcal{A}, t)$.

Example 2.27. Let $\mathcal{A} = (\{s_0\}, \{a\}, \{1, 2\}, \delta, \Omega, s_0)$ be an automaton with

$$\delta(s_0, a) \coloneqq [1]s_0 \wedge [2]s_0,$$
$$\Omega(s_0) \coloneqq 1.$$

Suppose t is an $\{a\}$-labeled input tree. Let us inspect the game $\mathcal{G}(\mathcal{A}, t)$. It starts at position $(\varepsilon, [1]s_0 \wedge [2]s_0)$ which is owned by Adam. Thus, Adam chooses either $(\varepsilon, [1]s_0)$ or $(\varepsilon, [2]s_0)$. Both positions are again owned by Adam, and if he chose $(\varepsilon, [i]s_0)$, then

he has, in the next move, point to some node $v_1 \in i_t(\varepsilon)$. If he succeeds, then the game continues from position $(v_1, [1]s_0 \wedge [2]s_0)$ (if he cannot point to a successor v_1, Adam loses). Now the game continues in this fashion – Adam has to select successor $v_2, v_3, \ldots$ in order to win the game. Since $\Omega(s_0) = 1$, Adam indeed has to select infinitely many nodes v_i in order to win $\mathcal{G}(\mathcal{A}, t)$. Thus, we easily see that Adam has a winning strategy in $\mathcal{G}(\mathcal{A}, t)$ iff t is infinite. Hence, Eve has a winning strategy in $\mathcal{G}(\mathcal{A}, t)$ iff t is finite and so $\mathcal{A}$ accepts t iff t is finite. ⊣

From m-2APTAs to 1NTAs It is well-known that standard two-way alternating automata (without the parity acceptance condition) that work on finite trees have an exponential translation to 1NTAs [62, 128]. Moreover, in [133], Vardi shows how to translate two-way alternating parity automata that work on infinite trees (in which every node has a fixed branching degree) to equivalent non-deterministic parity tree automata. Our goal here is to show, assuming that a given m-2APTA accepts finite trees only, that we can provide an according translation to 1NTA:

Theorem 2.28. *Let $\mathcal{A}$ be an m-2APTA that accepts finite trees only. Then there exists a 1NTA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. The number of states of $\mathcal{A}'$ is exponential in the number of states of $\mathcal{A}$, and $\mathcal{A}'$ can be constructed in exponential time.*

We provide a proof sketch of Theorem 2.28 in Appendix B.1. Its proof is essentially a slight modification of the result provided in [133] for two-way alternating parity automata working on infinite trees.

Language emptiness As in the case of 1NTA, the *(language) emptiness problem* for a class of 2APTAs is the problem, given a 2APTA $\mathcal{A}$ from that class, to decide whether $\mathcal{L}(\mathcal{A}) = \varnothing$. In [134], the following is shown:

Theorem 2.29. *The emptiness problem for $\updownarrow$-2APTAs is decidable in* ExpTime.

Furthermore, using Theorems 2.23 and 2.28, we can conclude:

Theorem 2.30. *The emptiness problem for m-2APTAs is decidable in* ExpTime.

Closure Properties of 2APTAs It is not hard to check that 2APTAs are closed under Boolean operations:

Proposition 2.31. *The class of $\updownarrow$-2APTAs is closed under intersections, complements, and unions. That is, given $\updownarrow$-2APTAs $\mathcal{A}$ and $\mathcal{B}$ running on Γ-labeled trees, the following hold:*

(i) *There is a $\updownarrow$-2APTA $\mathcal{A} \cap \mathcal{B}$ such that $\mathcal{L}(\mathcal{A} \cap \mathcal{B}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$.*

(ii) *There is a $\updownarrow$-2APTA $\mathcal{A} \cup \mathcal{B}$ such that $\mathcal{L}(\mathcal{A} \cup \mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$.*

(iii) *There is a $\updownarrow$-2APTA $\overline{\mathcal{A}}$ such that $\mathcal{L}(\overline{\mathcal{A}}) = \mathcal{T}(\Gamma) \setminus \mathcal{L}(\mathcal{A})$.*

According statements hold for m-2APTA as well. Moreover, all the mentioned automata can be constructed in linear time.

Proof. Let $\mathcal{A} = (S_\mathcal{A}, \Gamma, \mathsf{Dir}, \delta_\mathcal{A}, \Omega_\mathcal{A}, s_\mathcal{A})$ and $\mathcal{B} = (S_\mathcal{B}, \Gamma, \mathsf{Dir}, \delta_\mathcal{B}, \Omega_\mathcal{B}, s_\mathcal{B})$ and assume w.l.o.g. that $S_\mathcal{A} \cap S_\mathcal{B} = \varnothing$.

We let $\mathcal{A} \cap \mathcal{B} \coloneqq (S_\mathcal{A} \cup S_\mathcal{B} \cup \{s_{\mathcal{A}\cap\mathcal{B}}\}, \Gamma, \mathsf{Dir}, \delta_{\mathcal{A}\cap\mathcal{B}}, \Omega_{\mathcal{A}\cap\mathcal{B}}, s_{\mathcal{A}\cap\mathcal{B}})$, where $s_{\mathcal{A}\cap\mathcal{B}} \notin S_\mathcal{A} \cup S_\mathcal{B}$ and

$$\begin{aligned}
\delta_{\mathcal{A}\cap\mathcal{B}}(s_{\mathcal{A}\cap\mathcal{B}}, a) &\coloneqq \langle 0\rangle s_\mathcal{A} \wedge \langle 0\rangle s_\mathcal{B}, \quad \text{for all } a \in \Gamma,\\
\delta_{\mathcal{A}\cap\mathcal{B}}(s, a) &\coloneqq \delta_\mathcal{A}(s, a), \quad \text{for all } s \in S_\mathcal{A} \text{ and } a \in \Gamma,\\
\delta_{\mathcal{A}\cap\mathcal{B}}(s, a) &\coloneqq \delta_\mathcal{B}(s, a), \quad \text{for all } s \in S_\mathcal{B} \text{ and } a \in \Gamma,
\end{aligned}$$

and

$$\begin{aligned}
\Omega_{\mathcal{A}\cap\mathcal{B}}(s_{\mathcal{A}\cap\mathcal{B}}) &\coloneqq \max(\Omega_\mathcal{A}(S_\mathcal{A}) \cup \Omega_\mathcal{B}(S_\mathcal{B})),\\
\Omega_{\mathcal{A}\cap\mathcal{B}}(s) &\coloneqq \Omega_\mathcal{A}(s), \quad \text{for all } s \in S_\mathcal{A},\\
\Omega_{\mathcal{A}\cap\mathcal{B}}(s) &\coloneqq \Omega_\mathcal{B}(s), \quad \text{for all } s \in S_\mathcal{B},
\end{aligned}$$

It is easy to check that $\mathcal{L}(\mathcal{A} \cap \mathcal{B}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$.

For the case of complementation, we set $\overline{\mathcal{A}} \coloneqq (\overline{S}_\mathcal{A}, \Gamma, \mathsf{Dir}, \overline{\delta}_\mathcal{A}, \overline{\Omega}_\mathcal{A}, \overline{s}_\mathcal{A})$, where $\overline{S}_\mathcal{A} \coloneqq S_\mathcal{A}$, $\overline{\Omega}_\mathcal{A} \colon s \mapsto \Omega_\mathcal{A}(s) + 1$, $\overline{s}_\mathcal{A} \coloneqq s_\mathcal{A}$, and, for all $s \in \overline{S}_\mathcal{A}$ and all $a \in \Gamma$, we let $\overline{\delta}_\mathcal{A}(s, a)$ be the positive Boolean formula that arises from $\delta_\mathcal{A}(s, a)$ by flipping (i) true and false, (ii) conjunctions and disjunctions, and (iii) $\langle d\rangle s$ and $[d]s$. Given an input tree t, it is not hard to show Eve has a winning strategy in $\mathcal{G}(\mathcal{A}, t)$ iff Adam has a winning strategy in $\mathcal{G}(\overline{\mathcal{A}}, t)$. Since parity games are determined, we immediately obtain that $\mathcal{L}(\overline{\mathcal{A}}) = \mathcal{T}(\Gamma) \setminus \mathcal{L}(\mathcal{A})$.

The construction of $\mathcal{A} \cup \mathcal{B}$ by using intersection and union, i.e.,

$$\mathcal{A} \cup \mathcal{B} \coloneqq \overline{(\overline{\mathcal{A}} \cap \overline{\mathcal{B}})}.$$

Notice that all constructions presented are feasible in linear time. □

Notation. When constructing intersections, complements, and unions of automata, we often use the notations $\mathcal{A} \cap \mathcal{B}$, $\overline{\mathcal{A}}$, and $\mathcal{A} \cup \mathcal{B}$ to denote the according operations, though, strictly speaking, the ways these automata are constructed are not unique – we implicitly assume that these automata are constructed as presented in the proof of Proposition 2.31. A similar convention is employed for other types of automata that we encounter in this thesis.

We can use Theorem 2.28 and Proposition 2.22 in order to immediately conclude that we can form the projection of an m-2APTA at an exponential cost:

Proposition 2.32. *Suppose $\mathcal{A}$ is an m-2APTA that runs on m-ary $(\Gamma \times \Lambda)$-labeled trees. Then there exists an m-2APTA $\mathcal{A}_\Gamma$, whose state set is of exponential size in the size of the state set of $\mathcal{A}$, such that $\mathcal{L}(\mathcal{A}_\Gamma) = \pi_\Gamma(\mathcal{L}(\mathcal{A}))$. Moreover, we can construct $\mathcal{A}_\Gamma$ in exponential time from $\mathcal{A}$.*

Proof. According to Theorem 2.28 and Proposition 2.22, we can construct a 1NTA $\mathcal{A}' = (S, \Gamma, s_0, \delta, F)$ that runs on Γ-labeled trees such that $\mathcal{L}(\mathcal{A}') = \pi_\Gamma(\mathcal{L}(\mathcal{A}))$. $\mathcal{A}'$ has

exponentially many states in the number of states of $\mathcal{A}$, and the time needed to construct it is exponential. Let $\mathcal{A}_\Gamma \coloneqq (S_\Gamma, \Gamma, \{-1, 0, 1, \dots, m\}, \delta_\Gamma, \Omega_\Gamma, s_{0,\Gamma})$ be the m-2APTA constructed as follows. We set $S_\Gamma \coloneqq S$, $s_{0,\Gamma} \coloneqq s_0$, and $\Omega_\Gamma(s) \coloneqq 1$ for all $s \in S_\Gamma$. Moreover, for $s \in S_\Gamma$ and $a \in \Gamma$ we set

$$\delta_\Gamma(s, a) \coloneqq \bigvee \{\langle 1\rangle s_1 \wedge \cdots \wedge \langle k\rangle s_k \mid s_1 \cdots s_k \in \delta(s, a), s_1, \dots, s_k \in S, 0 \leq k \leq m\},$$

where the empty conjunction equals true and the empty disjunction equals false. It is not hard to check that $\mathcal{L}(\mathcal{A}_\Gamma) = \mathcal{L}(\mathcal{A}')$, whence the claim follows. □

CHAPTER 3

Ontology-Mediated Querying

The main purpose of this chapter is to introduce the basic concepts of *(rule-based) ontology-mediated querying*. The central notion of interest to us is that of an *ontology-mediated query* (OMQ) [37]. Roughly speaking, an ontology-mediated query is a composite query consisting of a standard database query plus an ontology, where the latter serves as a "mediator" between the data and the database query. Thus, an ontology provides a unified conceptual view of the data, and user queries are formulated in the schema of the ontology. From a logical point of view, an ontology is just a logical theory that is employed to derive knowledge which is not explicitly given by the data, but implicitly encoded using the logical rules of the ontology.

Hence, one choice to be taken when specifying an OMQ is the ontology which is used to formulate the OMQ. Popular choices in the literature include description logics (DLs) [12] and other fragments of first-order logic. In this work we focus on ontology-mediated queries whose ontology is formulated via *tuple-generating dependencies* (TGDs) [25]. Tuple-generating dependencies (also known as *existential rules* [14] or *existential Datalog$^{\pm}$ rules* [46]) are, roughly speaking, Datalog rules that do not only allow the inference of new relations among existing objects, but that also allow the "invention" of new objects. Syntactically speaking, TGDs are Datalog rules that allow the use of existential quantifiers in their rule heads. Thus, just as Datalog can be conceived to be "conjunctive queries plus the use of recursion," TGDs can be conceived as "Datalog rules plus value invention" [43]. The use of TGDs in formulating ontologies is a rather recent application of them – TGDs had become prominent for other applications, e.g., data exchange [76] and operations on schema mappings [75]. This chapter is dedicated to introduce the basic terminology concerning TGDs – in particular, we are going to introduce the *chase procedure* which serves as a central algorithmic tool throughout this thesis.

It is well-known that answering conjunctive queries under arbitrary sets of TGDs is undecidable [24]. This negative result has led to a flurry of research activity on the identification of classes of sets of TGDs for which query answering is decidable. In this chapter, we are going to revisit the main principles that lie behind fragments which

permit decidable query answering, as well as concrete classes of TGDs that implement these principles and thus have a decidable query answering problem.

Outline This chapter is organized as follows. In Section 3.1 we are going to introduce tuple-generating dependencies and their terminology. We proceed to introduce the notion of ontology-mediated query in Section 3.2, and we define the chase procedure in Section 3.3. Finally, we are going to treat decidability paradigms for classes of TGDs as well as concrete decidable classes of TGDs in Section 3.4.

3.1 Tuple-Generating Dependencies

A *tuple-generating dependency* [25] (TGD, also known as *existential rule* [14] or *Datalog$^{\pm}$ rule* [46]) is a first-order sentence τ of the form

$$\forall \bar{x}\,(q(\bar{x}) \to p(\bar{x})),$$

where $q(\bar{x})$ and $p(\bar{x})$ are conjunctive queries such that $p(\bar{x})$ uses no equality atoms. When writing down TGDs, we usually omit the preceding universal quantifiers, and, in order to avoid confusion, we shall assume w.l.o.g. that the existentially quantified variables of $q(\bar{x})$ are distinct from the existentially quantified variables in $p(\bar{x})$. Thus, τ is mostly presented in the form

$$\varphi(\bar{x}, \bar{z}) \to \exists \bar{y}\, \psi(\bar{x}, \bar{y}),$$

where $\varphi(\bar{x}, \bar{z})$ and $\psi(\bar{x}, \bar{y})$ are conjunctions of atoms with free variables as indicated, $[\bar{y}] \cap [\bar{z}] = \varnothing$, $q(\bar{x}) = \exists \bar{z}\, \varphi(\bar{x}, \bar{z})$, and $p(\bar{x}) = \exists \bar{y}\, \psi(\bar{x}, \bar{y})$. Moreover, when presenting $\varphi(\bar{x}, \bar{z})$, we often separate its atoms by comma instead of conjunction (in reminiscence of Datalog rules). The set of atoms contained in $\varphi(\bar{x}, \bar{z})$ are called the *body (atoms)* of τ, while the set of atoms contained in $\psi(\bar{x}, \bar{y})$ are the *head (atoms)* of τ, and we let $\mathsf{body}(\tau) \coloneqq \mathsf{body}(q(\bar{x}))$ and $\mathsf{head}(\tau) \coloneqq \mathsf{body}(p(\bar{x}))$. When speaking about the "body" of τ (respectively, the "head" of τ), we shall often mean the CQ $q(\bar{x})$ (respectively, $p(\bar{x})$).

We write $\mathsf{var}_{\exists}(\tau)$ for the variables $\bar{y}$, i.e., those variables that are existentially quantified in the head of τ. The variables $\bar{x}$ are called the *frontier variables* of τ. Notice that due to our convention that existentially quantified variables in the head are different from the variables in the body, a variable is a frontier variable of τ iff it has an occurrence in the body and in the head – variables with this property as said to be *propagated* to the head. We say that τ is *constant-free*, if it does not have any occurrence of a constant. Accordingly, we say that a set of TGDs $\mathcal{O}$ is *constant-free* if each TGD contained in it is constant-free. We write $\mathsf{sig}(\tau)$ for the set of relation names occurring in τ, and $\mathsf{sig}(\mathcal{O})$ for $\bigcup_{\tau \in \mathcal{O}} \mathsf{sig}(\tau)$.

Let $\mathfrak{A}$ be a structure. By the standard semantics of first-order logic, $\mathfrak{A}$ is a model of τ if the following condition holds: for every homomorphism h from $\varphi(\bar{x}, \bar{z})$ to $\mathfrak{A}$, there exists a homomorphism h' such that (i) h' is a homomorphism from $\psi(\bar{x}, \bar{y})$ to $\mathfrak{A}$, and (ii) h' and h agree on the frontier variables $\bar{x}$. As TGDs are just logical sentences, we employ the standard logical notations to denote modelhood for (sets of) TGDs. That is,

given a structure $\mathfrak{A}$, we write $\mathfrak{A} \models \tau$ in case $\mathfrak{A}$ is a model of the TGD τ, and for a set of TGDs $\mathcal{O}$ we write $\mathfrak{A} \models \mathcal{O}$ if $\mathfrak{A}$ is a model of every $\tau \in \mathcal{O}$.

Notation. We write TGD for the class of all finite sets of TGDs.

3.2 (Rule-Based) Ontology-Mediated Queries

As said in the introductory part of this section, *ontology-mediated queries* (OMQs) are compound queries that consist of a standard database query "on top" of an ontology. The following notion (which follows the definition of OMQ given in [37]) captures this intuition and is central to this thesis:

Definition 3.1. An *ontology-mediated query* (OMQ) is a triple $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$, where

(i) $\mathbf{S}$ is a relational schema, called the *data schema* of Q,

(ii) $\mathcal{O}$ is an *ontology*, i.e., a finite set of existential rules, and

(iii) $q(\bar{x})$ is a UCQ over $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ and is called the *query of* Q.

When $\mathbf{S}$ comprises the entire schema of $\mathcal{O}$, we shall denote Q simply by $(\mathcal{O}, q(\bar{x}))$.

Whenever we consider a class $\mathcal{C}$ of finite sets of TGDs, then an *OMQ based on* $\mathcal{C}$ is an OMQ whose ontology belongs to $\mathcal{C}$.

Let us emphasize here that we explicitly name the schema $\mathbf{S}$ in OMQs to indicate that the query the OMQ gives rise to is an $\mathbf{S}$-query (cf. Definition 2.9) and is thus evaluated over $\mathbf{S}$-databases. Notice though that $q(\bar{x})$ can be formulated over the schema $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$, i.e., $q(\bar{x})$ is used to query the data in conjunction with the ontology.

We remark again that, from a general point of view, there is no need to restrict ontologies to be classes of finite sets of TGDs. Rather, ontologies could, in principle, be any logical theory that permits a well-defined semantics based on relational structures (for example, an ontology could be an arbitrary set of first-order sentences). However, in this thesis we exclusively focus on ontology languages that are based on existential rules, and we call such OMQ-languages *rule-based*. Let us emphasize in particular that we restrict ourselves to the case where the heads of rules consist of a CQ and contain no disjunction (see [41, 42, 108] for work on disjunctive rules). The problems studied in this thesis for classes of OMQs with disjunctions in rule heads are still rather unexplored and subject to future research.

Definition 3.2. Given an $\mathbf{S}$-database $\mathfrak{D}$, we define the semantics of Q as follows:

$$Q\colon \mathfrak{D} \longmapsto \{\bar{a} \mid (\mathfrak{D}, \mathcal{O}) \models q(\bar{a}), \text{ where } \bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}\}.$$

Tuples contained in $Q(\bar{a})$ are also called the *certain answers of* Q *to* $\mathfrak{D}$, and sometimes we call them the certain answers of $q(\bar{x})$ to $(\mathfrak{D}, \mathcal{O})$ and denote them by $\mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$.

Remark 3.3. Although the semantics of Q is defined to be a mapping from $\mathbf{S}$-databases to tuples of constants, we shall often abuse notation and extend Q to capture also structures,

i.e., given an $\mathbf{S}$-structure $\mathfrak{A}$, we write $Q(\mathfrak{A})$ for the set of all tuples $\bar{a}$ over $\mathsf{adom}(\mathfrak{A})$ such that $(\mathfrak{A}, \mathcal{O}) \models q(\bar{a})$. Accordingly, we write $\mathfrak{A} \models Q(\bar{a})$ in case $\bar{a} \in Q(\mathfrak{A})$.

Example 3.4. Let $Q = (\mathbf{S}, \mathcal{O}, q(x))$, where $\mathbf{S} = \{P/1, F/2\}$,

$$q(x) = \exists y, z\,(P(x) \land F(y, x) \land F(z, y) \land P(z)),$$

and $\mathcal{O}$ consists of the following rule:

$$\tau\colon \quad P(x) \to \exists y, z\,(F(y, x) \land P(y)).$$

Intuitively, τ states that every person has a father who is himself a person. The query $q(x)$ asks for all persons x that have a grandfather who is a person. Given the database $\mathfrak{D} = \{P(a), F(a, b)\}$, we can conclude that $Q(\mathfrak{D}) = \{a\}$ and so $(\mathfrak{D}, \mathcal{O}) \models q(a)$. On the other hand, notice that we cannot infer that b is himself a person, i.e., $(\mathfrak{D}, \mathcal{O}) \not\models P(b)$, and so $(\mathfrak{D}, \mathcal{O}) \not\models q(b)$. Notice also that the fact $F(a, b)$ is not necessary to derive the certain answer a, i.e., $Q(\{P(a)\}) = \{a\}$. ⊣

We call a class $\mathcal{L}$ of finite sets of TGDs an *ontology language*. Elements from $\mathcal{L}$ are called *$\mathcal{L}$-ontologies*.

Definition 3.5. An *ontology-mediated query language* is a pair $(\mathcal{L}, \mathcal{Q})$, where $\mathcal{L}$ is an ontology language and $\mathcal{Q}$ a query language.

Remark 3.6. We remark that we only consider ontology-mediated query languages of the form $(\mathcal{L}, \mathcal{Q})$, where $\mathcal{Q}$ is a class of (U)CQs.

The query answering problem for OMQs Recall that, for a query language $\mathcal{L}$, we denote by $\mathsf{Eval}(\mathcal{L})$ the query answering problem for $\mathcal{L}$. Given an ontology-mediated query language $(\mathcal{L}, \mathcal{Q})$, we shall denote by $\mathsf{Eval}(\mathcal{L}, \mathcal{Q})$ the query answering problem for $(\mathcal{L}, \mathcal{Q})$. Via the notion of certain answers, the query answering problem is thus equally defined for OMQs as it is for standard database query languages.

Sometimes we shall speak about the query answering problem for a class $\mathcal{C}$ of sets of TGDs rather than a class of OMQs. In this case, we mean – given a database $\mathfrak{D}$, a set of TGDs $\mathcal{O} \in \mathcal{C}$, a (U)CQ $q(\bar{x})$, and a tuple of constants $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$ – the problem of deciding whether $(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$ holds.

3.3 The Chase Procedure

Let $\mathcal{O}$ be a set of TGDs, $\mathfrak{A}$ an $\mathbf{S}$-structure, $q(\bar{x})$ a conjunctive query, and $\bar{a}$ a tuple over $\mathsf{dom}(\mathfrak{D})$. Recall that $(\mathfrak{A}, \mathcal{O}) \models q(\bar{a})$ iff for every interpretation $(\mathfrak{B}, \bar{a})$ it holds that, if $\mathfrak{B}$ is a model of $\mathcal{O}$ and $\mathfrak{B} \supseteq \mathfrak{A}$, then $\mathfrak{B} \models q(\bar{a})$. Our goal here is to give a construction of a model $\mathfrak{U}$ such that

$$\mathfrak{U} \models q(\bar{a}) \iff (\mathfrak{A}, \mathcal{O}) \models q(\bar{a}),$$

for all $\bar{a} \in \mathsf{adom}(\mathfrak{A})^{|\bar{x}|}$. Hence, the problem of checking whether a CQ is entailed by a structure plus a (finite) set of TGDs is reduced to the problem of checking whether that CQ is satisfied in $\mathfrak{U}$.

Remark 3.7. We are mostly interested in the case where $\mathfrak{A}$ is actually an $\mathbf{S}$-database instead of an arbitrary structure. However, for technical reasons, we also use the notions developed here for structures. Therefore, we keep the definitions more general.

Definition 3.8. A structure $\mathfrak{U}$ is a *universal model of* $\mathfrak{A}$ *and* $\mathcal{O}$ if (i) $\mathfrak{U} \supseteq \mathfrak{A}$ and $\mathfrak{U}$ is a model of $\mathcal{O}$, and (ii) if $\mathfrak{B} \supseteq \mathfrak{A}$ and $\mathfrak{B}$ is a model of $\mathcal{O}$, then there exists a homomorphism from $\mathfrak{U}$ to $\mathfrak{B}$.

The following lemma states that universal models are the right structures we are looking for in order to reduce the question whether a (U)CQ is entailed by $(\mathfrak{A}, \mathcal{O})$ to the question whether the universal model satisfies the CQ at hand:

Lemma 3.9. *If* $\mathfrak{U}$ *is a universal model of* $\mathfrak{A}$ *and* $\mathcal{O}$*, then for every UCQ* $q(\bar{x})$ *over* $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ *and all tuples* $\bar{a}$ *over* $\mathsf{adom}(\mathfrak{A})$ *it holds that*

$$\mathfrak{U} \models q(\bar{a}) \iff (\mathfrak{A}, \mathcal{O}) \models q(\bar{a}).$$

Proof. Let $q(\bar{x}) = \bigvee_{i=1}^{n} q_i(\bar{x})$ and suppose first that $\mathfrak{U} \models q(\bar{a})$. Hence, there is a homomorphism h from some $q_i(\bar{x})$ to $\mathfrak{U}$ such that $h(\bar{x}) = \bar{a}$. Now let $(\mathfrak{B}, \pi)$ be an interpretation such that $\mathfrak{B} \supseteq \mathfrak{A}$, $\pi(\bar{x}) = \bar{a}$, and $\mathfrak{B} \models \mathcal{O}$. Thus, the facts of $\mathfrak{B}$ extend those of $\mathfrak{A}$, and $\mathfrak{B}$ is a model of $\mathcal{O}$, whence by the universality of $\mathfrak{U}$ it follows that there is a homomorphism h' from $\mathfrak{U}$ to $\mathfrak{B}$. Thus, $h' \circ h$ is a homomorphism from $q_i(\bar{x})$ to $\mathfrak{B}$ such that $(h' \circ h)(\bar{x}) = \bar{a}$, and so $\mathfrak{B}, \pi \models q(\bar{x})$ as required.

Suppose now that every interpretation $(\mathfrak{B}, \pi)$ with $\pi(\bar{x}) = \bar{a}$, $\mathfrak{B} \models \mathcal{O}$, and $\mathfrak{B} \supseteq \mathfrak{A}$ satisfies $\mathfrak{B}, \pi \models q(\bar{x})$. In particular, this holds for $\mathfrak{U}$ as well, and so $\mathfrak{U} \models q(\bar{a})$. □

We are now going to show how to canonically construct a suitable universal model according to the facts given in $\mathfrak{A}$ and the rules provided by $\mathcal{O}$. The procedure is known as the *chase procedure* and is a popular algorithmic tool in the context of database dependencies. It has its roots in deciding implications of dependencies [104] and in deciding containment of CQs under functional and inclusion dependencies [97]. When we talk about the "chase" we either mean the procedure itself or its resulting universal model. Intuitively, the chase procedure aims to successively satisfy all tuple-generating dependencies that are present in the ontology at hand. It does so by introducing new facts that have to be added to satisfy the rule heads. Thus it resembles the operational semantics given for Datalog queries, but it may need to introduce fresh domain elements in order to satisfy existential quantifiers of rule heads.

Let $\mathfrak{A}$ be an $\mathbf{S}$-structure and $\mathcal{O}$ be a finite set of existential rules. It will be convenient to assume a globally fixed, countably infinite set $\mathsf{nulls} = \{\lambda_1, \lambda_2, \ldots\}$ of *(labeled) nulls* at our disposal that is disjoint from const and vars. These labeled nulls serve as placeholders to fulfill the demands posed by existential quantifiers in rule heads.

Definition 3.10. Suppose $\mathfrak{J}$ is a structure and suppose $q_\tau(\bar{x})$ is the body of a rule of some $\tau \in \mathcal{O}$. If there is a homomorphism h from $q_\tau(\bar{x})$ to $\mathfrak{J}$, then we say that τ is *h-applicable to* $\mathfrak{J}$.

Definition 3.11. Let $\mathfrak{J}$ be a structure and suppose $\tau \in \mathcal{O}$ is h-applicable to $\mathfrak{J}$. Suppose the head of τ has the form $\exists \bar{y}\, \varphi(\bar{x}, \bar{y})$, where $\varphi(\bar{x}, \bar{y})$ is a conjunction of atoms and $\bar{x}$ are the frontier variables of τ. Let $\mathfrak{J}'$ be a structure that has the following properties:

(i) $\mathsf{dom}(\mathfrak{J}') = \mathsf{dom}(\mathfrak{J}) \cup \{\bar{\lambda}\}$, where $\bar{\lambda}$ is a sequence of labeled nulls of length $|\bar{y}|$ that do not appear in $\mathsf{dom}(\mathfrak{J})$.

(ii) $\mathfrak{J}'$ has precisely the following facts: all the facts of $\mathfrak{J}$ plus, in addition, the facts occurring in $\varphi(\bar{x}/h(\bar{x}), \bar{y}/\bar{\lambda})$.

Then we say that $\mathfrak{J}'$ *results from* $\mathfrak{J}$ *by an application of* (h, τ) and we call the atoms occurring in $\varphi(\bar{x}/h(\bar{x}), \bar{y}/\bar{\lambda})$ the *atoms derived from* $\mathfrak{J}$ *using* (h, τ).

Definition 3.12. A *chase sequence for* $\mathfrak{A}$ *and* $\mathcal{O}$ is a sequence π of structures $\mathfrak{J}_0, \mathfrak{J}_1, \ldots$ such that

(i) $\mathfrak{J}_0 = \mathfrak{A}$,

(ii) for each $\mathfrak{J}_{i+1}$, there is a homomorphism h_i and a rule $\tau_i \in \mathcal{O}$ such that $\mathfrak{J}_{i+1}$ results from $\mathfrak{J}_i$ by an application of (h_i, τ_i). The atoms derived from $\mathfrak{J}_i$ using (h_i, τ_i) are called the *atoms derived in the* $(i+1)$*-st chase step.*

(iii) for each $k \geq 0$, if there is a homomorphism h such that some $\tau \in \mathcal{O}$ is h-applicable to $\mathfrak{J}_k$, then there exists an $m > k$ such that $\mathfrak{J}_m$ results from $\mathfrak{J}_{m-1}$ by an application of (h, τ).

We denote by $\mathsf{chase}_\pi(\mathfrak{A}, \mathcal{O})$ the structure $\bigcup_{i \geq 0} \mathfrak{J}_i$, and we write $\mathsf{chase}^k_\pi(\mathfrak{A}, \mathcal{O})$ for the structure $\bigcup_{0 \leq i \leq k} \mathfrak{J}_i$.

Intuitively, item (iii) of the definition above states that the construction of the chase sequence π is "fair" in the sense that admissible rule applications occur within a finite number of steps. Notice also that our definition of chase sequences always forces that every chase sequence is infinite. We call such an infinite chase sequence $\mathfrak{J}_0, \mathfrak{J}_1, \ldots$ *finite* if there is an $m \geq 0$ such that $\mathfrak{J}_k = \mathfrak{J}_m$ for all $k \geq m$.

Remark 3.13. Here we defined the so-called *oblivious* chase procedure which successively adds facts to the constructed structure regardless of whether they are required at all (for example, a rule head may already be satisfied by the structure constructed so far, but the chase procedure nevertheless introduces new witnessing facts). Alternative ways to define the chase procedure are the *restricted* chase [24, 44], the *semi-oblivious chase* [105] (a.k.a. *Skolem chase*), and the *core chase* [69]. These chase variants differ in terms of the conditions they demand for inferring new facts. For example, the restricted chase only introduces new facts when the rule in consideration is not yet satisfied by the current instance. In this thesis, we will only consider the oblivious chase, since it is more convenient for our setting. We want to remark that the (results of the) oblivious chase and the other chase variants turn out to be equally appropriate from a query answering perspective – indeed, the resulting instances are all homomorphically equivalent [44, 69, 105]. However, the different chase variants may have different termination properties in the sense that,

given the same database and set of rules, one may terminate while the other does not [101, 114].

The following lemma states that chase sequences give rise to universal models [44]:

LEMMA 3.14. *Let π be a chase sequence for $\mathfrak{A}$ and $\mathcal{O}$. Then the structure $\mathsf{chase}_\pi(\mathfrak{A}, \mathcal{O})$ is a universal model of $\mathfrak{A}$ and $\mathcal{O}$.*

Moreover, two different chase sequences for $\mathfrak{A}$ and $\mathcal{O}$ give rise to homomorphically equivalent structures:

LEMMA 3.15. *Let π and π' be chase sequences for $\mathfrak{A}$ and $\mathcal{O}$. Then the structures $\mathsf{chase}_\pi(\mathfrak{A}, \mathcal{O})$ and $\mathsf{chase}_{\pi'}(\mathfrak{A}, \mathcal{O})$ are homomorphically equivalent.*

PROOF. Immediate since, by Lemma 3.14, both $\mathsf{chase}_\pi(\mathfrak{A}, \mathcal{O})$ and $\mathsf{chase}_{\pi'}(\mathfrak{A}, \mathcal{O})$ are universal models of $\mathfrak{A}$ and $\mathcal{O}$. □

Notation. By Lemma 3.15, for the purpose of evaluating CQs against chase sequences, we canonically fix a chase sequence $\pi_{\mathfrak{A},\mathcal{O}}$ for each pair $(\mathfrak{A}, \mathcal{O})$. We write $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ for $\mathsf{chase}_{\pi_{\mathfrak{A},\mathcal{O}}}(\mathfrak{A}, \mathcal{O})$ in the following and, accordingly, we write $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ instead of $\mathsf{chase}^k_{\pi_{\mathfrak{A},\mathcal{O}}}(\mathfrak{A}, \mathcal{O})$. The sequence $\pi_{\mathfrak{A},\mathcal{O}}$ is called *the* chase sequence for $\mathfrak{A}$ and $\mathcal{O}$, and we call $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ the *result of chasing* $\mathfrak{A}$ *by* $\mathcal{O}$.

When explicitly writing down $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ as a set of facts F, we shall employ a rather liberal notational convention: an expression of the form $\mathsf{chase}(\mathfrak{A}, \mathcal{O}) = F$ expresses the fact that $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ is isomorphic to the structure corresponding to F. This allows for a more liberal naming of labeled nulls, and we thus do not need to impose any order that specifies which particular labeled null is used in a particular chase step.

From Lemmas 3.9, 3.14 and 3.15 we immediately obtain:

COROLLARY 3.16. *Let $\mathfrak{A}$ be a database, $\mathcal{O}$ an ontology, and $q(\bar{x})$ a UCQ. For all $\bar{a} \in \mathsf{adom}(\mathfrak{A})^{|\bar{x}|}$, it holds that $(\mathfrak{A}, \mathcal{O}) \models q(\bar{a})$ iff $\mathsf{chase}(\mathfrak{A}, \mathcal{O}) \models q(\bar{a})$.*

Example 3.17. Let $Q = (\mathbf{S}, \mathcal{O}, q(x))$ be the OMQ given in Example 3.4. Suppose furthermore that $\mathfrak{D} = \{P(a)\}$. Then $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ consists of the following facts:

$$P(a), \quad F(\lambda_1, a), P(\lambda_1), \quad F(\lambda_2, \lambda_1), P(\lambda_2), \quad F(\lambda_3, \lambda_2), P(\lambda_3), \quad \dots$$

Here, by the white-space between atoms we indicate which group of atoms is derived in a particular chase step. Notice again that we implicitly fixed a particular chase sequence $\pi_{\mathfrak{D},\mathcal{O}}$ – indeed, the concretely chosen labeled nulls in each step above are immaterial.

Now recall from Example 3.4 that $q(x) = \exists y, z\,(P(x) \wedge F(y, x) \wedge F(z, y) \wedge P(z))$. Let h be a function such that $h(x) \coloneqq a$, $h(y) \coloneqq \lambda_1$, $h(z) \coloneqq \lambda_2$. Then h is a homomorphism from $q(x)$ to $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ and thus $\mathsf{chase}(\mathfrak{D}, \mathcal{O}) \models q(a)$. By Corollary 3.16 we thus obtain $(\mathfrak{D}, \mathcal{O}) \models q(a)$.

Let us also remark at this point that $\mathsf{chase}(\mathfrak{D}, \mathcal{O}) \models q(\lambda_1)$ holds. However, the statement $(\mathfrak{D}, \mathcal{O}) \models q(\lambda_1)$ is not well-defined, since the assignment $\{x \mapsto \lambda_1\}$ is not an

assignment over $\mathfrak{D}$ at all. Therefore, none of the labeled nulls can be a certain answer, and this justifies the restriction to answer tuples over $\mathsf{adom}(\mathfrak{D})$ in Corollary 3.16. ⊣

We close this section by observing that all the rule-based OMQs we consider are closed under homomorphisms:

Proposition 3.18. *Let $\mathfrak{A}$ be an $\mathbf{S}$-structure and $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ be an OMQ from $(\mathsf{TGD}, \mathsf{UCQ})$. It holds that Q is closed under homomorphisms, i.e., if h is a homomorphism from $\mathfrak{A}$ to another $\mathbf{S}$-structure $\mathfrak{B}$, then for all tuples $\bar{a}$ over $\mathsf{adom}(\mathfrak{A})$,*

$$(\mathfrak{A}, \mathcal{O}) \models q(\bar{a}) \implies (\mathfrak{B}, \mathcal{O}) \models q(h(\bar{a})).$$

An according statement holds for weak homomorphisms assuming that $\mathcal{O}$ and $q(\bar{x})$ are constant-free.

Proof hint. By an easy induction on the length of a chase derivation that witnesses $(\mathfrak{A}, \mathcal{O}) \models q(\bar{a})$. Notice in particular that (U)CQs are closed under homomorphisms and the fact that rule bodies are essentially CQs. □

3.4 Decidable Classes of TGDs

As mentioned in the introductory part of this chapter, query answering under arbitrary sets of TGDs is undecidable [24], and this holds even when the set of TGDs is considered to be fixed [44]. In fact, OMQs based on arbitrary sets of TGDs are even capable of expressing any recursively enumerable query that is closed under homomorphisms [124]. Moreover, in [14] it is shown that any set of TGDs can be encoded into a single rule, which entails that, in general, query answering under singleton sets of TGDs is undecidable as well.

The negative results on query answering under existential rules have led to a flurry of research activity on the identification of classes of sets of TGDs for which query answering is decidable. The purpose of this chapter is to introduce these classes and their guiding principles. By a *decidable class* $\mathcal{L}$ of finite sets of TGDs, we mean a class of finite sets TGDs such that the query answering problem $\mathsf{Eval}(\mathcal{L}, \mathsf{UCQ})$ is decidable. As in the title of this section, we sometimes sloppily call $\mathcal{L}$ a "decidable class of TGDs," though $\mathcal{L}$ in reality, of course, consists of a class of *sets of* TGDs. We first present *abstract* decidable classes of sets of TGDs and then present concrete *syntactic* decidable classes.[1] For the former, the term "abstract" refers to the fact that the distinguishing class property of the sets of TGDs at hand is not necessarily effectively decidable, while, for the latter, the term "syntactic" indicates that the class at hand is specified by effectively decidable ("syntactic") criteria.

The terminology and structure of our synopsis of decidable classes of TGDs is largely based on [117], yet we remark that especially the treatment on syntactic classes is far from being exhaustive. The aim of this section is rather to provide a guideline of the general principles and ideas that lie behind the classes of rules that we are going to encounter in this thesis.

[1] To the best of our knowledge, the categorization into abstract and syntactic classes is due to [14].

3.4.1 Abstract Classes

As mentioned above, by an *abstract class* of TGDs we mean a class of finite sets of TGDs that is distinguished by a property that is not necessarily effectively decidable. According to [14, 117], the three main abstract decidability paradigms present in the literature are: (i) *finite expansion sets* which exhibit the property that the (restricted) chase terminates, (ii) *finite tree-width sets* which have the property that the instance constructed by the chase, though possibly infinite, has finite tree-width, and (iii) *first-order rewritable sets* which ensure that the set of TGDs at hand together with a given (U)CQ can be compiled into a finite first-order query that can be equivalently employed for query answering purposes.

Finite Expansion Sets

Intuitively, a finite expansion set is a set of TGDs such that, when a database is chased with that set of TGDs, after finitely many rule applications, subsequent rule applications become superfluous for query answering purposes (see [117]). In this sense, the chase can be considered to terminate after finitely many steps, and query answering can be performed on the resulting finite instance. The following definition formalizes this intuition:

Definition 3.19. A set of TGDs $\mathcal{O}$ is a *finite expansion set*, if for every database $\mathfrak{D}$, there exists a computable $k \geq 0$ such that there is a homomorphism from $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$.

It is not hard to see that query answering for OMQs based on finite expansion sets is decidable:

Theorem 3.20. *Suppose $\mathcal{O}$ is a finite expansion set, $\mathfrak{D}$ a database, $q(\bar{x})$ a UCQ, and a tuple $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$. Then the problem of deciding whether $(\mathfrak{A}, \mathcal{O}) \models q(\bar{a})$ is decidable.*

Proof. Suppose $k \geq 0$ is chosen such that there is a homomorphism from $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$. Thus, $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$ is a universal model for $\mathfrak{D}$ and $\mathcal{O}$, since it is homomorphically equivalent to $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$. Moreover, $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$ is obviously finite, and so we can decide $(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$ by first computing $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$ and then check whether $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$. Notice that the latter task amounts to the evaluation of a UCQ over a structure, which is clearly decidable (see Theorem 2.15). □

Finite Tree-Width Sets

Unlike finite expansion sets, for finite tree-width sets it may not be possible to restrict oneself to a finite number of rule applications when chasing a database. Rather, the structure constructed by the chase exhibits a "tree-like" shape which can be exploited for the purpose of query answering. Let us formalize this properly:

Definition 3.21. Let $\mathfrak{A}$ be a structure. A *tree decomposition of* $\mathfrak{A}$ is a tuple $\delta = (\mathcal{T}, (X_v)_{v \in T})$, where $\mathcal{T} = (T, E)$ is a rooted tree, and $(X_v)_{v \in T}$ a collection of subsets of $\mathsf{dom}(\mathfrak{A})$, called the *bags* of δ, such that the following conditions are satisfied:

(i) $\mathsf{dom}(\mathfrak{A}) \subseteq \bigcup_{v \in T} X_v$,

(ii) for every fact $R(a_1, \ldots, a_n)$ of $\mathfrak{A}$, there is a $v \in T$ such that $\{a_1, \ldots, a_n\} \subseteq X_v$, and

(iii) for every $a \in \mathsf{dom}(\mathfrak{A})$, the set $\{v \mid a \in X_v\}$ induces a connected subtree of $\mathcal{T}$.

The *width* of δ, denoted $\mathsf{wd}(\delta)$, is $\max\{|X_v| : v \in T\} - 1$. The *tree-width* of $\mathfrak{A}$, denoted $\mathsf{tw}(\mathfrak{A})$, is $\min\{\mathsf{wd}(\delta) \mid \delta \text{ is a tree decomposition of } \mathfrak{A}\}$.

Remark 3.22. We want to point of the following:

▶ The tree-width of a given infinite structure $\mathfrak{A}$ may not exist in the sense that there is no tree decomposition whose bags have finite cardinality at all. In this case, we set $\mathsf{tw}(\mathfrak{A}) := \infty$. In case $\mathsf{tw}(\mathfrak{A}) \leq \infty$, it always holds that $\mathsf{tw}(\mathfrak{A}) \geq 0$, since we require that $\mathsf{dom}(\mathfrak{A})$ is non-empty. Moreover, the fact that the tree $\mathcal{T}$ of δ is rooted is strictly speaking not required, but it is convenient for our purposes to refer to a fixed root node.

▶ Given a tree decomposition $\delta = (\mathcal{T}, (X_v)_{v \in T})$, we call $\mathcal{T}$ the tree δ *is based on*. Moreover, we often abuse terminology and treat δ itself as a tree – for example, when we refer to a "node" of δ, etc., we actually mean a node of $\mathcal{T}$, etc.

▶ Item (iii) of Definition 3.21 is called the *connectivity* property.

Example 3.23. Let $\mathfrak{D} = \{R(a, b), R(b, c), R(c, a)\}$ and consider the singleton set of TGDs $\mathcal{O} = \{R(x, y) \to \exists z\, R(y, z)\}$. Then:

$$\mathsf{chase}(\mathfrak{D}, \mathcal{O}) = \mathfrak{D} \cup \{R(b, \lambda_1), R(c, \lambda_2), R(a, \lambda_3), R(\lambda_1, \lambda_{1,1}), R(\lambda_2, \lambda_{2,1}), R(\lambda_3, \lambda_{3,1}), \ldots\}.$$

Figure 3.1 shows a tree decomposition of $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ that has width two. It is easy to check that there is no tree decomposition of $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ of smaller width, whence we conclude that $\mathsf{tw}(\mathsf{chase}(\mathfrak{D}, \mathcal{O})) = 2$. ⊣

Definition 3.24. Let $\mathcal{O}$ be a finite set of TGDs. We say that $\mathcal{O}$ is a *finite tree-width set*, if for every database $\mathfrak{D}$, there is a $k \geq 0$ such that $\mathsf{tw}(\mathsf{chase}(\mathfrak{D}, \mathcal{O})) \leq k$.

Notice that every finite expansion set is trivially a finite tree-width set.

A class $\mathcal{L}$ of first-order sentences has the *finite tree-width property*, if every sentence $\varphi \in \mathcal{L}$ has a model of finite tree-width. It is a celebrated result of Courcelle [65] that the satisfiability problem for classes of sentences that have the finite tree-width property is decidable – in fact, this holds even for monadic second-order logic. Using Courcelle's theorem we can conclude:

Theorem 3.25. *If $\mathcal{O}$ is a finite tree-width set, $\mathfrak{D}$ a database, $q(\bar{x})$ a UCQ, and $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$, then the problem of deciding whether $(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$ is decidable.*

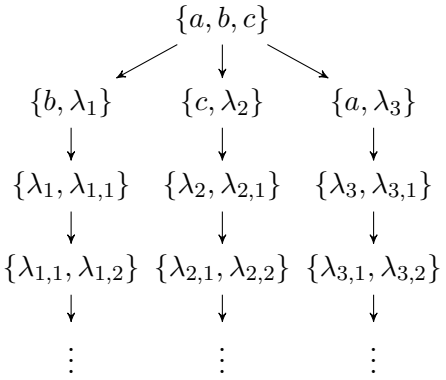

Figure 3.1: A tree decomposition of $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ from Example 3.23 that has the minimal width two.

Proof. Let $\varphi_{\mathfrak{D},\mathcal{O},q(\bar{a})}$ be the sentence

$$\bigwedge \mathfrak{D} \wedge \bigwedge \mathcal{O} \wedge \neg q(\bar{a}),$$

where $\bigwedge \mathfrak{D}$ denotes the conjunction of all facts occurring in $\mathfrak{D}$ seen as atomic formulas. Let $\mathcal{L}$ denote the class of all such sentences $\varphi_{\mathfrak{D},\mathcal{O},q(\bar{a})}$, where $\mathcal{O}$ is a finite tree-width set. We claim that $\mathcal{L}$ has the finite tree-width property. Indeed, if $\varphi_{\mathfrak{D},\mathcal{O},q(\bar{a})} \in \mathcal{L}$ has a model $\mathfrak{A}$, then it is a model of $\mathfrak{D}$ and $\mathcal{O}$. Also recall that $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ is a model of $\mathfrak{D}$ and $\mathcal{O}$. By the universality of $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ (see Lemmas 3.14 and 3.15), it follows that there is a homomorphism h from $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ to $\mathfrak{A}$. We claim that $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ is also a model of $\neg q(\bar{a})$. Indeed, if not, then $\mathsf{chase}(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$, whence $\mathfrak{A} \models q(\bar{a})$ follows since the satisfaction of (U)CQs is easily seen to be closed under homomorphisms. This contradicts the fact that $\mathfrak{A}$ is a model of $\varphi_{\mathfrak{D},\mathcal{O},q(\bar{a})}$. Hence, $\varphi_{\mathfrak{D},\mathcal{O},q(\bar{a})}$ has a model of finite tree-width, whence it follows by Courcelle's result that the satisfiability problem for $\mathcal{L}$ is decidable. It remains to be noticed that $\varphi_{\mathfrak{D},\mathcal{O},q(\bar{a})}$ is unsatisfiable iff $(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$. □

First-Order Rewritable Sets

First-order rewritable sets of TGDs (introduced in the context of description logics [48]) achieve decidability by reducing the query answering problem to the problem of answering a plain first-order query over a given database. More formally:

Definition 3.26. A set of TGDs $\mathcal{O}$ is a *first-order rewritable set*, if for every UCQ q, one can effectively construct a (finite) first-order query $\varphi_{q,\mathcal{O}}$ such that $\varphi_{q,\mathcal{O}}(\mathfrak{D}) = \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$ for every database $\mathfrak{D}$. We say that $\varphi_{q,\mathcal{O}}$ is a *first-order rewriting of q with respect to $\mathcal{O}$*.

First-order rewritable sets have several merits. Firstly, having the plain first-order query $\varphi_{q,\mathcal{O}}$ at hand, one can evaluate $\varphi_{q,\mathcal{O}}$ over $\mathfrak{D}$ using the available relational database management technology, since first-order queries essentially amount to SQL-queries. Secondly, the fact that such a first-order rewriting exists at all has implications in terms

of data complexity: since the first-order rewriting is by definition independent from the data, the data complexity of answering queries under first-order rewritable sets of TGDs matches the data complexity of evaluating first-order queries, which is in AC_0 (cf. Theorem 2.13). The fact that query answering under first-order rewritable sets of TGDs is decidable in combined complexity follows immediately from the fact that the first-order rewriting is required to be effectively constructible by definition. The following theorem summarizes these facts:

Theorem 3.27. *Query answering under first-order rewritable sets is decidable and, moreover, feasible in* AC_0 *in terms of data complexity.*

We are going to revisit the concept of *first-order rewritability* in the context of OMQs in Chapter 5, where we are going to study first-order rewritability as a decision problem. Therefore, we shall omit more details at this point and refer the reader to Chapter 5 for more information on first-order rewritability.

3.4.2 Syntactic Classes

We have seen in the previous subsection that there are semantic classes of sets of TGDs which guarantee that query answering under the corresponding class is decidable. However, it turns out that the properties we presented are *inherently* semantic – i.e., for each of the presented classes, the problem of deciding whether a given set of TGDs belongs to that class is undecidable [14].

Therefore, one aims to define syntactic classes of sets of TGDs that ensure decidability of query answering. We are now going to give an overview on these concrete syntactic classes that have been studied in the literature. The presentation here is not exhaustive, but focuses rather on general principles and recurring themes that appear in the exploration of new decidable classes. Moreover, we focus on those classes which we are going to encounter at a later point in this thesis.

Linear Sets of TGDs

A *linear* TGD [45] is a TGD that contains only one atom in its body and one atom in its head – accordingly, we call a set of TGDs with that property *linear*. Linear TGDs extend *inclusion dependencies* [74] which are, roughly speaking, rules that state that values of a certain column should be contained in the values of another column. In general, sets of linear TGDs are not finite expansion sets. Indeed, consider the singleton set $\mathcal{O} \coloneqq \{R(x,y) \rightarrow \exists z\, R(y,z)\}$ and the database $\mathfrak{D} \coloneqq \{R(a,a)\}$. Then $\mathsf{chase}(\mathfrak{D},\mathcal{O}) = \{R(a,\lambda_1), R(\lambda_1,\lambda_2), \ldots\}$, and we can easily see that there is no $k \geq 0$ such that $\mathsf{chase}(\mathfrak{D},\mathcal{O})$ maps to $\mathsf{chase}^k(\mathfrak{D},\mathcal{O})$. Thus, $\mathcal{O}$ is not a finite expansion set, yet linear.

On the other hand, it can be shown that sets of linear TGDs are always first-order rewritable. This is established in [45] by establishing the so-called *bounded derivation depth property* (BDDP) for linear TGDs. Roughly speaking, the BDDP states that, for the purpose of query answering, it suffices to rely on a finite initial part of the chase, where the size of that part depends only on the query and the set of TGDs, but not on

the given database. Using this initial part, one is able to construct a first-order rewriting of the query with respect to the given ontology; see [45] for more details.[2] The fact that sets of linear TGDs are first-order rewritable sets immediately implies that query answering under such sets is decidable and, moreover, even in AC_0 with respect to data complexity. For combined complexity, it turns out that query answering under linear sets of TGDs is PSPACE-complete. This result is obtained by adjusting a corresponding proof of PSPACE-completeness from [97] for the case of inclusion dependencies.

Apart from the fact that linear sets of TGDs are first-order rewritable sets, they also turn out to be finite tree-width sets. This follows immediately from the fact that linear TGDs are also *guarded* and the fact that guarded sets of TGDs are finite tree-width set. We shall introduce the concept of guardedness below.

Notation. We write L for the class of all finite sets of linear TGDs.

Acyclic (Non-Recursive) Sets of TGDs

For defining this class, we need one additional notion first. Let $\mathcal{O}$ be a set of TGDs. The *predicate graph* of $\mathcal{O}$ is the directed graph whose node set equals the set of relation names occurring in $\mathcal{O}$, and a relation name R has an edge to a relation name S in that graph iff there is a rule in $\mathcal{O}$ whose body mentions R and whose head mentions S. We say that $\mathcal{O}$ is *acyclic* [63] (or *non-recursive*) if its dependency graph has no directed cycles. For example, the set of TGDs

$$R(x,y), T(y,v) \rightarrow \exists z\, S(y,z), \quad S(y,z), A(y) \rightarrow G(y),$$

is non-recursive, while the set consisting solely of

$$R(x,y), T(y,z) \rightarrow T(x,z)$$

is not. Notice that the notion of acyclicity refers to an entire set of TGDs and is not defined *locally* on a rule level. We call such syntactic properties *global* (for example, linearity of TGDs is, in contrast, a local property). The problem of deciding whether a given set of TGDs is acyclic is clearly feasible in polynomial, since it amounts to successively computing reachable nodes in a graph.

Intuitively, acyclicity/non-recursiveness guarantees that the structure of the rules exhibits no recursive nature. In this respect, acyclic TGDs can be seen as an extension of *non-recursive* Datalog queries that are defined *mutatis mutandis* to acyclic sets of TGDs (see, e.g., [55, 112, 121]). It is thus easy to see that the chase under such sets always terminates, and so acyclic sets are clearly finite expansion sets (and thus also finite tree-width sets). In fact, it is also not hard to check that acyclic sets of TGDs are also first-order rewritable sets, as one can employ a simple backward chaining algorithm: with such an algorithm, one can, starting off with a UCQ q and a set of acyclic TGDs $\mathcal{O}$, successively apply rewriting steps in order to rewrite q according to $\mathcal{O}$ until no more rewriting steps can be applied. This process terminates, since acyclic sets of TGDs do

[2] In fact, the BDDP turns out to be equivalent to first-order rewritability [80]

not exhibit recursion at all and a similar algorithm for non-recursive Datalog queries is folklore [121]. The resulting query $q_{\mathcal{O}}$ that is obtained successively from these steps is then a first-order rewriting of q with respect to $\mathcal{O}$.

The fact that acyclic sets of TGDs are first-order rewritable sets implies that their associated query answering problem is in AC_0 in terms of data complexity. The combined complexity of them is investigated in [103], where it is shown that answering queries under acyclic sets of TGDs is NExpTime-complete.

Notation. We write NR for the class of all finite non-recursive sets of TGDs.

We remark that the type of acyclicity discussed here is the most basic form of constraining sets of rules to be finite expansion sets. Acyclicity has been extended to the notions of *weak acyclicity* [76] (see also below), *stratification* [69], *safety* [107], *acyclicity of a graph of rule dependencies* [13, 16], *joint acyclicity* [100], *super-weak acyclicity* [105], and *model-faithful acyclicity* and *model-summarizing acyclicity* [90]. We refer the reader to [90, 122] for a comprehensive overview of these concepts.

Guarded Sets of TGDs

Sets of TGDs that fall into the *guarded* family of classes are arguably among the most intensively studied classes in this thesis. We say that a TGD τ is *guarded*, if it contains an atom in its body that contains all the variables in the body as arguments – such an atom is called a *guard* of τ, and when we speak of *the* guard of τ, we shall simply mean the left-most among all guards to avoid ambiguity. For example, the rule $T(x, y, z), P(z) \rightarrow \exists v\, R(x, v)$ is guarded – its guard is $T(x, y, z)$ – while $T(x, y), T(y, z) \rightarrow T(x, z)$ is not. A set of TGDs is guarded iff every TGD contained in it is guarded. Notice that guardedness is a local property.

Notation. We write G for the class of all finite sets of guarded TGDs.

The notion of guardedness for TGDs matches a corresponding notion for Datalog rules [79]: a *guarded Datalog rule* is one that has an atom in its body containing all the body variables as arguments, and a *guarded Datalog query* consists of guarded Datalog rules only.

Guarded TGDs have been intensively studied in [44] and owe their name to their close relationship to the *guarded fragment* of first-order logic that has been introduced in [5]. Roughly speaking, the guarded fragment of first-order logic is a fragment of first-order logic devised to inherit the convenient computational and model-theoretic properties of modal logic. Syntactically, the guarded fragment is obtained by relativizing quantifier occurrences to take either form

$$\forall \bar{y}\, (\alpha(\bar{x}, \bar{y}) \rightarrow \varphi(\bar{y})) \quad \text{or} \quad \exists \bar{y}\, (\alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{y})),$$

where $\alpha(\bar{x}, \bar{y})$ is an atomic formula (possibly an equality atom) that mentions all the free variables of $\varphi(\bar{y})$. Intuitively, the relativization of the quantifiers ensures that guarded

quantification occurs locally in the sense that one quantifies over elements which are explicitly related via the relation denoted by $\alpha(\bar{x}, \bar{y})$.

The satisfiability problem for guarded sentences is decidable – in fact, 2ExpTime-complete as shown in [86]. The decidability of the guarded fragment is based on the fact that satisfiable guarded sentences always have a "tree-like" model, that is, one whose tree-width is finite and depends only on the signature over which the formula at hand is formulated. Moreover, it turns out that there is a natural notion of *guarded bisimulation* – which is a generalization of standard bisimulation from the modal-logic world – and it turns out that guarded first-order logic is in fact the fragment of first-order logic which is invariant under guarded bisimulation [5, 88]. This result again shows the tight relationship of the guarded fragment to modal logic, since modal logic is known to be, according to a classical result by van Benthem [129], the bisimulation invariant fragment of first-order logic.

Turning our attention to guarded sets of TGDs again, just as the guarded fragment generalizes modal logics, guarded TGDs generalize rules formulated in the description logics that belong to the $\mathcal{EL}$-family [10, 11]. Essentially, guarded TGDs allow for the use of relation names whose arity is greater than two.

It is easy to see that every linear TGD is also guarded, since the single body atom can be taken as a guard. Thus, sets of guarded TGDs are clearly not finite expansion sets. Moreover, just as ontologies formulated in $\mathcal{EL}$, guarded sets of TGDs are not first-order rewritable sets – for more details we refer to Chapter 5. The fact that sets of guarded TGDs fall into the class of finite tree-width sets is established in [44], where it is shown that the members of the more general class of weakly guarded sets of TGDs (see below) are finite tree-width sets.

It can be shown that OMQs based on guarded TGDs can be effectively rewritten into equivalent Datalog queries [17, 85]. This entails that the data complexity of answering queries under guarded sets of TGDs is in PTime, since the data complexity of answering Datalog queries is in PTime (Theorem 2.18). In [45], PTime-membership with respect to data complexity is shown by relying on first-principled methods rather than rewritings. Roughly speaking, the authors show that, to evaluate a query under a set of guarded TGDs, one can evaluate the query on a finite portion of the chase whose size depends on the relational schema and the query only, and which can be constructed in polynomial time in the size of the data. PTime-hardness of data complexity can be obtained by reducing from the problem of deciding whether a propositional logic program entails a propositional variable [45, 66].

A detailed analysis of the combined complexity of query answering under guarded TGDs can be found in [44]. It is shown that the combined complexity of query answering under guarded TGDs is 2ExpTime-complete in general. The upper bound can be obtained by relying on the decision procedure for the satisfiability problem of the guarded fragment of first-order logic, while the lower bound is obtained by simulating the behavior of an alternating Turing machine running in exponential space. Since alternating exponential space equals 2ExpTime, the lower bound follows. In case one considers query answering under guarded TGDs when the arity of the relational schema is fixed, it turns out that

one cannot simply rely on the decision procedure for the guarded fragment. For this case, in [44], the authors devise a sophisticated decision procedure which runs in ExpTime. The upper bounds in [44] are, in fact, shown for the case of weakly guarded sets of TGDs that generalize guarded TGDs (see below), and are thus immediately inherited for the more specific case of guarded TGDs.

Frontier-Guarded Sets of TGDs

In [14], the authors provide a more generic class of guarded TGDs, called *frontier-guarded TGDs*.[3] Recall that the frontier variables of a TGD are all the variables from the body which are propagated to the head. We say that a TGD τ is *frontier-guarded*, if it contains an atom in its body which has all the frontier variables as arguments – such an atom is called *frontier-guard* of τ, and when we speak about *the* frontier-guard, we shall mean the left-most one to avoid ambiguity. Accordingly, a set of TGDs is frontier-guarded if every TGD contained in it is frontier-guarded. Notice that every guarded TGD is trivially frontier-guarded. Moreover, notice that rules which propagate only a single variable to the head are also trivially frontier-guarded, but not necessarily guarded. For example, the rule $R(x, y), T(y, z) \rightarrow A(y)$ is frontier-guarded, but not guarded.

Notation. We write FG for the class of all finite frontier-guarded sets of TGDs.

It is clear that frontier-guarded sets of TGDs are neither first-order rewritable nor finite expansion sets, since they extend the class of guarded sets of TGDs. However, in [14] it is shown that frontier-guarded sets are, as in the case of guarded sets of TGDs, always finite tree-width sets.

Concerning complexity, it turns out that query answering under frontier-guarded sets of TGDs matches, regarding the case of schemas of bounded width, the complexity of answering queries under guarded TGDs, i.e., 2ExpTime-complete. The lower bound is inherited from the guarded case, while the upper bound is shown in [15] and exploits a result from [18] concerning the complexity of answering arbitrary conjunctive queries against arbitrary guarded first-order theories. Moreover, query answering under frontier-guarded sets of TGDs is PTime-complete in terms of data complexity [15]. Answering queries against sets of frontier-guarded TGDs remains 2ExpTime-hard even when the arity of the used schema is bounded [15, 18].

Sticky Sets of TGDs

Sticky sets of TGDs were introduced in [47, 117] with the aim of providing a class of sets of TGDs with tractable data complexity, yet that allows one to express more complex joins in rule bodies than other decidable classes of sets of TGDs. For the definition of stickiness, we need some additional concepts.

Recall that we allow the use of equality atoms in rule bodies for technical reasons and to express $\top$. We call an ontology *equality-free*, if none of its TGDs has equality

[3] We remark that this fragment is also implicitly treated in [127]

atoms in its body, except for equality atoms of the form $x = x$, where x is a variable. We can safely eliminate those equality atoms that are different from $x = x$ by exhaustively identifying variables and constants as dictated by the equality atoms. Given a set of TGDs $\mathcal{O}$, we call the *result of $\mathcal{O}$ by identifying variables* the ontology $\mathcal{O}'$ that contains, for each $\tau \in \mathcal{O}$, a TGD τ' that results by identifying equalities accordingly, provided τ' has no equality atom $a = b$ in its body, where $a \neq b$ are constants from const (the body of such a TGD can never have a model).

Given a schema $\mathbf{S}$, by a *position* we mean an expression of the form $R[i]$, where $R/n \in \mathbf{S}$ and $1 \leq i \leq n$. Given a relational atom α occurring in τ, we denote by $\mathsf{pos}(\alpha, v)$ the set of positions at which v occurs in α – formally, for a relational atom $R(t_1, \ldots, t_n)$, we say that t_j *occurs at position* $R[i]$ iff $i = j$ and $R = S$.

Let $\mathcal{O}$ be a finite set of TGDs whose only equality atoms in rule bodies are of the form $x = x$. We apply an inductive marking procedure on the variables occurring in the rules of $\mathcal{O}$ as follows:

(i) for each $\tau \in \mathcal{O}$ and each variable v in its body, if there is an atom α in the head of τ that has no occurrence of v, then each occurrence of v in a relational atom in the body of τ is marked.

(ii) for each pair $(\tau, \tau') \in \mathcal{O} \times \mathcal{O}$, each atom α in the head of τ, and each frontier variable v occurring in α, if there exists an atom β in the body of τ' in which a marked variable occurs at each position of $\mathsf{pos}(\alpha, v)$, then each occurrence of v in a relational atom in the body of τ is marked.

We say $\mathcal{O}$ is *sticky* if there is no $\tau \in \mathcal{O}$ such that a marked variable occurs in its body more than once. If $\mathcal{O}$ is a set of TGDs that does have arbitrary equality atoms in rule bodies, then we call $\mathcal{O}$ *sticky*, if the result of $\mathcal{O}$ by identifying variables is sticky. Notice that stickiness is a global property.

Notation. We write S for the class of all (finite) sticky sets of TGDs.

Example 3.28. Consider the following set of rules:

$$\begin{aligned} R(x, y), R(y, z) &\rightarrow \exists v\, S(y, z, v), \\ T(x, y) &\rightarrow \exists z, w\, V(x, z, w), \\ V(x, y, z) &\rightarrow \exists v\, (T(x, v) \wedge R(z, v)). \end{aligned}$$

In a first step, we mark the following variables (marked variable occurrences are indicated by a "hat" on top of them):

$$\begin{aligned} R(\hat{x}, y), R(y, z) &\rightarrow \exists v\, S(y, z, v), \\ T(x, \hat{y}) &\rightarrow \exists z, w\, V(x, z, w), \\ V(\hat{x}, \hat{y}, \hat{z}) &\rightarrow \exists v\, (T(x, v) \wedge R(z, v)). \end{aligned}$$

In a second step, we also mark the variable x in the body of the second rule:

$$\begin{aligned} R(\hat{x}, y), R(y, z) &\rightarrow \exists v\, S(y, z, v), \\ T(\hat{x}, \hat{y}) &\rightarrow \exists z, w\, V(x, z, w), \\ V(\hat{x}, \hat{y}, \hat{z}) &\rightarrow \exists v\, (T(x, v) \wedge R(z, v)). \end{aligned}$$

Now notice that, in every rule body, each marked variable appears only once. Hence, we can conclude that $\mathcal{O}$ is a sticky set of TGDs. ⊣

It can easily be seen that the property of stickiness can be checked in polynomial time, since the above procedure terminates in at most polynomially many steps due to the fact that, at each step, either a new body variable is marked, or the marking is finished.

Sticky sets are not finite tree-width sets and are neither finite expansion sets [47]. In [47, 117] it is shown that sticky sets of TGDs are actually first-order rewritable sets, and hence the query answering under sticky sets of TGDs is in AC_0 with respect to data complexity. Moreover, it is shown that the combined complexity of query answering under such sets of TGDs is complete for ExpTime. The ExpTime upper bound is obtained by providing a query answering algorithm that runs in NP with access to an oracle running in alternating polynomial space. Since alternating polynomial space equals ExpTime, and since $\mathrm{NP}^{\textsc{ExpTime}} = \textsc{ExpTime}$, the claimed upper bound follows. The ExpTime lower bound is obtained by reducing from the so-called *fact inference problem* for Datalog programs. This problem receives as input a set of Datalog rules Π, a database $\mathfrak{D}$, and a fact α, and the question is whether $(\mathfrak{D}, \Pi) \models \alpha$ holds (here, we view the Datalog rules as first-order sentences; see Subsection 2.3.3).

Weakly (Frontier-)Guarded Sets of TGDs

Notice that the definitions of the classes of (frontier-)guarded TGDs rule out the possibility to include arbitrary plain Datalog rules into ontologies. The purpose of *weakly guarded* [44] and *weakly frontier-guarded* [15] sets of TGDs is to allow for the limited use of non-guarded rules that do not interfere with the creation of labeled nulls during the chase procedure in a "dangerous" manner. Roughly speaking, a TGD τ is *weakly guarded* (*weakly frontier-guarded*, respectively) if it has an atom in its body that contains all the body variables (frontier variables, respectively) as arguments that appear at positions in the body of τ where there may occur labeled nulls during the construction of the chase [44]. Let us formalize this intuition. Let $\mathcal{O}$ be an equality-free finite set of TGDs. We inductively define the set of *affected positions* as follows:

- If π is a position such that in some head of a TGD an existentially quantified variable occurs at π, then π is affected.
- If π is a position such that in some head of a TGD τ a frontier variable x occurs at π and if x occurs in the body of τ at affected positions only, then π is affected.

We say that a set of TGDs $\mathcal{O}$ is *weakly guarded* (*weakly frontier-guarded*, respectively), if every rule $\tau \in \mathcal{O}$ has an atom in its body that contains all the body variables (frontier

variables, respectively) of τ as arguments that occur at affected positions only – such an atom is called a *weak guard* (*weak frontier-guard*, respectively). If $\mathcal{O}$ is not equality-free, then $\mathcal{O}$ is *weakly (frontier-)guarded* if the result of $\mathcal{O}$ by identifying variables is weakly (frontier-)guarded.

Notice that the property of weak (frontier-)guardedness is a global one. It is easy to see that deciding whether a given set of TGDs is weakly (frontier-)guarded is feasible in polynomial time, since computing the set of affected positions is.

Example 3.29. Consider the set $\mathcal{O}$ consisting of the two rules

$$R(x,y), S(y,z) \rightarrow \exists v\, S(x,v), \qquad R(x,y), S(y,z) \rightarrow T(x,z).$$

The position $S[2]$ is affected due to the existentially quantified variable v in the first rule. Thus, $T[2]$ is affected as well, since z only occurs in the second rule at the affected $S[2]$. It is easy to see that $\mathcal{O}$ is weakly-guarded: the first and the second rule have the weak-guard $S(y,z)$. However, $\mathcal{O}$ is clearly not guarded. ⊣

It is easy to see that weakly (frontier-)guarded sets of TGDs extend (frontier-)guarded sets of TGDs. Hence, it is immediate that the members of neither of them are finite expansion sets or first-order rewritable sets. However, weakly guarded sets of TGDs are finite tree-width sets [44], and this holds true for weakly frontier-guarded sets of TGDs as well [15].

The complexity of answering queries under weakly guarded sets of TGDs is investigated in [44], where it is shown that answering queries against weakly guarded sets of TGDs is 2ExpTime-complete in general, and ExpTime-complete if the width of the underlying schema is assumed to be bounded. Moreover, the problem is ExpTime-complete with respect to data complexity. Notice that this is in contrast to the PTime-completeness of the data complexity of answering queries under guarded TGDs. Hence, OMQs based on weakly guarded sets of TGDs cannot, unlike those based on guarded TGDs, rewritten into Datalog queries due to the exponential gap concerning data complexity.

For the case of weakly frontier-guarded sets of TGDs, in [15] it is shown that answering queries under weakly frontier-guarded sets of TGDs is 2ExpTime-complete with respect to combined complexity, even for schemas of bounded width. This result is obtained by reducing the problem to that of answering queries under frontier-guarded sets of TGDs. Moreover, it turns out that the problem is also ExpTime-complete with respect to data complexity.

Other Weak Classes

The definition of weakly (frontier-)guarded TGDs relax the definition (frontier-)guarded TGDs by carefully taking into account the evolution of labeled nulls within the construction of the chase. This allows one to arrive at classes of TGDs that are still able to express Datalog queries, yet that have a decidable query answering problem.

It turns out that similar ideas can be exploited for other classes. We are not going to encounter the underlying details of these classes in this thesis, and we therefore restrict

ourselves here to a brief summary of results on them, but we do not bother the reader with their detailed definitions.

Among these classes are the classes of *weakly acyclic* sets of TGDs [76] and *weakly sticky* sets of TGDs [47, 117]. The former has actually historically been the first class of TGDs of this kind and has been introduced in [76] as an extension of acyclic TGDs in the context of data exchange. Weakly acyclic sets of TGDs belong to the class of finite expansion sets, and their complexity is pinpointed in [47, 117]. It turns out that query answering under weakly acyclic sets of TGDs is 2ExpTime-complete in terms of combined complexity. The upper bound is implicit in [76] and explicitly stated in [117]. A matching 2ExpTime lower bound was subsequently established in [47, 117]. Regarding data complexity, it turns out that answering queries under weakly acyclic sets of TGDs is PTime-complete [76]. The upper bound is implicit in [76], while the lower bound follows immediately from the PTime-completeness of the fact inference problem for fixed Datalog queries [66]. Just as weakly (frontier-)guarded sets of TGDs can be used to express Datalog queries, weakly acyclic sets of TGDs can be as well. Thus, it should be clear that, unlike acyclic sets of TGDs, they cannot be first-order rewritable in general, since Datalog queries are not first-order rewritable. For more details on first-order rewritability we again refer to Chapter 5.

The class of weakly sticky sets of TGDs is defined in [47, 117] and extends the class of sticky sets of TGDs. It can be shown that query answering under such sets is 2ExpTime-complete in terms of combined, and PTime-complete in terms of data complexity. Again, notice that weakly sticky sets are clearly not first-order rewritable sets, as OMQs based on them are able to express all Datalog queries.

Warded Sets of TGDs

We have observed above that the "weak" classes of TGDs introduced in the literature take a more liberal stance on the way how they enforce their syntactic restrictions than the "non-weak" classes. This results in languages that are able to capture Datalog queries in their full generality, i.e., all the weak classes above capture Datalog. On the downside of this is their combined complexity: the classes we have encountered all have a 2ExpTime-complete query answering problem in terms of combined complexity. *Warded* sets of TGDs have been introduced in [84] with the aim to create a language that is expressive enough to capture Datalog, yet that restricts its syntax in order to obtain tractable data complexity, as well as a reasonable combined complexity that makes it amenable to practical implementation.

To formally define wardedness, we need some additional technical notions first. Let $\mathcal{O}$ be an equality-free finite set of TGDs. We call a position a *non-affected position* of $\mathcal{O}$ if it is not an affected position of $\mathcal{O}$. Consider a TGD $\tau \in \mathcal{O}$ and let x be a variable occurring in the body of τ.

- We say that x is *harmless* if it has at least one occurrence in the body of τ at a position which is non-affected. We call x *harmful* if it is not harmless.
- We call x *dangerous* if it is harmful and belongs to the frontier variables of τ.

We say that $\mathcal{O}$ is *warded* if, for each $\tau \in \mathcal{O}$, either there are no dangerous variables in the body of τ, or there exists an atom $\alpha \in \mathsf{body}(\tau)$, called a *ward*, such that the following conditions hold:

(i) All the dangerous variables that occur in the body of τ also occur in α.

(ii) Each variable of $\mathsf{var}(\alpha) \cap \mathsf{var}(\mathsf{body}(\tau) \setminus \{\alpha\})$ is harmless.

Finally, a finite set of TGDs $\mathcal{O}$ that is not equality-free is called *warded*, if the result of $\mathcal{O}$ by identifying variables is warded.

Notation. We denote by W the class of all finite warded sets of TGDs.

Example 3.30. An OWL 2 QL ontology can be stored in a database using atoms of the form $\mathsf{Restriction}(c, p)$ stating that the class c is a restriction of the property p, $\mathsf{SubClass}(c, c')$ stating that c is a subclass of c', and $\mathsf{Inverse}(p, p')$ stating that p is the inverse property of p'. We can then compute all the logical inferences of the given ontology using the TGDs:

$$\begin{aligned}
\mathsf{SubClass}(x, y) &\rightarrow \mathsf{SubClass}^*(x, y),\\
\mathsf{SubClass}^*(x, y), \mathsf{SubClass}(y, z) &\rightarrow \mathsf{SubClass}^*(x, z),\\
\underline{\mathsf{Type}(x, y)}, \mathsf{SubClass}^*(y, z) &\rightarrow \mathsf{Type}(x, z),\\
\underline{\mathsf{Type}(x, y)}, \mathsf{Restriction}(y, z) &\rightarrow \exists w\, \mathsf{Triple}(x, z, w),\\
\underline{\mathsf{Triple}(x, y, z)}, \mathsf{Inverse}(y, w) &\rightarrow \mathsf{Triple}(z, w, x),\\
\underline{\mathsf{Triple}(x, y, z)}, \mathsf{Restriction}(w, y) &\rightarrow \mathsf{Type}(x, w).
\end{aligned}$$

The first two TGDs are responsible for computing the transitive closure of the $\mathsf{SubClass}$ relation, while the third TGD transfers the class type, i.e., if a is of type b and b is a subclass of c, then a is also of type c. Moreover, the fourth TGD states that if a is of type b and b is the restriction of the property p, then a is related to some c via the property p, which is encoded by the atom $\mathsf{Triple}(a, p, c)$. Analogously, the last two TGDs encode the usual meaning of inverses and the effect of restrictions on types.

The above set of TGDs is clearly warded – the underlined atoms are the wards. If no atom is underlined, then there are no dangerous variables. A variable in an atom with predicate $\mathsf{Restriction}$, $\mathsf{SubClass}$, $\mathsf{SubClass}^\star$, or $\mathsf{Inverse}$, is trivially harmless. The frontier variables that appear at $\mathsf{Type}[1]$, $\mathsf{Triple}[1]$, or $\mathsf{Triple}[3]$ are dangerous. ⊣

Notice that warded sets of TGDs are also weakly frontier-guarded, but not vice versa (in fact, it is easy to see that there are guarded sets of rules that are not warded). In [84] it is shown that query answering under warded sets of TGDs is ExpTime-complete in combined, and PTime-complete in data complexity. The ExpTime-hardness of combined and PTime-hardness of data complexity follows from the fact that OMQs based on warded TGDs allow us to express all Datalog queries. Matching upper bounds are shown by employing space-bounded alternating algorithms. We are going to revisit these results in Chapter 6 when we identify a space-efficient fragment of the class W.

We also mention that warded sets of TGDs represent the logical formalism behind the implementation of Vadalog [26, 27], a commercially used reasoner.

CHAPTER 4

Containment and Equivalence for OMQs

This chapter focuses on a crucial static analysis task for OMQs, namely *containment*: for two OMQs Q_1 and Q_2 with data schema $\mathbf{S}$, does $Q_1(\mathfrak{D}) \subseteq Q_2(\mathfrak{D})$ hold for every database $\mathfrak{D}$ over $\mathbf{S}$? A task closely related to containment is that of *equivalence*: given Q_1 and Q_2 with data schema $\mathbf{S}$, does $Q_1 \equiv Q_2$ hold, i.e., does $Q_1(\mathfrak{D}) = Q_2(\mathfrak{D})$ hold for every $\mathbf{S}$-database $\mathfrak{D}$? It is clear that equivalence can be solved by calling an oracle for containment twice. Hence, the focus in this chapter arguably lies on containment.

Apart from the traditional applications of containment, such as query optimization or view-based query answering, it has been recently shown that containment of OMQs – and the methods devised to solve it – have applications in other important static analysis tasks, namely, distribution over components [33], and first-order rewritability [34]. Surprisingly, despite its prominence, no work to date has carried out an in-depth investigation of containment for OMQs based on TGDs.

As one might expect, when considered in its full generality, i.e., without any restrictions on the set of TGDs, the containment problem for OMQs is undecidable. To understand, on the other hand, which restrictions on the TGDs used lead to decidability, we recall the two main reasons that render the general containment problem undecidable:

Undecidability of query evaluation As we have seen in Chapter 3, evaluation of OMQs is, in general, undecidable, and it can be reduced to containment. More precisely, containment of OMQs is undecidable whenever query evaluation for at least one of the involved languages (i.e., the language of the left-hand or the right-hand side query) is undecidable.

Undecidability of containment for Datalog Decidability of query evaluation does not ensure decidability of query containment. A prime example is Datalog – Datalog containment is undecidable [126]. Thus, OMQ-containment is undecidable if the involved languages extend Datalog.

	Arbitrary arity	*Bounded arity*
Linear	PSpace-complete	Π_2^{P}-complete
Sticky	coNExpTime-complete	Π_2^{P}-complete
Non-recursive	In coNExpTime$^{\mathrm{NP}}$ and PTime$^{\mathrm{NExpTime}}$-hard	In coNExpTime$^{\mathrm{NP}}$ and PTime$^{\mathrm{NExpTime}}$-hard
Guarded	2ExpTime-complete	2ExpTime-complete
Frontier-guarded	2ExpTime-complete	2ExpTime-complete

Table 4.1: Complexity of the containment problem for OMQs using different classes of sets of TGDs.

In view of the above observations, we focus on languages that have a decidable query evaluation problem, and that do not extend Datalog. In Chapter 3 we have already encountered the main classes of TGDs which give rise to OMQ languages with these desirable properties (i) (frontier-)guarded sets of TGDs, which subsume linear TGDs, (ii) non-recursive (i.e., acyclic) sets of TGDs, and (iii) sticky sets of TGDs.

While the decidability of containment for these OMQ languages can be established via translations into query languages with a decidable containment problem, such translations do not lead to optimal complexity upper bounds (details are given below). Therefore, the main goal of this chapter is to develop specially tailored decision procedures for the containment problem under the OMQ languages in question, and, ideally, obtain precise complexity bounds.

Contributions As said above, the goal of this chapter is to provide specifically tailored algorithms for solving the containment problems for OMQ languages that are based on the main classes of TGDs for which query evaluation is decidable. The main results concerning containment problems for OMQ languages treated in this chapter are depicted in Table 4.1: there, we state the complexity of containment – both for the general and the case of schemas of bounded width – for different classes of TGDs, assuming that the query component in the OMQs are UCQs. We divide the contributions of this chapter as follows:

Linear, non-recursive, and sticky sets of TGDs The OMQ languages based on linear, non-recursive, and sticky sets of TGDs share a useful property: they are *UCQ-rewritable* (implicit in [82]), that is, their OMQs falling into one of these languages can be (effectively) rewritten into equivalent UCQs. This property immediately yields decidability for their associated containment problems, since containment of UCQs is decidable [125].

However, the obtained complexity bounds are not optimal, since the UCQ-rewritings are unavoidably very large [82]. To obtain more precise bounds, we reduce containment to query evaluation, an idea that is often applied in the context of query containment (see, e.g., [54, 55, 125]).

Consider a UCQ-rewritable OMQ language $\mathcal{L}$. If Q_1 and Q_2 belong to $\mathcal{L}$, both with data schema $\mathbf{S}$, then we can establish a *small witness property*, which states that non-containment of Q_1 in Q_2 can be witnessed via a database over $\mathbf{S}$ whose size is bounded by an integer $k \geq 0$, where k is the maximal size of a disjunct in a UCQ-rewriting of Q_1. For linear TGDs, such an integer k is polynomial, but for non-recursive and sticky sets of TGDs it is exponential (implicit in [82]). The small witness property allows us to devise a simple non-deterministic algorithm, which makes use of query evaluation as a subroutine, for checking non-containment of Q_1 in Q_2: guess an $\mathbf{S}$-database $\mathfrak{D}$ of size at most k, and then check if there is a certain answer in $Q_1(\mathfrak{D})$ that is not a certain answer in $Q_2(\mathfrak{D})$. This algorithm allows us to obtain optimal upper bounds for OMQs based on linear and sticky sets of TGDs. However, the exact complexity of containment for OMQs based on non-recursive sets of TGDs remains open:

- For OMQs based on linear TGDs, the containment problem is in PSpace, and in Π_2^{P} if the arity is fixed. PSpace–hardness is shown by reduction from query evaluation, while the Π_2^{P}-hardness is implicit in [35].
- For OMQs based on sticky sets of TGDs, the problem is in coNExpTime, and in Π_2^{P} for schemas of bounded arity. Hardness for coNExpTime is shown by exploiting a tiling problem of the exponential grid, while Π_2^{P}-hardness follows from [35].
- For OMQs based on non-recursive sets of TGDs, containment is in $\text{coNExpTime}^{\text{NP}}$ and hard for $\text{PTime}^{\text{NExpTime}}$, even for schemas of bounded arity. The lower bound is shown by exploiting a suitable tiling problem from [72].

Regarding OMQs based on non-recursive sets of TGDs, although our upper bound is not optimal, it is nearly optimal. Indeed, $\text{NExpTime}^{\text{NP}}$, which forms the Δ_2-level of the exponential hierarchy (EH), and $\text{PTime}^{\text{NExpTime}}$, which forms the Δ_2-level of the *strong* EH,[1] are tightly related: if the oracle access in $\text{NExpTime}^{\text{NP}}$ is restricted "too much," then it collapses to $\text{PTime}^{\text{NExpTime}}$ [92].

(Frontier-)guarded sets of TGDs The OMQ language based on guarded TGDs is not UCQ-rewritable, which forces us to develop different tools to study its containment problem. Let us remark that guarded OMQs can be rewritten as guarded Datalog queries (by exploiting the translations devised in [17, 85]), for which containment is decidable in 2ExpTime [40]. But, again, the known rewritings are very large [85], and hence the reduction of containment for guarded OMQs to containment for guarded Datalog does not yield optimal upper bounds. To solve the containment problem for guarded OMQs, we devise procedures based on two-way alternating parity tree automata. More precisely, we first devise automata-based procedures for containment of guarded OMQs that have atomic queries as their query component. To this end, we first show that such OMQs exhibit a convenient tree witness property: if Q_1 is not contained in Q_2, then this is

[1] The strong EH collapses to its Δ_2-level [92].

witnessed by a "tree-like" database. We then use this property to reduce containment of Q_1 in Q_2 to the question whether the language of a suitable two-way alternating parity tree automaton is empty. This reduction then proves that containment for guarded OMQs (that have an atomic query) is in 2ExpTime, and in ExpTime when we assume that the arity of the underlying schema is bounded. Appropriate lower bounds are inherited from [34].

Having a procedure for containment of guarded OMQs with atomic queries at hand, we solve the task of containment between frontier-guarded OMQs (and guarded OMQs with an arbitrary CQ as a query) by reducing it the the containment problem for the atomic case. The reduction is based on the technique of *treeification* [18, 20] and is inspired by a similar translation of sentences of guarded negation least fixed point logic (GNFP) into sentences of guarded least fixed point logic (GFP) [20]. Although this reduction is exponential, it will still provide us with a 2ExpTime upper bound for containment of frontier-guarded OMQs (and likewise for OMQs based on guarded TGDs with a CQ as a query).

Combining languages The above complexity results refer to the containment problem relative to a certain OMQ language $\mathcal{L}$, i.e., both queries fall into $\mathcal{L}$. However, it is natural to consider the version of the problem where the involved OMQs fall into different languages. Unsurprisingly, if the left-hand side query is expressed in a UCQ-rewritable language (based on linear, non-recursive, or sticky sets of TGDs), we can use the algorithm that relies on the small witness property discussed above, which provides optimal upper bounds for almost all the considered cases (the only exception is the containment of sticky in non-recursive OMQs over schemas of unbounded arity). Things are more interesting if the ontology of the left-hand side query is expressed using guarded or frontier-guarded TGDs, while the ontology of the right-hand side query is not (frontier-)guarded. By using automata techniques, we show that containment of (frontier-)guarded in non-recursive OMQs is in 3ExpTime, while containment of (frontier-)guarded in sticky OMQs is in 2ExpTime. We establish matching lower bounds, even over schemas of fixed arity, by refining techniques from [55].

Related work Investigating the complexity of containment and equivalence of queries is a classical task in the study of database query languages. Due to its popularity, this summary of related work is far from being exhaustive.

It is a classical result that containment among CQs is NP-complete [54] and essentially amounts to answering CQs. The fact that containment of Datalog queries is undecidable is shown in [126]. Containment of Datalog queries in (U)CQs and non-recursive Datalog queries is studied in [55], where it is shown that deciding whether a Datalog query is contained in a UCQ (respectively, a non-recursive Datalog query) is 2ExpTime-complete (respectively, 3ExpTime-complete). Conversely, containment of UCQs in Datalog queries amounts to answering Datalog queries (as one can view CQs as databases) and is thus ExpTime-complete [53, 63]. Containment of non-recursive Datalog queries is studied in [29] and it is shown that this problem is coNExpTime-complete. The case for various description logics is investigated in [34, 35]. We also mention the works [20, 28, 89],

which study the satisfiability problem for guarded (negation) fixed point logics, as these logics allow to capture (frontier-)guarded Datalog. Finally, we would like to mention the paper [40], which studies containment of expressive query languages that extend guarded Datalog.

Outline We set up basic terminology for containment and its problem definition in Section 4.1. Containment for UCQ-rewritable languages is studied in Section 4.2, while containment for guarded-based classes is treated in Section 4.3. Finally, in Section 4.4 we study containment for the cases when the left-hand side and the right-hand side OMQ do not fall into the same class. Some lengthy proofs are deferred to Appendix B.2.

4.1 Containment: The Basics

Let Q_1 and Q_2 be two OMQs that have the same data schema $\mathbf{S}$. As said in the introductory part of this chapter, we say that Q_1 is *contained in* Q_2, written $Q_1 \subseteq Q_2$, if for every $\mathbf{S}$-database $\mathfrak{D}$ it holds that $Q_1(\mathfrak{D}) \subseteq Q_2(\mathfrak{D})$. Notice thus that $Q_1 \equiv Q_2$ iff $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. Hence, deciding equivalence of two OMQs amounts to two checks of containment, and this thus justifies that we put our focus on deciding containment among OMQs. Let us define the problem of containment among OMQs more formally.

Given two OMQ languages $\mathcal{L}_1$ and $\mathcal{L}_2$, we define the problem of containment for $\mathcal{L}_1$ and $\mathcal{L}_2$ as follows:

Problem:	$\mathsf{Cont}(\mathcal{L}_1, \mathcal{L}_2)$
Input:	Two OMQs $Q_1 \in \mathcal{L}_1$ and $Q_2 \in \mathcal{L}$ that have the same data schema $\mathbf{S}$.
Question:	Is it the case that $Q_1 \subseteq Q_2$?

Whenever $\mathcal{L}_1 = \mathcal{L}_2 =: \mathcal{L}$, then we write $\mathsf{Cont}(\mathcal{L})$ instead of $\mathsf{Cont}(\mathcal{L}, \mathcal{L})$.

In what follows, we establish some simple but fundamental results, which help to better understand the nature of our problem. We first investigate the relationship between evaluation and containment, which in turn allows us to obtain an initial boundary for the decidability of our problem. In particular, we observe that, for reasonable OMQ languages of our interest, one can only obtain a decidability result for containment if the evaluation problem for the involved OMQ languages is decidable. We then focus on the OMQ languages mentioned in the introductory part of this chapter and observe that, once we fix the class of TGDs, it does not make a difference whether we consider CQs or UCQs. In other words, we show that an OMQ in $(\mathcal{C}, \mathsf{UCQ})$, where $\mathcal{C} \in \{\mathsf{FG}, \mathsf{G}, \mathsf{L}, \mathsf{NR}, \mathsf{S}\}$, can be rewritten as an OMQ in $(\mathcal{C}, \mathsf{CQ})$. This fact simplifies our later complexity analysis since for establishing upper (respectively, lower) bounds it suffices to focus on CQs (respectively, UCQs).

4.1.1 Evaluation vs. Containment

As one might expect, evaluation and containment of OMQs are strongly related. In fact, as we explain below, the former can be easily reduced to the latter. But let us first

introduce some auxiliary notation. Consider a database $\mathfrak{D}$ and a tuple $\bar{c} = c_1, \ldots, c_n$ over $\mathsf{adom}(\mathfrak{D})$. We denote by $q_{\mathfrak{D},\bar{c}}(\bar{x})$, where $\bar{x} = x_{c_1}, \ldots, x_{c_n}$, the CQ obtained from the conjunction of facts occurring in $\mathfrak{D}$ after replacing each constant c with the variable x_c.

Consider now an OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $(\mathcal{C}, \mathsf{CQ})$, where $\mathcal{C}$ is some class of TGDs, an $\mathbf{S}$-database $\mathfrak{D}$, and a tuple $\bar{c}$ over $\mathsf{adom}(\mathfrak{D})$. It is not difficult to show that:

$$\bar{c} \in Q(\mathfrak{D}) \iff \underbrace{(\mathsf{sig}(\mathcal{O}), \varnothing, q_{\mathfrak{D},\bar{c}})}_{Q_1} \subseteq \underbrace{(\mathsf{sig}(\mathcal{O}), \mathcal{O}, q(\bar{x}))}_{Q_2}.$$

Let $\mathcal{L}_\varnothing$ be the OMQ language that consists of all OMQs of the form $(\mathbf{S}, \varnothing, q)$, i.e., the set of TGDs is empty, where q is a CQ. It is clear that $Q_1 \in \mathcal{L}_\varnothing$ and $Q_2 \in (\mathcal{C}, \mathsf{CQ})$. Therefore, for every OMQ language $\mathcal{L} = (\mathcal{C}, \mathsf{CQ})$, where $\mathcal{C}$ is a class of TGDs, we immediately obtain:

Proposition 4.1. $\mathsf{Eval}(\mathcal{L})$ *can be reduced in polynomial time to* $\mathsf{Cont}(\mathcal{L}_\varnothing, \mathcal{L})$.

We now show that the problem of answering BCQs under sets of TGDs is also reducible to the complement of containment. Consider now an OMQ $Q = (\mathbf{S}, \mathcal{O}, q)$ from $(\mathcal{C}, \mathsf{BCQ})$, where $\mathcal{C}$ is some class of TGDs and q a BCQ, and an $\mathbf{S}$-database $\mathfrak{D}$. It is easy to see that

$$\mathfrak{D} \models Q \iff \underbrace{(\mathbf{S}, \mathcal{O}^*_{\mathfrak{D}}, q^*)}_{Q_1} \not\subseteq \underbrace{(\mathbf{S}, \varnothing, \exists x\, G(x))}_{Q_2},$$

where $\mathcal{O}^*_{\mathfrak{D}}$ is obtained from $\mathcal{O}$ by renaming each predicate R in $\mathcal{O}$ into $R^* \notin \mathbf{S}$ and adding the set of TGDs

$$\{\top \to R^*(c_1, \ldots, c_k) \mid R(c_1, \ldots, c_k) \in \mathfrak{D}\}, \tag{4.1}$$

q^* is obtained from q by renaming each predicate R into $R^* \notin \mathbf{S}$, and the predicate $G/1$ does not occur in $\mathbf{S}$. Indeed, the above equivalence holds since $G \notin \mathbf{S}$ implies that $Q_2(\mathfrak{D}) = \varnothing$, for every $\mathbf{S}$-database $\mathfrak{D}$.

Assuming that $\mathcal{C}$ is closed under adding the TGDs (4.1), it holds that $Q_1 \in (\mathcal{C}, \mathsf{CQ})$, while $Q_2 \in \mathcal{L}_\varnothing$. We write $\overline{\mathsf{Cont}(\mathcal{L}_1, \mathcal{L}_2)}$ for the complement of $\mathsf{Cont}(\mathcal{L}_1, \mathcal{L}_2)$. Hence, for every OMQ language $\mathcal{L} = (\mathcal{C}, \mathsf{CQ})$, where $\mathcal{C}$ is a class of TGDs that is closed under adding TGDs of the form (4.1), the following holds:

Proposition 4.2. $\mathsf{Eval}(\mathcal{L})$ *can be reduced in polynomial time to* $\overline{\mathsf{Cont}(\mathcal{L}, \mathcal{L}_\varnothing)}$.

Remark 4.3. Actually, we only reduced $\mathsf{Eval}(\mathcal{C}, \mathsf{BCQ})$ to $\overline{\mathsf{Cont}(\mathcal{L}, \mathcal{L}_\varnothing)}$. However, it is well-known and easy to show that $\mathsf{Eval}(\mathcal{C}, \mathsf{CQ})$ can be polynomially reduced to $\mathsf{Eval}(\mathcal{C}, \mathsf{BCQ})$ for all classes of our interest (see, e.g., [44, 117]).

For those OMQ languages that contain $\mathcal{L}_\varnothing$ we obtain as a corollary of Propositions 4.1 and 4.2 an initial boundary for the decidability of containment. Namely, decidability of containment can only hold if at least one of the two involved languages has a decidable query evaluation problem. More formally:

Corollary 4.4. $\mathsf{Cont}(\mathcal{L}_1, \mathcal{L}_2)$ *is undecidable if* $\mathsf{Eval}(\mathcal{L}_1)$ *is undecidable or* $\mathsf{Eval}(\mathcal{L}_2)$ *is undecidable.*

Notice that the converse of Corollary 4.4 does not hold, since evaluation for Datalog queries is decidable, yet containment among Datalog queries is undecidable [126]. The undecidability of containment for Datalog queries immediately implies that containment for OMQ languages based on the weakly acyclic, weakly (frontier-)guarded, and weakly sticky is undecidable, as they all extend Datalog queries (cf. Section 3.4). Therefore, the focus of the next section is on decision procedures for containment for OMQ languages that are based on non-weak versions.

From UCQs to CQs Before we proceed to investigate containment problems for concrete OMQ languages, let us state the following useful result:

Proposition 4.5. *Given an OMQ $Q \in (\mathcal{C}, \mathsf{UCQ})$, where $\mathcal{C} \in \{\mathsf{FG}, \mathsf{G}, \mathsf{L}, \mathsf{NR}, \mathsf{S}\}$, we can construct in polynomial time an OMQ $Q' \in (\mathcal{C}, \mathsf{CQ})$ such that $Q \equiv Q'$.*

The proof of the above result can be found in Appendix B.2, and relies on the idea of encoding Boolean operations (in our case the 'or' operator) using a set of atoms – this idea has been used in several works (see, e.g., [29, 41, 83]). Proposition 4.5 allows us to focus on OMQs that are based on CQs. Assuming that $\mathcal{C}_1, \mathcal{C}_2 \in \{\mathsf{FG}, \mathsf{G}, \mathsf{L}, \mathsf{NR}, \mathsf{S}\}$ and that $\mathcal{L}$ is a complexity class that is closed under polynomial time reductions, then:

$$\mathsf{Cont}((\mathcal{C}_1, \mathsf{CQ}), (\mathcal{C}_2, \mathsf{CQ})) \text{ is } \mathcal{L}\text{-complete} \iff \mathsf{Cont}((\mathcal{C}_1, \mathsf{UCQ}), (\mathcal{C}_2, \mathsf{UCQ})) \text{ is } \mathcal{L}\text{-complete.}$$

We will employ this result in particular in Section 4.2 as it simplifies the presentation of our results (for Section 4.3, the case where we have UCQs as queries is no more complicated than the case of CQs).

4.2 Containment for UCQ-Rewritable Classes

We now focus on OMQ languages that enjoy the crucial property of *UCQ-rewritability*. In Section 3.4, we already defined the notion of first-order rewritable sets of TGDs. The notion of UCQ-rewritability for OMQs lifts this definition to the level of OMQs and demands the existence of a rewriting that is actually a UCQ. The formal definition is as follows:

Definition 4.6. We call an OMQ language $(\mathcal{C}, \mathsf{CQ})$ *UCQ-rewritable* if, for each OMQ $Q \in (\mathcal{C}, \mathsf{CQ})$, we can effectively construct a UCQ $q(\bar{x})$ such that $Q \equiv q$. We call q a *UCQ-rewriting* of Q.

We proceed to establish our desired small witness property based on for UCQ-rewritable languages. By the definition of UCQ-rewritability, for each language $\mathcal{L}$ that is UCQ-rewritable, there exists a computable function $f_{\mathcal{L}}$ from $\mathcal{L}$ to the natural numbers such that the following holds: for every OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $\mathcal{L}$, and UCQ-rewriting $q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$ of Q, it is the case that $\max_{1 \leq i \leq n}\{|q_i|\} \leq f_{\mathcal{L}}(Q)$. Then:

Proposition 4.7. *Consider a UCQ-rewritable language $\mathcal{L}$, and two OMQs $Q \in \mathcal{L}$ and $Q' \in (\mathsf{TGD}, \mathsf{CQ})$, both with data schema $\mathbf{S}$. If $Q \not\subseteq Q'$, then there exists an $\mathbf{S}$-database $\mathfrak{D}$ such that* (i) *$Q(\mathfrak{D}) \not\subseteq Q'(\mathfrak{D})$, and* (ii) *the number of atoms of $\mathfrak{D}$ is bounded by $f_{\mathcal{L}}(Q)$.*

Proof sketch. Assume that $q(\bar{x}) = \bigvee_{i=1}^{n} q_i(\bar{x})$ is a UCQ-rewriting of Q, and let $\bar{x} = x_1, \ldots, x_n$. Since, by hypothesis, $Q \not\subseteq Q'$, we conclude that $q \not\subseteq Q'$, which in turn implies that there exists an $i \in [n]$ such that $q_i \not\subseteq Q'$. It is easy to show that $([x_1]_{q_i}, \ldots, [x_n]_{q_i}) \notin Q'(\mathfrak{D}_{q_i})$, where $\mathfrak{D}$ is the structure corresponding to q_i (viewed as a database). Since $([x_1]_{q_i}, \ldots, [x_n]_{q_i}) \in q_i(\mathfrak{D}_{q_i})$, we get that $([x_1]_{q_i}, \ldots, [x_n]_{q_i}) \in Q(\mathfrak{D}_{q_i})$. Therefore, $Q(\mathfrak{D}_{q_i}) \not\subseteq Q'(\mathfrak{D}_{q_i})$, and the claim follows since the number of atoms of $\mathfrak{D}_{q_i}$ is clearly bounded by $f_{\mathcal{L}}(Q)$. □

In Proposition 4.7 we only assume that the left-hand side query falls into a UCQ-rewritable language, without any assumption on the language of the right-hand side query. Thus, we immediately get a decision procedure for $\mathsf{Cont}(\mathcal{L}_1, \mathcal{L}_2)$ if $\mathcal{L}_1$ is UCQ-rewritable and $\mathsf{Eval}(\mathcal{L}_2)$ is decidable. Given $Q_1 \in \mathcal{L}_1$ and $Q_2 \in \mathcal{L}_2$, both with data schema $\mathbf{S}$, we perform the following steps:

(i) Guess an $\mathbf{S}$-database $\mathfrak{D}$ that hast at most $f_{\mathcal{L}_1}(Q_1)$ atoms, and a tuple $\bar{c}$ over $\mathsf{adom}(\mathfrak{D})$.

(ii) Verify that $\bar{c} \in Q_1(\mathfrak{D})$ and $\bar{c} \notin Q_2(\mathfrak{D})$.

We immediately get:

Theorem 4.8. *$\mathsf{Cont}(\mathcal{L}_1, \mathcal{L}_2)$ is decidable provided $\mathcal{L}_1$ is UCQ-rewritable and $\mathsf{Eval}(\mathcal{L}_2)$ is decidable.*

This generic result shows that $\mathsf{Cont}(\mathcal{C}, \mathsf{CQ})$ is decidable for every class $\mathcal{C} \in \{\mathsf{L}, \mathsf{NR}, \mathsf{S}\}$, but it does not say anything on the precise complexity. This will be the subject for the remainder of this section.

4.2.1 Linearity

The problem of computing UCQ-rewritings for OMQs in $(\mathsf{L}, \mathsf{CQ})$ has been studied in [82], where a resolution-based procedure, called XRewrite, has been proposed – we provide a short description of XRewrite in Appendix A. This rewriting algorithm takes as an input a query $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $(\mathsf{L}, \mathsf{CQ})$ and constructs a UCQ-rewriting $q'(\bar{x})$ over $\mathbf{S}$ by starting from $q(\bar{x})$ and exhaustively applying rewriting steps to the rules of $\mathcal{O}$ using resolution. Due to the fact that the set of TGDs is linear i.e., their bodies consist of single atoms only, it is not possible to obtain a CQ that has more atoms than the original one during the execution of XRewrite. Therefore:

Proposition 4.9. *The function $f_{(\mathsf{L},\mathsf{CQ})}$ satisfies, for any $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $(\mathsf{L}, \mathsf{CQ})$, $f_{(\mathsf{L},\mathsf{CQ})}(Q) \leq |q|$.*

Having the above result in place, it can be shown that the algorithm underlying Theorem 4.8 guesses a polynomially-sized witness for non-containment, and then calls

a $\mathcal{C}$-oracle for solving query evaluation under linear OMQs, where $\mathcal{C}$ is PSPACE in general, and NP if the arity is fixed. These complexity classes are obtained from the fact that $\mathsf{Eval}(\mathsf{L}, \mathsf{CQ})$ is PSPACE-complete (cf. Subsection 3.4.2). Therefore, $\overline{\mathsf{Cont}(\mathsf{L}, \mathsf{CQ})}$ is in PSPACE in general, and in Σ_2^{P} in case of fixed arity. Regarding the lower bounds, Proposition 4.1 allows us to inherit the PSPACE-hardness of $\mathsf{Eval}(\mathsf{L}, \mathsf{CQ})$. Unfortunately, in the case of fixed arity, we can only obtain NP-hardness, while Proposition 4.2 allows us to obtain coNP-hardness. Nevertheless, it is implicit in [35, Theorem 9], where the containment problem for OMQ languages based on description logics is considered, that $\mathsf{Cont}(\mathsf{L}, \mathsf{CQ})$ is Π_2^{P}-hard, even for TGDs of the form $B(x) \to H(x)$.

THEOREM 4.10. $\mathsf{Cont}(\mathsf{L}, \mathsf{CQ})$ *is* PSPACE*-complete, and* Π_2^{P}*-complete if the width of the schema is fixed.*

4.2.2 Non-Recursiveness

Although the OMQ language $(\mathsf{NR}, \mathsf{CQ})$ is not explicitly considered in [82], where the algorithm XRewrite is defined, the same algorithm can deal with $(\mathsf{NR}, \mathsf{CQ})$. By analyzing the UCQ-rewritings constructed by XRewrite, whenever the input query falls into $(\mathsf{NR}, \mathsf{CQ})$, we can establish the following result:

PROPOSITION 4.11. *For any* $Q = (\mathbf{S}, \mathcal{O}, q)$ *from* $(\mathsf{NR}, \mathsf{CQ})$ *it holds that*

$$f_{(\mathsf{NR},\mathsf{CQ})}(Q) \leq |q| \cdot n^{p(|\mathsf{sig}(\mathcal{O})|)},$$

where $n \coloneqq \max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\}$*, and* $p(\cdot)$ *is a polynomial depending on* Q*.*

Proposition 4.11 implies that non-containment for queries that fall in $(\mathsf{NR}, \mathsf{CQ})$ is witnessed via a database of at most exponential size. We show next that this bound is optimal:

PROPOSITION 4.12. *There are classes of OMQs from* $(\mathsf{NR}, \mathsf{CQ})$*,*

$$\{Q_1^n \coloneqq (\mathbf{S}, \mathcal{O}_1^n, q_1)\}_{n \geq 1} \quad \textit{and} \quad \{Q_2^n \coloneqq (\mathbf{S}, \mathcal{O}_2^n, q_2)\}_{n \geq 1},$$

where $|\mathsf{sig}(\mathcal{O}_1^n)| = |\mathsf{sig}(\mathcal{O}_2^n)| = n+2$*, such that, for every* $\mathbf{S}$*-database* $\mathfrak{D}$*, if* $Q_1^n(\mathfrak{D}) \not\subseteq Q_2^n(\mathfrak{D})$*, then* $\mathfrak{D}$ *has at least* 2^{n-1} *atoms.*

PROOF SKETCH. Let $\mathbf{S} = \{S/2\}$. $\mathcal{O}_1^n$ consists of

$$\begin{aligned} S(x,y) &\to P_1(x,y), \\ P_i(x,y), P_i(y,z) &\to P_{i+1}(x,z), \quad 1 \leq i \leq n-1, \\ P_n(x,y) &\to \mathsf{Ans}(x,y), \end{aligned}$$

while $q_1 \coloneqq \exists x, y\, \mathsf{Ans}(x,y)$.

The set $\mathcal{O}_2^n$ consists of the TGDs

$$\begin{aligned} S(x,y) &\to R_1(x,y), \\ R_i(x,y) &\to R_{i+1}(x,y), \quad 1 \leq i \leq n-1, \\ R_i(x,y), R_i(y,z) &\to R_{i+1}(x,z), \quad 1 \leq i \leq n-1, \\ R_i(x,x) &\to \mathsf{Ans}(x,x), \quad 1 \leq i \leq n, \end{aligned}$$

while $q_2 \coloneqq \exists x\, \mathsf{Ans}(x,x)$. This completes our construction.

We show that $\{Q_1^n\}_{n\geq 1}$ and $\{Q_2^n\}_{n\geq 1}$ are the desired classes of OMQs belonging to $(\mathsf{NR},\mathsf{CQ})$. Consider an arbitrary $\mathbf{S}$-database $\mathfrak{D}$ and let $\mathcal{G} = (V,E)$ be the directed graph with $V \coloneqq \mathsf{adom}(\mathfrak{D})$ and $E \coloneqq S^{\mathfrak{D}}$. It can be shown that $\mathsf{chase}(\mathfrak{D},\mathcal{O}_1^n) \models P_i(c,d)$ implies that $\mathcal{G}$ contains a path of length 2^{i-1} from c to d. Moreover, if $R_i(c,d)$ appears in $\mathsf{chase}(\mathfrak{D},\mathcal{O}_2^n)$, then $\mathcal{G}$ contains a path of length *at most* 2^{i-1} from c to d.

Observe that if $\mathcal{G}$ contains a cycle of length at most 2^{n-1}, then an atom of the form $\mathsf{Ans}(c,c)$ occurs in $\mathsf{chase}(\mathfrak{D},\mathcal{O}_2^n)$, which in turn implies that $\mathfrak{D} \models Q_2^n$, and thus, $Q_1^n(\mathfrak{D}) \subseteq Q_2^n(\mathfrak{D})$. We conclude that $Q_1^n(\mathfrak{D}) \not\subseteq Q_2^n(\mathfrak{D})$ implies that $\mathcal{G}$ does not contain a cycle of length at most 2^{n-1}. In this case, it can be shown that $\mathfrak{D} \models Q_1^n$ implies that $\mathfrak{D}$ contains facts

$$S(c_1,c_2), S(c_2,c_3), \ldots, S(c_{m-2},c_{m-1}), S(c_{m-1},c_m),$$

where $m = 2^{n-1}+1$, which proves the claim. □

Let us now focus on the complexity of $\mathsf{Cont}(\mathsf{NR},\mathsf{CQ})$. By naively combining the algorithm underlying Theorem 4.8 and the exponential bound provided by Proposition 4.11, we get that $\overline{\mathsf{Cont}(\mathsf{NR},\mathsf{CQ})}$ is feasible in non-deterministic exponential time with access to a NExpTime-oracle – the oracle is needed for solving $\mathsf{Eval}(\mathsf{NR},\mathsf{CQ})$. This rough upper bound can be significantly improved. In fact, it can be decreased to $\text{NExpTime}^{\text{NP}}$, which is nearly optimal (more details are given below), by employing a refined version of the algorithm underlying Theorem 4.8. Recall that $\text{NExpTime}^{\text{NP}}$ forms the second level of the exponential hierarchy, and it collects all the decision problems that can be solved via an alternating exponential time algorithm with two alternations starting from an existential state, that is, it can perform a series of existential steps followed by a series of universal steps. The refined version of the algorithm underlying Theorem 4.8 is such an algorithm.

Before giving this algorithm, let us recall a crucial property of non-recursive OMQs which is implicit in [103]. Given an $\mathbf{S}$-database $\mathfrak{D}$, an OMQ $Q = (\mathbf{S},\mathcal{O},q(\bar{x}))$ from $(\mathsf{NR},\mathsf{CQ})$, and a tuple $\bar{c}$ over $\mathsf{adom}(\mathfrak{D})$, if $\bar{c} \in Q(\mathfrak{D})$, then there exists a finite[2] chase sequence

$$\mathfrak{D} = \mathfrak{J}_0,\ \mathfrak{J}_1,\ \mathfrak{J}_2,\ \ldots,\ \mathfrak{J}_{n-1},\ \mathfrak{J}_{n(\mathfrak{D},\mathcal{O})},$$

[2]Recall that by definition, all chase sequences are infinite. However, we agreed that by a finite chase sequence we mean one that repeats the same structure indefinitely at some point (cf. Section 3.3).

for $\mathfrak{D}$ and $\mathcal{O}$, where

$$n(\mathfrak{D}, \mathcal{O}) := |\mathfrak{D}| \cdot (\max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\})^{|\mathsf{sig}(\mathcal{O})|}$$

such that $\bar{c} \in q(\mathfrak{J}_{n(\mathfrak{D},\mathcal{O})})$. Having this property in place, we can now present our alternating algorithm. Given $Q_1 = (\mathbf{S}, \mathcal{O}_1, q_1(\bar{x}))$ and $Q_2 = (\mathbf{S}, \mathcal{O}_2, q_2(\bar{x}))$ from $(\mathsf{NR}, \mathsf{CQ})$, our algorithm proceeds as follows:

(i) Guess an $\mathbf{S}$-database $\mathfrak{D}$ with at most $f_{(\mathsf{NR},\mathsf{CQ})}(Q_1)$ atoms, and a tuple $\bar{c} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$.

(ii) Guess a chase sequence

$$\mathfrak{D} = \mathfrak{J}_0,\ \mathfrak{J}_1,\ \mathfrak{J}_2,\ \ldots,\ \mathfrak{J}_{n-1},\ \mathfrak{J}_{n(\mathfrak{D},\mathcal{O}_1)}$$

for $\mathfrak{D}$ and $\mathcal{O}_1$.

(iii) Guess a mapping h, which is the identity on $\mathsf{adom}(\mathfrak{D})$, from the variables in q_1 to $\mathsf{adom}(\mathfrak{J}_{n(\mathfrak{D},\mathcal{O}_1)})$.

(iv) If h is a homomorphism from q_1 to $\mathfrak{J}_{n(\mathfrak{D},\mathcal{O}_1)}$ such that $h(\bar{x}) = \bar{c}$, then proceed; otherwise, REJECT.

(v) Universally select each chase sequence

$$\mathfrak{D} = \mathfrak{J}_0,\ \mathfrak{J}_1,\ \mathfrak{J}_2,\ \ldots,\ \mathfrak{J}_{n-1},\ \mathfrak{J}_{n(\mathfrak{D},\mathcal{O}_2)}$$

for $\mathfrak{D}$ and $\mathcal{O}_2$.

(vi) Universally select each mapping h, which is the identity on $\mathsf{adom}(\mathfrak{D})$, from the variables in q_2 to $\mathsf{adom}(\mathfrak{J}_{n(\mathfrak{D},\mathcal{O}_2)})$.

(vii) If h is a homomorphism from q_2 to $\mathfrak{J}_{n(\mathfrak{D},\mathcal{O}_2)}$ such that $h(\bar{x}) = \bar{c}$, then REJECT; otherwise, ACCEPT.

The above algorithm is an alternating exponential time algorithm with two alternations that starts from an existential state. Moreover, it accepts iff $Q_1 \not\subseteq Q_2$, and the desired upper bound follows.

It is not known whether our problem is coNEXPTIME$^{\text{NP}}$-complete. Nevertheless, we provide a nearly matching lower bound. In fact, PTIME$^{\text{NEXPTIME}}$-hardness. More details on how the above complexity classes are related are discussed below. Let us now explain how PTIME$^{\text{NEXPTIME}}$-hardness is obtained – a detailed proof is given in Appendix B.2. We exploit a tiling problem that has been recently introduced in [72]. Roughly speaking, an instance of this tiling problem is a triple $(m, \mathbb{T}_1, \mathbb{T}_2)$, where m is a natural number in unary representation, and $\mathbb{T}_1, \mathbb{T}_2$ are standard tiling problems for the $(2^n \times 2^n)$-grid. The question is whether, for every initial condition w of length m, $\mathbb{T}_1$ has no solution with w or $\mathbb{T}_2$ has some solution with w. The initial condition w simply fixes the first m tiles of the first row of the grid. We construct in polynomial time two OMQs $Q_1, Q_2 \in (\mathsf{NR}, \mathsf{CQ})$ such that $(m, \mathbb{T}_1, \mathbb{T}_2)$ has a solution iff $Q_1 \subseteq Q_2$. The idea is to force every input database to store an initial condition w of length m, and then encode the problem whether $\mathbb{T}_i$ has a solution with w into Q_i, for each $i \in \{1, 2\}$. Then:

Theorem 4.13. $\mathsf{Cont}(\mathsf{NR}, \mathsf{CQ})$ *is in* $\text{coNExpTime}^{\text{NP}}$*, and* $\text{PTime}^{\text{NExpTime}}$*-hard. The lower bound holds even if the width of the schema is fixed.*

Let us now briefly comment on $\text{NExpTime}^{\text{NP}}$ versus the class $\text{PTime}^{\text{NExpTime}}$. It is known that $\text{NExpTime}^{\text{NP}}$ is a delicate class: if we restrict its oracle access "too much," it collapses to $\text{PTime}^{\text{NExpTime}}$ [92]. For example, following the notation of [92], $\text{PTime}^{\text{NExpTime}}$ coincides with $\text{NExpTime}^{\text{NP}[poly]_{tree}}$, where only polynomially many oracle calls are allowed throughout the computation tree of the Turing machine. Also, $\text{PTime}^{\text{NExpTime}}$ coincides with $\text{NExpTime}^{\text{NP}[poly]_{path}[exp]_{yes,tree}}$, where only polynomially many oracle calls are allowed on each path of the computation tree, and exponentially many calls with a Yes answer throughout the computation tree of the Turing machine. These results support our claim that $\text{PTime}^{\text{NExpTime}}$ is a nearly matching lower bound for the problem $\mathsf{Cont}(\mathsf{NR}, \mathsf{CQ})$.

4.2.3 Stickiness

We now focus on OMQs that fall into $(\mathsf{S}, \mathsf{CQ})$. As shown in [82], given an OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $(\mathsf{S}, \mathsf{CQ})$, there exists an execution of $\mathsf{XRewrite}$ that constructs a UCQ-rewriting $q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$ over $\mathbf{S}$ with the following property: for each $i \in [n]$, if a variable v occurs in q_i in more than one atom, then v already occurs in q. This property is used in [82] to bound the number of atoms that can appear in a single CQ q_i. Let us write $T(q)$ for the set of terms (constants and variables) occurring in q, and $C(\mathcal{O})$ for the set of constants occurring in $\mathcal{O}$. The following result is established in [82] – recall that $\mathsf{wd}(\mathbf{S})$ denotes width of $\mathbf{S}$:

Proposition 4.14. *For any* $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ *from* $(\mathsf{S}, \mathsf{CQ})$ *it holds that*

$$f_{(\mathsf{S},\mathsf{CQ})}(Q) \leq |\mathbf{S}| \cdot (|T(q)| + |C(\mathcal{O})| + 1)^{\mathsf{wd}(\mathbf{S})}\,.$$

We now study the complexity of $\mathsf{Cont}(\mathsf{S}, \mathsf{CQ})$. Let us first look at schemas of unbounded width. Proposition 4.14 implies that the algorithm underlying Theorem 4.8 runs in exponential time assuming access to a $\mathcal{C}$-oracle, where $\mathcal{C}$ is a complexity class powerful enough to solve $\mathsf{Eval}(\mathsf{S}, \mathsf{CQ})$ and its complement. But, since $\mathsf{Eval}(\mathsf{S}, \mathsf{CQ})$ is in ExpTime (see Subsection 3.4.2), both $\mathsf{Eval}(\mathsf{S}, \mathsf{CQ})$ and its complement are feasible in NExpTime, and thus the oracle call is not really needed. From this discussion, we conclude that $\overline{\mathsf{Cont}(\mathsf{C}, \mathsf{CQ})}$ is in NExpTime. A matching lower bound is obtained by a reduction from the standard tiling problem for the $(2^n \times 2^n)$-grid – details are again given in Appendix B.2. In fact, the same lower bound is established in [33]. However, our result is stronger as it shows that the problem remains hard even if the right-hand side query uses linear TGDs of a simple form – this is also discussed in Section 4.4, where containment of queries that fall into different OMQ languages is studied.

Regarding schemas of bounded width, Proposition 4.14 provides a witness for non-containment of polynomial size, which implies that the algorithm underlying Theorem 4.8 runs in polynomial time with access to an NP-oracle. Therefore, $\overline{\mathsf{Cont}(\mathsf{S}, \mathsf{CQ})}$ is in Σ_2^{P}, while a matching lower bound is implicit in [35]. Summarizing, we obtain:

THEOREM 4.15. $\mathsf{Cont}(\mathsf{S}, \mathsf{CQ})$ *is* coNEXPTIME*-complete, and hardness holds even if the sets of TGDs use only two constants. In the case of schemas of bounded width, it is* Π_2^{P}*-complete even for constant-free TGDs.*

4.3 Containment for Guarded-Based Classes

The languages from the previous section shared the property of UCQ-rewritability. Unfortunately, OMQs whose ontology is formulated using guarded or frontier-guarded rules do not have this property (see Chapter 5 for more details). We thus have to employ different techniques in order to solve the containment problem for these classes. This is the goal of the present section.

Recall that we denote by G the class of all finite guarded sets of TGDs, and by FG the class of all finite frontier-guarded sets of TGDs. The main goal of this section is to establish the following theorem:

THEOREM 4.16. (i) *For each* $\mathcal{L} \in \{\mathsf{G}, \mathsf{FG}\}$ *and each* $\mathcal{Q} \in \{\mathsf{UCQ}, \mathsf{CQ}, \mathsf{AQ}\}$*, the problem* $\mathsf{Cont}(\mathcal{L}, \mathcal{Q})$ *is* 2EXPTIME*-complete. Hardness for* 2EXPTIME *holds even for the problem* $\mathsf{Cont}(\mathsf{G}, \mathsf{BCQ})$ *when we only use unary and binary relation symbols.*

(ii) *For schemas of bounded width, the problem* $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ})$ *is* EXPTIME*-complete.*

Lower bounds The 2EXPTIME lower bound in item (i) for the case $\mathcal{L} \in \{\mathsf{G}, \mathsf{FG}\}$ and $\mathcal{Q} \in \{\mathsf{UCQ}, \mathsf{CQ}\}$ is inherited from results on the description logic $\mathcal{ELI}$ obtained in [34], where it is shown that containment among OMQs from $(\mathcal{ELI}, \mathsf{BCQ})$ is 2EXPTIME-hard. Since every OMQ from $(\mathcal{ELI}, \mathsf{BCQ})$ can be rewritten (in polynomial time) into an equivalent OMQ from $(\mathsf{G}, \mathsf{BCQ})$, the 2EXPTIME lower bound immediately follows.[3] Likewise, the EXPTIME-hardness in item (i) is obtained from the fact that $\mathsf{Cont}(\mathcal{EL}, \mathsf{BAQ})$ is EXPTIME-hard, as shown in [34]. In [23] it is shown that deciding containment of a (Boolean) guarded Datalog query into a Boolean *acyclic* CQ (see below) is hard for 2EXPTIME. As we shall see in Subsection 4.3.2, Boolean acyclic CQs can easily be rewritten (in polynomial time) to OMQs that fall into $(\mathsf{G}, \mathsf{BAQ})$. Moreover, guarded Datalog queries correspond to OMQs from $(\mathsf{G}, \mathsf{AQ})$ that have no existential quantifiers in rule heads. Hence, 2EXPTIME-hardness of $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ})$ follows.

Upper bounds We thus mostly focus in this section on obtaining the upper bounds named in Theorem 4.16. Toward establishing these upper bounds, we will first prove the following result:

THEOREM 4.17. *There is a* 2EXPTIME *algorithm for the problem* $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$*, and the second exponent of its runtime only depends on the maximum arity of the relation symbols used in any of the two input OMQs. Hence,* $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$ *is in* EXPTIME *when we assume schemas of bounded width.*

[3] Recall that answering OMQs from $(\mathsf{G}, \mathsf{BCQ})$ with schemas of bounded width is feasible in EXPTIME [44], and thus $\mathsf{Eval}(\mathcal{ELI}, \mathsf{BCQ})$ is in EXPTIME. Thus, containment is in general more difficult than evaluation for guarded-based OMQs with schemas of bounded width.

In order to show that $\mathsf{Cont}(\mathsf{FG}, \mathsf{UCQ})$ is in 2ExpTime, we are going to facilitate Theorem 4.17 as follows:

- By employing a simple reduction, we show that we can reduce $\mathsf{Cont}(\mathsf{FG}, \mathsf{UCQ})$ to $\mathsf{Cont}(\mathsf{FG}, \mathsf{UBCQ})$, that is, we can focus of unions of Boolean CQs.
- Since Boolean UCQs correspond to frontier-guarded rules (with an empty sequence of frontier-variables), we can in turn reduce $\mathsf{Cont}(\mathsf{FG}, \mathsf{UBCQ})$ to $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$.
- We provide an exponential reduction from $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$ to $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$. Although this reduction is exponential, it only increases the arity of the relation symbols used polynomially. We show that this reduction thus still shows that $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$ is solvable in 2ExpTime. The reduction is based on the notion of *treeification* [18, 20], and is inspired by a similar reduction given in [20], where it is shown how to reduce the satisfiability problem for sentences belonging to guarded negation fixed point logic (GNFP) to the satisfiability problem for sentences belonging to guarded fixed point logic (GFP).

For obtaining the ExpTime upper bound in item (i) of Theorem 4.16 we simply reduce $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ})$ to the problem $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$ in polynomial time.

Constants and normal form assumption Let us emphasize that in the following, we assume that all rules are constant-free, as this simplifies the presentation of the material in this section. We will comment at the end of the section on how to extend the results to capture also the case where constants are allowed. Notice that, since we use constants in the proof of Proposition 4.5, we prove the upper bound in item (i) of Theorem 4.16 for the case of UCQs.

Apart from constant-free rules, let us also assume that all rules contain only a single atom in their heads. It is easy to transform (frontier-)guarded rules into such a normal form. In fact, given the rule

$$\tau \colon q(\bar{x}) \to \exists \bar{y} \, (\beta_1 \wedge \cdots \wedge \beta_k),$$

where $\beta_1 \wedge \cdots \wedge \beta_k$ is a conjunction of relational atoms, we pick a fresh relation symbol R_τ of arity $|\bar{x}| + |\bar{y}|$ and rewrite τ into the rules

$$q(\bar{x}) \to \exists \bar{y} \, R_\tau(\bar{x}, \bar{y}),$$
$$R_\tau(\bar{x}, \bar{y}) \to \beta_i, \quad 1 \leq i \leq k.$$

Notice that the resulting rules are still (frontier-)guarded. Doing this exhaustively for all rules enables us to rewrite any OMQ based on guarded or frontier-guarded TGDs into an equivalent one whose every rule has only one atom in its head. Moreover, it is easy to see that this transformation can be performed in polynomial time. Therefore, it is safe to assume sets of rules that are given in this normal form for the remainder of this section.

4.3.1 The Case of Atomic Queries

The goal of this part is to provide a proof of Theorem 4.17. To this end, we proceed as follows:

(i) We first show that, in order to decide containment, it suffices to restrict ourselves to input databases which are "tree-like" – more precisely, *acyclic* [135] in the standard database sense.

(ii) Following [28, 55, 87], we encode acyclic databases into trees that carry labels over a finite alphabet in order to make them amenable to the tree automata techniques presented in Section 2.4.

(iii) We devise 2APTA in order to reduce the question of containment to that of language emptiness for 2APTA.

Guardedness Revisited

Toward a proof of Theorem 4.17 – and in particular toward a proof of the tree witness property – we need several additional technical notions first.

Firstly, recall that a *tree decomposition* of a structure $\mathfrak{A}$ is a tuple $\delta = (\mathcal{T}, (X)_{v \in T})$ such that

(i) $\mathsf{dom}(\mathfrak{A}) \subseteq \bigcup_{v \in T} X_v$,

(ii) for every fact $R(a_1, \ldots, a_n)$ of $\mathfrak{A}$, there is a $v \in T$ with $\{a_1, \ldots, a_n\} \subseteq X_v$, and

(iii) for every $a \in \mathsf{dom}(\mathfrak{A})$, the set $\{v \mid a \in X_v\}$ induces a connected subtree of $\mathcal{T}$.

The *width* of δ is $\mathsf{wd}(\delta) \coloneqq \max\{|X_v| : v \in T\} - 1$, and the *tree-width* of $\mathfrak{A}$ is

$$\mathsf{tw}(\mathfrak{A}) \coloneqq \min\{\mathsf{wd}(\delta) \mid \delta \text{ is a tree decomposition of } \mathfrak{A}\}.$$

Notation. Given a tree decomposition $\delta = (\mathcal{T}, (X_v)_{v \in T})$ of $\mathfrak{A}$, we write $\mathfrak{A}_\delta(v)$ for the structure $\mathfrak{A} \restriction X_v$, and we omit the subscript "δ" whenever clear from context.

Acyclicity As mentioned in the introductory part of this section, one of the main goals here is to make the problem $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ})$ amenable to tree automata techniques. To this end, we need a notion that captures "tree-likeliness" for structures that have relation symbols of arbitrary arity. A first candidate for this notion is to consider structures of bounded tree-width as "tree-like." However, as we shall see, for the case of guardedness we can even impose a stronger concept of tree-likeliness – namely the well-known notion of *acyclicity* [135] from database theory. Let us make this notion more precise.

Let $\mathfrak{A}$ be an $\mathbf{S}$-structure over $\mathbf{S}$. We say that $X \subseteq \mathsf{dom}(\mathfrak{A})$ is *guarded in* $\mathfrak{A}$, if either $|X| = 1$ or there are $a_1, \ldots, a_n \in \mathsf{dom}(\mathfrak{A})$ such that

- $X \subseteq \{a_1, \ldots, a_n\}$ and
- there is an $R/n \in \mathbf{S}$ such that $\mathfrak{A} \models R(a_1, \ldots, a_n)$.

A tuple $\bar{a}$ is *guarded in* $\mathfrak{A}$ if the set $[\bar{a}]$ containing the elements of $\bar{a}$ is guarded in $\mathfrak{A}$.

Definition 4.18. Let $\delta = (\mathcal{T}, (X_v)_{v \in T})$ of $\mathfrak{A}$ be a tree decomposition of $\mathfrak{A}$. We say that $v \in T$ is *guarded* if there is a guarded set X in $\mathfrak{A}$ such that $X \supseteq X_v$. We call δ *guarded* if every $v \in T$ is guarded. We call $\mathfrak{A}$ *acyclic* if it has a guarded tree decomposition.

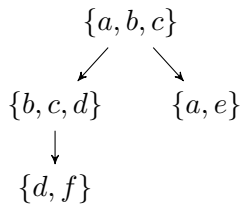

Figure 4.1: A guarded tree decomposition of $\mathfrak{D}$ from Example 4.19 showing that $\mathfrak{D}$ is acyclic.

Notice thus that the tree-width of an acyclic $\mathbf{S}$-structure is always bounded by the number $\max\{0, \mathsf{wd}(\mathbf{S}) - 1\}$.

Example 4.19. Let $\mathfrak{D} = \{R(a,b), T(a,b,c), R(b,c), R(c,a), T(b,c,d), R(a,e), R(d,f)\}$. It is not hard to check that $\mathfrak{D}$ is acyclic – a guarded tree decomposition of $\mathfrak{D}$ is shown in Figure 4.1. ⊣

The following lemma states that chasing an acyclic structure gives an acyclic result:

Lemma 4.20. *If an $\mathbf{S}$-structure $\mathfrak{A}$ is acyclic and $\mathcal{O}$ is a set of frontier-guarded TGDs, then $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ is acyclic as well. Moreover, for ever $k \geq 0$, $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ is acyclic.*

Proof. Let $\delta = (\mathcal{T}, (X_v)_{v \in T})$ be a guarded tree decomposition of $\mathfrak{A}$. Let $\pi = \mathfrak{J}_0, \mathfrak{J}_1, \ldots$ be an appropriate chase sequence for $\mathfrak{A}$ and $\mathcal{O}$, and recall that $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O}) = \bigcup_{i=0}^{k} \mathfrak{J}_k$. We successively augment δ by additional nodes according to the construction of π. More formally, we prove by induction on k that there is a guarded tree decomposition $\delta_k = (\mathcal{T}_k, (X_{k,v})_{v \in T_k})$ of $\bigcup_{i=0}^{k} \mathfrak{J}_i = \mathfrak{J}_k$. For $k = 0$, we just take $\delta_k \coloneqq \delta$.

Suppose now that $k \geq 1$. Let $\alpha(\bar{t}, \bar{\lambda}')$ be the atom derived in the k-th chase step, where $\bar{t}$ are elements contained in $\mathsf{dom}(\mathfrak{J}_{k-1})$ and the elements $\bar{\lambda}$ are fresh labeled nulls. Suppose this atom has been derived by an application of (τ, h), where $\tau \in \mathcal{O}$ is frontier-guarded. Hence, h maps the body of τ to $\mathfrak{J}_{k-1}$. Since τ is frontier-guarded, there is a frontier-guard $\beta(x_1, \ldots, x_n)$ in its body. Thus, the set $X \coloneqq \{h(x_1), \ldots, h(x_n)\} \supseteq [\bar{t}]$ is guarded in $\mathfrak{J}_{k-1}$. Let $v \in T_{k-1}$ be such that $X \subseteq X_{k-1,v}$. We construct δ_k as follows. If the sequence $\bar{\lambda}$ is not empty, then δ_k arises from δ_{k-1} by adding a new node v' to δ_{k-1} whose bag equals $[\bar{t}] \cup [\bar{\lambda}]$. Otherwise, if $\bar{\lambda}$ is empty, then $\delta_k \coloneqq \delta_{k-1}$. It is clear that δ_k is guarded, since the set $[\bar{t}] \cup [\bar{\lambda}]$ is guarded in $\mathfrak{J}_k$. Moreover, δ_k is clearly connected as $[\bar{t}] \subseteq X = X_{k-1,v}$. This completes the induction step.

This shows that $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ is acyclic for every $k \geq 0$. The fact that $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ is acyclic is witnessed by the tree decomposition $\delta^* = (\mathcal{T}^*, (X^*_v)_{v \in T^*})$, where $\mathcal{T}^*$ is the union of all $\mathcal{T}_k$, and $X^*_v = X_{k_0,v}$, where k_0 is the minimum k such that $v \in T_k$. It is not hard to see that δ^* is well-defined, since every δ_{k+1} extends δ_k in the sense that it only adds new nodes to δ_k. Clearly, δ^* is a guarded tree decomposition of $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$. □

(Strictly) acyclic CQs The notion of acyclicity can be easily extended to CQs and UCQs. Let $q(x_1, \ldots, x_n)$ be a conjunctive query over $\mathbf{S}$. Recall from Subsection 2.3.2 that

$\mathfrak{A}_q$ denotes the $\mathbf{S}$-structure corresponding to $q(x_1, \ldots, x_n)$, and recall that the domain of $\mathfrak{A}_q$ consists of the $\sim$-equivalence classes – which, for $x \in \mathsf{var}(q)$, we denote by $[x]_q$ – of the variables appearing in $q(x_1, \ldots, x_n)$.

Definition 4.21. We call $q(x_1, \ldots, x_n)$ *acyclic* if $\mathfrak{A}_q$ is acyclic. We say that a UCQ is *acyclic* iff each of its constituent CQs is.

Unlike structures, CQs have answer variables as distinguished elements. It turns out that many of our results only hold if we require that the answer variables are guarded by an atom in the body of the CQ at hand. The following definition strengthens acyclicity to that condition:

Definition 4.22. We say that $q(x_1, \ldots, x_n)$ is *strictly acyclic*, if it has a guarded tree decomposition $\delta = (\mathcal{T}, (X_v)_{v \in T})$ where $\{[x_1]_q, \ldots, [x_n]_q\} \subseteq X_v$ for some $v \in T$.

Hence, $q(x_1, \ldots, x_n)$ is strictly acyclic iff all its answer variables are summoned in one bag of δ. Notice that the body of every guarded TGD is trivially strictly acyclic, and that every CQ that has at most one answer variable (and thus also Boolean ones) are strictly acyclic as well.

Example 4.23. The CQ

$$q(x, z) \coloneqq \exists y \, (R(x, y) \wedge R(y, z))$$

is acyclic but not strictly acyclic. On the other hand,

$$q'(x) \coloneqq \exists y, z \, (R(x, y) \wedge R(y, z)) \equiv \exists z \, q(x, z)$$

is clearly strictly acyclic. ⊣

Recall that a first-order formula is *guarded* if each of its quantifier occurrences is of either forms

$$\forall \bar{x} \, (\alpha \to \varphi) \quad \text{and} \quad \exists \bar{x} \, (\alpha \wedge \varphi),$$

where α is an atomic formula that contains all the free variables of φ as arguments – we say that α *guards* φ. We call a formula *strictly guarded* if it is either atomic or of the form $\exists \bar{x} \, (\alpha \wedge \varphi)$, where (i) α guards φ, and (ii) φ is strictly guarded as well. Here we include the case where $\bar{x}$ is the empty sequence of variables, so that formulas of the form $\alpha \wedge \varphi$ (where α guards φ and φ is strictly guarded) also belong to the class of strictly guarded formulas.

Notice that every guarded sentence that is only composed of the logical connectives $\exists$ and $\wedge$ can be written as a strictly guarded formula. Moreover, it turns out that strictly guarded formulas correspond precisely to the strictly acyclic queries [77], and thus Boolean acyclic queries correspond to guarded sentences built up using conjunctions and existential quantification [81]:

Lemma 4.24. *A conjunctive query is strictly acyclic iff it is equivalent to a strictly guarded formula.*

Example 4.25. Consider again the CQ $q'(x) = \exists y, z\,(R(x,y) \land R(y,z))$ from Example 4.23. Clearly, $q'(x)$ is equivalent to the strictly guarded formula

$$\varphi(x) \coloneqq \exists y\,(R(x,y) \land \exists z\,(R(y,z) \land R(y,z))).$$

Here we pedantically followed the definition of strictly guarded formulas, which is why the subformula $R(y,z) \land R(y,z)$ contains a repetition of the same atom. We will adopt a more liberal notion when actually working with strictly guarded formulas so that $R(y,z)$ is considered itself to be strictly guarded.

Notice also that the nesting of quantifier occurrences allows us to reuse variables so that $\varphi(x)$ is equivalent to

$$\exists y\,(R(x,y) \land \exists x\,(R(y,x) \land R(y,x))) \equiv \exists y\,(R(x,y) \land \exists x\,R(y,x)).$$

Thus, care has to be taken when passing from a strictly guarded formula to an equivalent strictly acyclic query – variables have to be consistently renamed due to the fact that CQs are in prenex-form. ⊣

Guarded (bi)simulations Guarded bisimulations (see, e.g., [5, 20, 88]) generalize the modal bisimulations to the hypergraphs and arbitrary structures. In fact, guarded bisimulations are pivotal for the nice model-theoretic properties of the guarded fragment, just as standard modal bisimulations are for those of modal logics [88, 132]. We can view guarded bisimulations as winning strategies for the duplicator in a back&forth-game that accounts for the quantification pattern of guarded first-order formulas [88]. Positions in this guarded bisimulation game on two $\mathbf{S}$-structures $\mathfrak{A}$ and $\mathfrak{B}$ are partial isomorphisms between $\mathfrak{A}$ and $\mathfrak{B}$ whose domains (respectively, images) are guarded sets in $\mathfrak{A}$ (respectively, $\mathfrak{B}$). Following [88], let us write $p\colon \bar{a} \mapsto \bar{b}$ to indicate that p is a partial function from $\mathsf{dom}(\mathfrak{A})$ to $\mathsf{dom}(\mathfrak{B})$ whose domain is $[\bar{a}]$ and whose image is $[\bar{b}]$ with $b_i = p(a_i)$. For such a position $p\colon \bar{a} \mapsto \bar{b}$, we require that (i) $\bar{a}$ is a guarded tuple in $\mathfrak{A}$, (ii) $\bar{b}$ a guarded tuple in $\mathfrak{B}$, and (iii) p is an isomorphism between the induced substructures $\mathfrak{A} \restriction [\bar{a}]$ and $\mathfrak{B} \restriction [\bar{b}]$.

Suppose the game is at position $p\colon \bar{a} \mapsto \bar{b}$. When the spoiler makes a move, he first chooses a structure among $\mathfrak{A}$ and $\mathfrak{B}$, say he chooses $\mathfrak{A}$. The spoiler selects a guarded tuple $\bar{a}'$ of $\mathfrak{A}$ and some (guarded) tuple $\bar{a}_0$ such that $[\bar{a}_0] \subseteq [\bar{a}] \cap [\bar{a}']$. Upon that, the duplicator has to find an extension $\bar{b}'$ of the tuple $\bar{b}_0 \coloneqq p(\bar{a}_0)$ such that $\bar{b}'$ is guarded and the map $p'\colon \bar{a}' \mapsto \bar{b}'$ is again an isomorphism between $\mathfrak{A} \restriction [\bar{a}']$ and $\mathfrak{B} \restriction [\bar{b}']$. If the spoiler chooses to play on the side of $\mathfrak{B}$, the duplicator has to mirror the spoiler's move on $\mathfrak{A}$ accordingly, with the role of p taken by p^{-1}.

Definition 4.26 ([88]). For two $\mathbf{S}$-structures $\mathfrak{A}$ and $\mathfrak{B}$, a set of partial maps Z between $\mathsf{dom}(\mathfrak{A})$ and $\mathsf{dom}(\mathfrak{B})$ is a *guarded bisimulation* if it satisfies the following conditions, for every $p\colon \bar{a} \mapsto \bar{b}$ in Z:

(i) p is an isomorphism between $\mathfrak{A} \restriction [\bar{a}]$ and $\mathfrak{B} \restriction [\bar{b}]$. (*atom equivalence*)

(ii) For every guarded tuple $\bar{b}'$ of $\mathfrak{B}$ and each $\bar{b}_0$ with $[\bar{b}_0] \subseteq [\bar{b}] \cap [\bar{b}']$, there is some guarded tuple $\bar{a}'$ of $\mathfrak{A}$ and a $p'\colon \bar{a}' \mapsto \bar{b}'$ in Z such that $p'^{-1}(\bar{b}_0) = p^{-1}(\bar{b}_0)$. (*back*)

(iii) For every guarded tuple $\bar{a}'$ of $\mathfrak{A}$ and each $\bar{a}_0$ with $[\bar{a}_0] \subseteq [\bar{a}] \cap [\bar{a}']$, there is some guarded tuple $\bar{b}'$ of $\mathfrak{B}$ and a $p' \colon \bar{a}' \mapsto \bar{b}'$ in Z such that $p'(\bar{a}_0) = p(\bar{a}_0)$. (*forth*)

We write $\mathfrak{A}, \bar{a} \sim_{\mathrm{g}} \mathfrak{B}, \bar{b}$ if there is a guarded bisimulation between $\mathfrak{A}$ and $\mathfrak{B}$ that contains some $p \colon \bar{a} \mapsto \bar{b}$.

Notation. In the following, if a map $a_1, \ldots, a_n \mapsto b_1, \ldots, b_n$ is specified to be in a guarded bisimulation Z and $i_1, \ldots, i_k \in [n]$, then we assume also that all the maps $a_{i_1}, \ldots, a_{i_k} \mapsto b_{i_1}, \ldots, b_{i_k}$ are in Z, without mentioning them explicitly.

As said above, satisfaction of guarded formulas is invariant under guarded bisimulations [5, 20, 88]:

LEMMA 4.27. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be $\mathbf{S}$-structures. For every guarded formula $\varphi(\bar{x})$ and guarded tuples $\bar{a}$ and $\bar{b}$,*

$$\mathfrak{A}, \bar{a} \sim_{\mathrm{g}} \mathfrak{B}, \bar{b} \implies \Big(\mathfrak{A} \models \varphi(\bar{a}) \iff \mathfrak{B} \models \varphi(\bar{b})\Big).$$

Let us relax the notion of guarded bisimulation a bit by (i) only requiring that the maps $p \colon \bar{a} \mapsto \bar{b}$ are partial homomorphisms rather than partial isomorphisms, and (ii) by only requiring that the forth-condition is satisfied. We call a set of partial maps Z between $\mathfrak{A}$ and $\mathfrak{B}$ that satisfies these two conditions a *guarded simulation*. We write $\mathfrak{A}, \bar{a} \preceq_{\mathrm{g}} \mathfrak{B}, \bar{b}$ if there is a guarded simulation between $\mathfrak{A}$ and $\mathfrak{B}$ that contains some $p \colon \bar{a} \mapsto \bar{b}$. Obviously, every guarded bisimulation is also a guarded simulation, while the converse is not true. Guarded simulations are adequate for sustaining satisfaction of strictly guarded formulas (and thus strictly acyclic queries):

LEMMA 4.28. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be $\mathbf{S}$-structures. For every strictly guarded formula $\varphi(\bar{x})$ and guarded tuples $\bar{a}$ and $\bar{b}$:*

$$\mathfrak{A}, \bar{a} \preceq_{\mathrm{g}} \mathfrak{B}, \bar{b} \implies \Big(\mathfrak{A} \models \varphi(\bar{a}) \implies \mathfrak{B} \models \varphi(\bar{b})\Big).$$

PROOF. Let Z be a guarded simulation containing $p \colon \bar{a} \mapsto \bar{b}$. We proceed by structural induction on $\varphi(\bar{x})$. The case where $\varphi(\bar{x})$ is atomic is trivial, since $p \colon \bar{a} \mapsto \bar{b}$ is a homomorphism from $\mathfrak{A} \restriction [\bar{a}]$ to $\mathfrak{B} \restriction [\bar{b}]$.

Suppose now that $\varphi(\bar{x}) = \exists \bar{y} \, (\alpha(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y}))$, where α guards ψ, and ψ is strictly guarded. Suppose further that $\mathfrak{A} \models \varphi(\bar{a})$ and that $p \colon \bar{a} \mapsto \bar{b}$ is in Z. From $\mathfrak{A} \models \varphi(\bar{a})$, we know that there is a tuple $\bar{c}$ such that $\mathfrak{A} \models \alpha(\bar{a}, \bar{c}) \wedge \psi(\bar{a}, \bar{c})$. Thus, the tuple $\bar{a}' \coloneqq \bar{a}\bar{c}$ is a guarded tuple of $\mathfrak{A}$, whence it follows that there is a $p' \colon \bar{a}' \mapsto \bar{b}'$ in Z with $p'(\bar{a}) = p(\bar{a}) = \bar{b}$. By induction hypothesis, we know that $\mathfrak{B} \models \psi(\bar{b}')$, and we also know that $\mathfrak{B} \models \alpha(\bar{b}')$ since $p' \colon \bar{a}' \mapsto \bar{b}'$ is a homomorphism from $\mathfrak{A} \restriction [\bar{a}']$ to $\mathfrak{B} \restriction [\bar{b}']$. Thus, $\mathfrak{B} \models \exists \bar{y} \, (\alpha(\bar{b}, \bar{y}) \wedge \psi(\bar{b}, \bar{y}))$, i.e., $\mathfrak{B} \models \varphi(\bar{b})$ as desired. □

Guarded unraveling Our goal here is to uniformly construct tree-like models (more precisely, acyclic models) from models of guarded formulas. The construction is named *guarded unraveling* [5, 88] and extends the standard unraveling of Kripke structures to the case of arbitrary structures.

In the following paragraph, we largely follow the notions introduced in [4, 28]. Fix an $\mathbf{S}$-structure $\mathfrak{A}$. Let Π be the set of finite sequences of the form $X_0X_1\cdots X_n$, where, for $i \geq 0$, (i) X_i is a guarded set in $\mathfrak{A}$, and (ii) $X_{i+1} = X_i \cup \{a\}$ for some $a \in \mathsf{dom}(\mathfrak{A}) \setminus X_i$, or $X_i \supseteq X_{i+1}$. The sequences in Π can be arranged in a tree by their natural prefix-order, and each sequence $\pi = X_0X_1\cdots X_n$ identifies a unique node in this tree. In this context, we say that $a \in \mathsf{dom}(\mathfrak{A})$ is *represented at* π whenever $a \in X_n$. Two sequences π, π' are *a-equivalent*, if a is represented at each node on the unique shortest path between π and π'. For a represented at π, we denote by $[\pi]_a$ the a-equivalence class of π, and let us specify that $[\pi]_a = [\pi']_b$ iff (i) $a = b$, and (ii) π and π' are a-equivalent.

Definition 4.29. The *guarded unraveling of* $\mathfrak{A}$ is the $\mathbf{S}$-structure $\mathfrak{A}^*$ with domain

$$\mathsf{dom}(\mathfrak{A}^*) \coloneqq \{[\pi]_a \mid a \text{ is represented at } \pi\},$$

and

$$\mathfrak{A}^* \models R([\pi_1]_{a_1}, \ldots, [\pi_n]_{a_n}) \iff_{df} \mathfrak{A} \models R(a_1, \ldots, a_n) \text{ and } \exists \pi \in \Pi, \forall i \in \{1, \ldots, n\} \colon [\pi]_{a_i} = [\pi_i]_{a_i},$$

for all $R/n \in \mathbf{S}$.

Lemma 4.30. *For every $\mathbf{S}$-structure $\mathfrak{A}$ and every guarded set X in $\mathfrak{A}$, the guarded unraveling $\mathfrak{A}^*$ is acyclic.*

Proof. Let $\delta = (\mathcal{T}, (Y_\pi)_{\pi \in T})$, where $\mathcal{T}$ is the natural tree that arises from ordering the sequences in Π by their prefixes, where Π is as in the definition of $\mathfrak{A}^*$. For $\pi \in T$, let $Y_\pi \coloneqq \{[\pi]_a \mid a \text{ is represented at } \pi\}$. We need to show that δ is an appropriate guarded tree decomposition witnessing that $\mathfrak{A}^*$ is acyclic.

Let $[\pi]_a \in \mathsf{dom}(\mathfrak{A}^*)$ and suppose $\theta, \rho \in T$ are two nodes such that $[\pi]_a \in Y_\theta \cap Y_\rho$. We need to show that $[\pi]_a$ appears in all bags on the unique shortest path between θ and ρ in $\mathcal{T}$. Let τ be a node on that path. Now $[\pi]_a \in Y_\theta \cap Y_\rho$ means that $[\pi]_a = [\theta]_a = [\rho]_a$, whence it follows that θ and ρ are a-equivalent, and thus a is represented at τ. This means that $[\tau]_a = [\theta]_a = [\rho]_a = [\pi]_a$ and so $[\pi]_a \in Y_\tau$ as needed.

Suppose $\mathfrak{A}^* \models R([\pi_1]_{a_1}, \ldots, [\pi_n]_{a_n})$ for some $R/n \in \mathbf{S}$. Then there is a $\pi \in T$ such that $[\pi]_{a_i} = [\pi_i]_{a_i}$, for all $i = 1, \ldots, n$. Hence, $a_1, \ldots, a_n$ are all represented at π and so $\{[\pi_1]_{a_1}, \ldots, [\pi_n]_{a_n}\} \subseteq Y_\pi$.

It remains to be shown that δ is guarded. Let $\pi \in T$ and consider the set Y_π. The last element of π, say Z, is guarded in $\mathfrak{A}$. Hence, there are $a_1, \ldots, a_m$ such that $Z \subseteq \{a_1, \ldots, a_m\}$ and $\mathfrak{A} \models R(a_1, \ldots, a_m)$ for some $R/m \in \mathbf{S}$. Let $\{b_1, \ldots, b_s\} = \{a_1, \ldots, a_m\} \setminus Z$ and define $\rho \coloneqq \pi \cdot (Z \cup \{b_1\}) \cdot (Z \cup \{b_1, b_2\}) \cdots (Z \cup \{b_1, \ldots, b_s\})$. Then $\mathfrak{A}^* \models R([\rho]_{a_1}, \ldots, [\rho]_{a_m})$, as desired. □

Given the guarded unraveling $\mathfrak{A}^*$ of $\mathfrak{A}$, we let $\hat{\pi}$ be the natural projection

$$\hat{\pi} \colon [\pi]_a \longmapsto a, \qquad \text{for } [\pi]_a \in \mathsf{dom}(\mathfrak{A}^*).$$

Lemma 4.31. *The function $\hat{\pi}$ is a (weak) homomorphism from $\mathfrak{A}^*$ to $\mathfrak{A}$.*

PROOF. If $\mathfrak{A}^* \models R([\pi_1]_{a_1}, \ldots, [\pi_n]_{a_n})$ then by definition $\mathfrak{A} \models R(a_1, \ldots, a_n)$. □

Using $\hat{\pi}$, we can easily write down a guarded bisimulation between $\mathfrak{A}^*$ and $\mathfrak{A}$:

LEMMA 4.32. *For every structure $\mathfrak{A}$ and every guarded tuple $s_1, \ldots, s_k$ in $\mathfrak{A}^*$, it holds that*

$$\mathfrak{A}^*, s_1, \ldots, s_k \sim_{\mathrm{g}} \mathfrak{A}, \hat{\pi}(s_1), \ldots, \hat{\pi}(s_k).$$

PROOF. Let Z be the relation that contains for every guarded tuple $\bar{t} = t_1, \ldots, t_n$ of $\mathfrak{A}^*$ the map $p_{\bar{t}}$ with

$$p_{\bar{t}} \colon t_1, \ldots, t_n \longmapsto \hat{\pi}(t_1), \ldots, \hat{\pi}(t_n).$$

We claim that Z is a guarded bisimulation between $\mathfrak{A}^*$ and $\mathfrak{A}$ that witnesses

$$\mathfrak{A}^*, s_1, \ldots, s_k \sim_{\mathrm{g}} \mathfrak{A}, \hat{\pi}(s_1), \ldots, \hat{\pi}(s_k),$$

for every guarded tuple $\bar{s} \coloneqq s_1, \ldots, s_k$ of $\mathfrak{A}^*$. Suppose $\bar{s}$ is of the form $[\pi_1]_{a_1}, \ldots, [\pi_k]_{a_k}$ and $\bar{a} \coloneqq \hat{\pi}(\bar{s}) = a_1, \ldots, a_k$. Since $\bar{s}$ is guarded in $\mathfrak{A}^*$, there is a $\pi \in \Pi$ such that, for $i = 1, \ldots, k$, we have $[\pi]_{a_i} = [\pi_i]_{a_i}$.

We first argue that $p_{\bar{s}} \colon s_1, \ldots, s_k \mapsto [\pi_1]_{a_1}, \ldots, [\pi_k]_{a_k}$ is an isomorphism between $\mathfrak{A}^* \restriction [\bar{s}]$ and $\mathfrak{A} \restriction [\hat{\pi}(\bar{s})]$. Suppose that $p_{\bar{s}}(s_i) = p_{\bar{s}}(s_j)$ for some $i, j \in [k]$. Hence, $a_i = a_j$ and thus $[\pi_i]_{a_i} = [\pi]_{a_i} = [\pi]_{a_j} = [\pi_j]_{a_j}$. It follows that $s_i = s_j$ and hence $p_{\bar{s}}$ is injective. The fact that $p_{\bar{s}}$ is onto is easy to see as $p_{\bar{s}}([\{a_1, \ldots, a_k\}]_{a_1}, \ldots, [\{a_1, \ldots, a_k\}]_{a_k}) = a_1, \ldots, a_k$. For any sub-tuple $s_{i_1}, \ldots, s_{i_m}$ of $\bar{s}$ with $p_{\bar{s}}(s_{i_j}) = a_{i_j}$, and any $R/m \in \mathbf{S}$ it holds that

$$\begin{aligned}
\mathfrak{A} \restriction [\hat{\pi}(\bar{s})] \models R(a_{i_1}, \ldots, a_{i_m}) &\iff \mathfrak{A}^* \restriction [\bar{s}] \models R([\pi]_{a_{i_1}}, \ldots, [\pi]_{a_{i_m}}) \\
&\iff \mathfrak{A}^* \restriction [\bar{s}] \models R([\pi_{i_1}]_{a_{i_1}}, \ldots, [\pi_{i_m}]_{a_{i_m}}) \\
&\iff \mathfrak{A}^* \restriction [\bar{s}] \models R(s_{i_1}, \ldots, s_{i_m}).
\end{aligned}$$

Hence, $p_{\bar{s}}$ is indeed an isomorphism between $\mathfrak{A}^* \restriction [\bar{s}]$ and $\mathfrak{A} \restriction [\hat{\pi}(\bar{s})]$.

It remains to verify the back&forth-conditions. Suppose $\bar{b} \coloneqq b_1, \ldots, b_m$ is a guarded tuple of $\mathfrak{A}$ and $\bar{b}_0$ a sub-tuple of $\bar{b}$ such that $[\bar{b}_0] \subseteq [\bar{a}] \cap [\bar{b}]$. For $i = 1, \ldots, m$, let $\rho \coloneqq \pi \cdot \{b_1, \ldots, b_m\}$, and let $\bar{t} \coloneqq ([\rho]_{b_1}, \ldots, [\rho]_{b_m})$. By definition, $p_{\bar{t}} \colon \bar{t} \mapsto \bar{b}$ is in Z and $p_{\bar{t}}^{-1}(\bar{b}_0) = p_{\bar{s}}^{-1}(\bar{b}_0)$. This verifies the back-condition.

Suppose now that $\bar{t} \coloneqq (t_1, \ldots, t_m)$ is a guarded tuple of $\mathfrak{A}^*$ and let $t_i = [\rho_i]_{b_i}$. Consider also a tuple $\bar{t}_0$ such that $[\bar{t}_0] \subseteq [\bar{s}] \cap [\bar{t}]$. Let $\bar{t}_0 = (t_{i_1}, \ldots, t_{i_l}) = (s_{j_1}, \ldots, s_{j_l})$ for some appropriate $i_1, \ldots, i_l$ and $j_1, \ldots, j_l$. We claim that $\bar{b} \coloneqq (b_1, \ldots, b_m)$ is the guarded tuple of $\mathfrak{A}$ we are looking for. By definition, $p_{\bar{t}} \colon t_1, \ldots, t_m \mapsto b_1, \ldots, b_m$ is in Z. Moreover, since $\bar{t}_0 = (t_{i_1}, \ldots, t_{i_l}) = (s_{j_1}, \ldots, s_{j_l})$, we also have also $b_{i_r} = a_{j_r}$ for $r = 1, \ldots, l$, and so $p_{\bar{t}}(\bar{t}_0) = p_{\bar{s}}(\bar{t}_0)$, as required. □

COROLLARY 4.33. *For every guarded formula $\varphi(x_1, \ldots, x_k)$ and every guarded tuple $s_1, \ldots, s_k$ of $\mathfrak{A}^*$ it holds that*

$$\mathfrak{A} \models \varphi(\hat{\pi}(s_1), \ldots, \hat{\pi}(s_k)) \iff \mathfrak{A}^* \models \varphi(s_1, \ldots, s_k).$$

PROOF. Immediate by Lemmas 4.27 and 4.32. □

Acyclic Witnesses

The following theorem states our tree witness property:

Theorem 4.34. *Suppose* $Q_1 = (\mathbf{S}, \mathcal{O}_1, G_1(\bar{x}))$ *and* $Q_2 = (\mathbf{S}, \mathcal{O}_2, G_2(\bar{x}))$ *are OMQs from* $(\mathsf{G}, \mathsf{AQ})$*. Then the following are equivalent:*

(i) $Q_1 \subseteq Q_2$.

(ii) *For every acyclic* $\mathbf{S}$*-database* $\mathfrak{D}$*,* $Q_1(\mathfrak{D}) \subseteq Q_2(\mathfrak{D})$.

Hence, whenever Q_1 is *not* contained in Q_2, non-containment is already witnessed by an acyclic database. Since acyclicity can be understood to capture some form of tree-likeliness, one can say that Theorem 4.34 establishes that we can focus on tree-like instances when deciding containment and this will later be exploited when we resort to tree automata in order to decide containment.

To prove Theorem 4.34, we prove some additional results first.

Lemma 4.35. *Suppose* $\mathfrak{A}$ *and* $\mathfrak{A}'$ *are* $\mathbf{S}$*-structures, and let* $\mathcal{O}$ *be a set of guarded TGDs. Every guarded bisimulation between* $\mathfrak{A}$ *and* $\mathfrak{A}'$ *can be extended to a guarded simulation between* $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ *and* $\mathsf{chase}(\mathfrak{A}', \mathcal{O})$.

Proof. Suppose Z is a guarded bisimulation between $\mathfrak{A}$ and $\mathfrak{A}'$, and let $\pi = \mathfrak{J}_0, \mathfrak{J}_1, \ldots$ be an appropriate chase sequence for $\mathfrak{A}$ and $\mathcal{O}$ such that $\mathfrak{J}_0 = \mathfrak{A}$ and $\mathsf{chase}(\mathfrak{A}, \mathcal{O}) = \bigcup_{i \geq 0} \mathfrak{J}_i$. We shall extend Z successively to a guarded simulation Z according to π.

We now show by induction on i that here are sets of partial maps $Z_0 \subseteq Z_1 \subseteq \cdots$ such that, for every $i \geq 0$, Z_i a guarded simulation between $\mathfrak{J}_i$ and $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$.

For $i = 0$, we simply set $Z_0 \coloneqq Z$. Since Z_0 is already a guarded bisimulation between $\mathfrak{A}$ and $\mathfrak{A}'$, and since $\mathsf{chase}(\mathfrak{A}', \mathcal{O})$ is just an extension of $\mathfrak{A}'$, it is clear that Z_0 is a guarded simulation between $\mathfrak{J}_0 = \mathfrak{A}$ and $\mathsf{chase}(\mathfrak{A}', \mathcal{O})$.

Assume now that $i \geq 1$. We are going to define Z_i as follows, assuming by induction hypothesis that Z_{i-1} is already a guarded simulation between $\mathfrak{J}_{i-1}$ and $\mathsf{chase}(\mathfrak{A}', \mathcal{O})$. Suppose $\mathfrak{J}_i$ results from $\mathfrak{J}_{i-1}$ by an application of (h, τ), where τ is a guarded rule of $\mathcal{O}$ of the form

$$q_\tau(x_1, \ldots, x_n) \to \exists \bar{y}\, R(x_1, \ldots, x_n, y_1, \ldots, y_m),$$

where $q_\tau(x_1, \ldots, x_n)$ is a CQ, and h a homomorphism from $q_\tau(x_1, \ldots, x_n)$ to $\mathfrak{J}_{i-1}$. Assume moreover that there is a map in Z_{i-1} of the form

$$p\colon h(x_1), \ldots, h(x_n) \longmapsto b_1, \ldots, b_n.$$

Let $R(h(x_1), \ldots, h(x_n), \lambda_1, \ldots, \lambda_m)$ be the atom derived by the application of (h, τ), where $\lambda_1, \ldots, \lambda_m$ are fresh labeled nulls. Assume that $\lambda'_1, \ldots, \lambda'_m$ are elements such that

$$\mathsf{chase}(\mathfrak{A}', \mathcal{O}) \models R(b_1, \ldots, b_n, \lambda'_1, \ldots, \lambda'_m).$$

Notice that such elements must exist since Z_{i-1} is a guarded simulation, and therefore by Lemma 4.28 $\mathsf{chase}(\mathfrak{A}', \mathcal{O}) \models q_\tau(b_1, \ldots, b_n)$. Then we set

$$Z_i \coloneqq Z_{i-1} \cup \{p_i\colon h(x_1), \ldots, h(x_n), \lambda_1, \ldots, \lambda_m \longmapsto b_1, \ldots, b_n, \lambda'_1, \ldots, \lambda'_m\}.$$

It is not hard to check that Z_i is a guarded simulation between $\mathfrak{J}_i$ and $\mathsf{chase}(\mathfrak{A}', \mathcal{O})$ since $\mathfrak{J}_i = \mathfrak{J}_{i-1} \cup \{R(h(x_1), \ldots, h(x_n), \lambda_1, \ldots, \lambda_m)\}$. This completes the induction step.

Now let Z^* be the set of partial maps $\bigcup_{i \geq 0} Z_i$. It is easy to see that Z^* is a guarded simulation between $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ and $\mathsf{chase}(\mathfrak{A}', \mathcal{O})$. □

Lemma 4.36. *Suppose $Q = (\mathbf{S}, \mathcal{O}, G(\bar{x}))$ is an OMQs from $(\mathsf{G}, \mathsf{AQ})$, and suppose that $\mathfrak{D} \models Q(\bar{a})$ for some $\mathbf{S}$-database $\mathfrak{D}$. Then there is an acyclic $\mathbf{S}$-database $\mathfrak{B}$ such that the following hold:*

(i) $\mathfrak{B} \models Q(\bar{a})$.

(ii) *There is a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$ that is the identity on $\bar{a}$.*

Proof. Let $\mathfrak{D}$ be an $\mathbf{S}$-database such that $\mathfrak{D} \models Q(\bar{a})$. Let $\mathfrak{D}^*$ be the guarded unraveling of $\mathfrak{D}$, and notice that the map $\hat{\pi}$ which sends any equivalence class $[\pi]_a$ to $a \in \mathsf{adom}(\mathfrak{D})$ is a homomorphism from $\mathfrak{D}^*$ to $\mathfrak{D}$.

By Lemma 4.32, we know that there is a guarded bisimulation Z between $\mathfrak{D}^*$ and $\mathfrak{D}$ that contains all maps of the form

$$p_{\bar{s}} \colon s_1, \ldots, s_k \longmapsto \hat{\pi}(s_1), \ldots, \hat{\pi}(s_k),$$

for every guarded tuple $s_1, \ldots, s_k$ of $\mathfrak{D}^*$. Obviously, the set of maps $Z^{-1} \coloneqq \{p^{-1} \mid p \in Z\}$ is a guarded bisimulation between $\mathfrak{D}$ and $\mathfrak{D}^*$.

By Lemma 4.35, we know that Z^{-1} can be extended to a guarded simulation between $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ and $\mathsf{chase}(\mathfrak{D}^*, \mathcal{O})$, and so

$$\mathsf{chase}(\mathfrak{D}, \mathcal{O}), \hat{\pi}(s_1), \ldots, \hat{\pi}(s_k) \preceq_{\mathrm{g}} \mathsf{chase}(\mathfrak{D}^*, \mathcal{O}), s_1, \ldots, s_k,$$

for every guarded tuple $s_1, \ldots, s_k$ of $\mathfrak{D}^*$. Let $\bar{a} = a_1, \ldots, a_n$ and let

$$t_i \coloneqq [\{a_1, \ldots, a_n\}]_{a_i}, \quad \text{for } i = 1, \ldots, n.$$

We know that $\mathsf{chase}(\mathfrak{D}, \mathcal{O}) \models G(\hat{\pi}(t_1), \ldots, \hat{\pi}(t_n))$ by assumption, whence by Lemma 4.28 we obtain that $\mathsf{chase}(\mathfrak{D}^*, \mathcal{O}) \models G(t_1, \ldots, t_n)$. By Lemmas 4.20 and 4.30 we know that $\mathsf{chase}(\mathfrak{D}^*, \mathcal{O})$ is acyclic and thus it has a guarded tree decomposition $\delta = (\mathcal{T}, (X_v)_{v \in T})$. From $\mathsf{chase}(\mathfrak{D}^*, \mathcal{O}) \models G(t_1, \ldots, t_n)$, we obtain $(\mathfrak{D}^*, \mathcal{O}) \models G(t_1, \ldots, t_n)$, whence by the standard compactness theorem of first-order logic we know that there is a finite database $\mathfrak{D}' \subseteq \mathfrak{D}^*$ such that $(\mathfrak{D}', \mathcal{O}) \models G(t_1, \ldots, t_n)$. Let δ' be the tree decomposition that results from δ by restricting its bags to $\mathsf{adom}(\mathfrak{D}')$, i.e., $\delta' = (\mathcal{T}', (X'_v)_{v \in T'})$ with $\mathcal{T}' \coloneqq \mathcal{T}$ and $X'_v \coloneqq X_v \cap \mathsf{adom}(\mathfrak{D}')$. Notice that δ' may still be infinite, but only due to the repetition of nodes. Formally, we call a node $w \in T'$ a *duplicate* of $v \in T'$ if $X'_w = X'_v$. We *mark* now the duplicate nodes in δ' and for each distinct X'_v we only keep one duplicate *unmarked*. Since $\bigcup_{v \in T'} X'_v$ is finite, it is clear that every subtree in $\mathcal{T}'$ contains only finitely many unmarked nodes. We let $\delta^* = (\mathcal{T}^*, (Y_v)_{v \in T^*})$ be the tree decomposition that arises from δ' by deleting those subtrees from δ' that only contain marked nodes.

We still have to ensure that δ^* is guarded. Suppose $v \in \mathcal{T}^*$ is such that there is no guarded set Y in $\mathfrak{D}'$ such that $Y \supseteq Y_v$ (call such a node *bad*). However, by construction,

there is a guarded set X in $\mathfrak{D}^*$ such that $X \supseteq Y_v$ and thus there is a fact $R(\bar{s})$ in $\mathfrak{D}^*$ such that $[\bar{s}] \supseteq X \supseteq Y_v$. We add $R(\bar{s})$ to $\mathfrak{D}'$ and we repeat this process exhaustively for all bad nodes. Call the resulting structure $\mathfrak{B}'$. It is clearly finite and acyclic (it must be acyclic since $\mathfrak{D}^*$ is).

Let $\mathfrak{B}$ be the $\mathbf{S}$-database that is obtained from $\mathfrak{B}'$ by renaming the elements of $\mathsf{dom}(\mathfrak{B}')$ via a bijection ρ as follows: $t_1, \ldots, t_n$ are respectively renamed to $a_1, \ldots, a_n$, all the other elements appearing in $\mathsf{dom}(\mathfrak{B}')$ to elements of const that are distinct from $a_1, \ldots, a_n$. Then $\mathfrak{B} \models Q(\bar{a})$, and the function

$$h \colon s \longmapsto \rho(\hat{\pi}(s))$$

is clearly a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$ that is the identity on $\bar{a} = a_1, \ldots, a_n$. Thus, $\mathfrak{B}$ is the database we are looking for. □

Proof of Theorem 4.34. The direction from (i) to (ii) is trivial, and so we focus on the other direction. Assume that there is an $\mathbf{S}$-database $\mathfrak{D}$ such that $Q_1(\mathfrak{D}) \not\subseteq Q_2(\mathfrak{D})$. Hence, $\mathfrak{D} \models Q_1(\bar{a})$ but $\mathfrak{D} \not\models Q_2(\bar{a})$ for some (guarded) tuple $\bar{a}$. By Lemma 4.36, there is an acyclic $\mathbf{S}$-database $\mathfrak{B}$ such that $\mathfrak{B} \models Q_1(\bar{a})$, and there is a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$ that is the identity on $\bar{a}$. Since Q_2 is closed under weak homomorphisms (cf. Proposition 3.18), we obtain $\mathfrak{B} \not\models Q(\bar{a})$ as well. □

Encoding Tree-Like Structures

In the following, we are going to show how we encode tree-like structures into trees over a finite alphabet. This will be used later when we exploit automata techniques in order to solve containment for guarded-based classes. Formally speaking, what we do encode is (i) a tree decomposition δ of a structure $\mathfrak{A}$ together with (ii) information that specifies which facts of $\mathfrak{A}$ are true in which bag, i.e., information about the facts of the substructure of $\mathfrak{A}$ induced by the corresponding bag. This is formalized as follows.

Let $\mathfrak{A}$ be a database, and $\delta = (\mathcal{T}, (X_v)_{v \in T})$ a tree decomposition of $\mathfrak{A}$. An *adornment* of the pair $(\mathfrak{A}, \delta)$ is a function $\eta \colon T \to 2^{\mathfrak{A}}$ (i.e., η assigns sets of facts of $\mathfrak{A}$ to nodes of δ) such that

(i) $\eta(v) \subseteq \mathfrak{A} \restriction X_v$ for all $v \in T$, and

(ii) for every fact α of $\mathfrak{A}$, there is some $v \in T$ such that $\alpha \in \eta(v)$.

Therefore, the pair (δ, η) can be viewed as a representation of the structure $\mathfrak{A}$ along with a tree decomposition of it. Notice that, in the first item, we do not require that $\eta(v) = \mathfrak{A} \restriction X_v$. This has technical reasons and will be used later in Chapter 5, where we are going to reuse this terminology.

Encoding We now explain how to encode a tuple $(\mathfrak{A}, \delta, \eta)$, formed by an $\mathbf{S}$-structure $\mathfrak{A}$, a tree decomposition δ of $\mathfrak{A}$ of width $w - 1$, and an adornment η of $(\mathfrak{A}, \delta)$, as a $\Gamma_{\mathbf{S},w}$-labeled tree t of degree, where $\Gamma_{\mathbf{S},w}$ is an alphabet of doubly exponential size in w and exponential in $\mathbf{S}$, such that each node of δ corresponds to exactly one node of t and vice versa.

Fix a schema $\mathbf{S}$ and let $w \geq 1$. Let $U_{\mathbf{S},w}$ be a set containing $2w$ distinct elements. Elements in $U_{\mathbf{S},w}$ are used to encode the elements in $\mathsf{dom}(\mathfrak{A})$ in trees. Neighboring nodes in such trees may describe overlapping pieces of the encoded structure. In particular, if one name is used in neighboring nodes, this means that the name at hand refers to the same element – this is why we use $2w$ elements for bags.

Let $\mathbb{K}_{\mathbf{S},w}$ be the finite schema defined as follows.

- For every $a \in U_{\mathbf{S},w}$, there is a unary relation $D_a \in \mathbb{K}_{\mathbf{S},w}$.
- For each n-ary relation symbol $R \in \mathbf{S}$ and every n-tuple $\bar{a}$ over $U_{\mathbf{S},w}$, there is a unary relation $R_{\bar{a}} \in \mathbb{K}_{\mathbf{S},w}$.

We define $\Gamma_{\mathbf{S},w} \coloneqq 2^{\mathbb{K}_{\mathbf{S},w}}$.

Fix a function $f \colon \mathsf{dom}(\mathfrak{A}) \to U_{\mathbf{S},w}$ such that different elements that occur in neighboring bags of δ are always assigned different elements from $U_{\mathbf{S},w}$. Using f, we can encode $(\mathfrak{A}, \delta, \eta)$ as a $\Gamma_{\mathbf{S},w}$-labeled tree $t_{\mathfrak{A},\delta,\eta}$ such that each node of $\mathcal{T}$ corresponds to exactly one node of $t_{\mathfrak{A},\delta,\eta}$ and vice versa. In the following, for a node v from $\mathcal{T}$, we denote the corresponding node of $t_{\mathfrak{A},\delta,\eta}$ by $\hat{v}$, and vice versa. For each node $v \in \mathcal{T}$ we have that the node $\hat{v}$ in $t_{\mathfrak{A},\delta,\eta}$ is labeled with a symbol $\theta_{\hat{v}}$ from $\Gamma_{\mathbf{S},w}$ in such a way that the following holds:

- $D_a \in \theta_{\hat{v}}$ iff $f(c) = a$ for some $c \in \mathsf{dom}(\mathfrak{A}) \cap X_v$.
- $R_{\bar{a}} \in \theta_{\hat{v}}$ iff $R(\bar{c}) \in \eta(v)$ and $f(\bar{c}) = \bar{a}$.

Intuitively, a symbol $D_a \in t_{\mathfrak{A},\delta,\eta}(v)$ means that the $a \in U_{\mathbf{S},w}$ is used to name an element, while $R_{\bar{a}} \in t_{\mathfrak{A},\delta,\eta}(v)$ means that the relation R holds for the elements named by $\bar{a}$ in the node v.

Notation. In the following, if $\mathsf{wd}(\mathbf{S}) = 0$, then by $\Gamma_{\mathbf{S}}$ we mean the alphabet $\Gamma_{\mathbf{S},1}$, and if $\mathsf{wd}(\mathbf{S}) > 0$, then we are going to write $\Gamma_{\mathbf{S}}$ for $\Gamma_{\mathbf{S},\mathsf{wd}(\mathbf{S})}$. An according convention is applied for $U_{\mathbf{S}}$ with respect to $U_{\mathbf{S},w}$.

Decoding While every structure $\mathfrak{A}$, together with a tree decomposition δ of it of width $w-1$ and an adornment for $(\mathfrak{A}, \delta)$, can be encoded into a $\Gamma_{\mathbf{S},w}$-labeled tree t, the converse is not true in general. However, as shown below, it is possible to define certain consistency conditions such that every *consistent* $\Gamma_{\mathbf{S},w}$-labeled t can be decoded into an $\mathbf{S}$-structure, denoted $[\![t]\!]$, whose tree-width is bounded by $w-1$.

Let t be a $\Gamma_{\mathbf{S},w}$-labeled tree. For $\rho \in \Gamma_{\mathbf{S}}$ we let $\mathsf{names}(\rho) \coloneqq \{a \mid D_a \in \rho\}$, and for $v \in \mathsf{dom}(t)$ we set $\mathsf{names}(v) \coloneqq \mathsf{names}(t(v))$. We say that t is *consistent*, if it satisfies the following properties:

(i) For each node $v \in \mathsf{dom}(t)$, it holds that $|\mathsf{names}(v)| \leq w$.

(ii) For every symbol $R_{\bar{a}} \in \mathbb{K}_{\mathbf{S},w}$ and every node $v \in \mathsf{dom}(t)$, if $R_{\bar{a}} \in t(v)$, then $[\bar{a}] \subseteq \mathsf{names}(v)$.

Suppose now that t is consistent. We show how to decode t into a structure $[\![t]\!]$ whose tree-width is at most $w-1$. Let a be a name used in t. We say that two nodes v, w of

t are *a-equivalent*, if $D_a \in t(u)$ for all nodes u on the unique shortest path between v and w. Clearly, a-equivalence defines an equivalence relation, and we denote by $[v]_a$ the a-equivalence class of v. More formally, we define

$$[v]_a \coloneqq \{(w, a) \mid w \text{ is } a\text{-equivalent to } v\},$$

so that $[v]_a = [w]_b$ only if $a = b$. The domain of $[\![t]\!]$ is the set

$$\mathsf{dom}([\![t]\!]) \coloneqq \{[v]_a \mid v \text{ is a node of } t \text{ and } a \in \mathsf{names}(v)\},$$

and, for $R/n \in \mathbf{S}$, we define

$$[\![t]\!] \models R([v_1]_{a_1}, \ldots, [v_n]_{a_n}) \iff_{df} \begin{array}{l} R_{a_1,\ldots,a_n} \in t(v) \text{ and} \\ \exists v \in \mathsf{dom}(t), \forall i \in \{1, \ldots, n\} \colon [v]_{a_i} = [v_i]_{a_i}. \end{array}$$

It is not hard to see that $[\![t]\!]$ is indeed well-defined. Let $\delta_t = (\mathcal{T}, (X_v)_{v \in T})$ be the *standard tree decomposition for* $[\![t]\!]$, where $\mathcal{T}$ is the same tree in structure as t, and $X_v \coloneqq \{[v]_a \mid a \in \mathsf{names}(v)\}$, for all $v \in T$. Moreover, we define the *standard adornment* η_t for $([\![t]\!], \delta_t)$ by

$$\eta_t(v) \coloneqq \{R([v]_{a_1}, \ldots, [v]_{a_k}) \mid R_{a_1,\ldots,a_k} \in t(v)\}, \quad \text{for } v \in T.$$

Lemma 4.37. *If t is consistent then δ_t is a tree decomposition of $[\![t]\!]$ of width at most $w - 1$, and η_t an adornment of $([\![t]\!], \delta_t)$.*

Proof. We first show that δ_t is a tree decomposition of $[\![t]\!]$ of width at most $w - 1$. The fact that the width of δ_t is at most $w - 1$ is immediate by the consistency of t. We verify that δ_t satisfies the properties of a tree decomposition of $[\![t]\!]$.

The fact that each element from $\mathsf{dom}([\![t]\!])$ is contained in some bag is immediate. Suppose that $[\![t]\!] \models R([v_1]_{a_1}, \ldots, [v_n]_{a_n})$ for some $R/n \in \mathbf{S}$. By definition, there is a $v \in \mathsf{dom}(t)$ such that $[v]_{a_i} = [v_i]_{a_i}$ for all $i = 1, \ldots, n$ and $R_{a_1,\ldots,a_n} \in t(v)$. By consistency $\{a_1, \ldots, a_n\} \subseteq \mathsf{names}(v)$ and, hence, $\{[v_1]_{a_1}, \ldots, [v_n]_{a_n}\} = \{[v]_{a_1}, \ldots, [v]_{a_n}\} \subseteq X_v$.

It remains to show that δ_t is connected. Let $v, w \in \mathsf{dom}(t)$ be nodes such that $[u]_a \in X_v \cap X_w$ for some $u \in \mathsf{dom}(t)$ and some $a \in U_{\mathbf{S},w}$. Hence, $[u]_a = [v]_a = [w]_a$ and so v and w are a-connected. For any node v' on the shortest path between v and w in $\mathcal{T}$ we thus must have $[u]_a \in X_{v'}$, which proves that δ_t is indeed connected.

It is readily seen by the definition of η_t that η_t is indeed an adornment of $([\![t]\!], \delta_t)$. □

Remark 4.38. Notice that we do *not* require that consistency of a $\Gamma_{\mathbf{S},w}$-labeled tree t means that $[\![t]\!]$ is actually acyclic. For our purposes, the fact that the tree-width of $[\![t]\!]$ is at most $w - 1$ is sufficient.

We also remark that we will often view $[\![t]\!]$ as a database, i.e., we may treat the elements of $\mathsf{dom}([\![t]\!])$ as constants whenever clear from context.

Lemma 4.39. *Suppose $\mathfrak{A}$ is an $\mathbf{S}$-structure of tree-width at most $w - 1$ and δ a tree decomposition witnessing this. For every adornment η of $(\mathfrak{A}, \delta)$, it holds that $[\![t_{\mathfrak{A},\delta,\eta}]\!]$ is weakly isomorphic to $\mathfrak{A}$.*

PROOF HINT. Let $\delta = (\mathcal{T}, (X_v)_{v \in T})$ be a tree decomposition of $\mathfrak{A}$ that has width at most $w - 1$. Let $f \colon \mathsf{dom}(\mathfrak{A}) \to U_{\mathbf{S},w}$ be the function used to assign names to elements from $\mathsf{dom}(\mathfrak{A})$. It is easy to show that the function $\chi \colon a \mapsto [\hat{v}]_{f(a)}$, where $v \in T$ is such that $a \in X_v$, is well-defined and indeed a weak isomorphism from $\mathfrak{A}$ to $t_{\mathfrak{A},\delta,\eta}$. □

COROLLARY 4.40. *Suppose* $Q_1 = (\mathbf{S}, \mathcal{O}_1, G_1(\bar{x}))$ *and* $Q_2 = (\mathbf{S}, \mathcal{O}_2, G_2(\bar{x}))$ *are OMQs from* $(\mathsf{G}, \mathsf{AQ})$. *Then the following are equivalent:*

(i) $Q_1 \subseteq Q_2$.

(ii) *For every consistent* $\Gamma_{\mathbf{S}}$*-labeled tree* t, $Q_1([\![t]\!]) \subseteq Q_2([\![t]\!])$.

PROOF. This is immediate by Theorem 4.34, Lemma 4.39, and the fact that every acyclic $\mathbf{S}$-structure can be encoded into a consistent $\Gamma_{\mathbf{S}}$-labeled tree, since acyclic $\mathbf{S}$-databases have tree-width at most $\max\{0, \mathsf{wd}(\mathbf{S}) - 1\}$. □

Devising Automata

We now proceed with our automata-based procedure, and we will, as explained in the introductory part of this section, focus on the problem $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$, i.e., deciding containment of guarded OMQs that have a Boolean atomic query as its query component (thus, a single 0-ary predicate).

We use two-way alternating parity automata that run on finite labeled amorphous trees ($\updownarrow$-2APTA, see Section 2.4). Our goal is to reduce $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$ to the emptiness problem for 2APTA. Recall that, given a 2APTA $\mathcal{A}$, we denote by $\mathcal{L}(\mathcal{A})$ the *language* of $\mathcal{A}$, i.e., the set of labeled trees over its input alphabet that it accepts. Recall also that the emptiness problem for is the problem of deciding, given a 2APTA $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) = \varnothing$ holds. Thus, given $Q_1, Q_2 \in (\mathsf{G}, \mathsf{BAQ})$, we need to construct a $\updownarrow$-2APTA $\mathcal{A}$ such that $Q_1 \subseteq Q_2$ iff $\mathcal{L}(\mathcal{A}) = \varnothing$. We have already seen in Section 2.4 that deciding whether $\mathcal{L}(\mathcal{A})$ is empty is feasible in exponential time in the number of states, and in polynomial time in the size of the input alphabet (cf. Theorem 2.29). Therefore, in order to obtain the desired 2EXPTIME upper bound, we must construct $\mathcal{A}$ in doubly exponential time, while the number of states must be at most exponential.

We first need a way to check consistency of labeled trees. The construction of an automaton for this task is fairly standard in the literature on automata for guarded logics (see, e.g., [28, 31] for similar automata).

LEMMA 4.41. *For any schema* $\mathbf{S}$, *there exists a* $\updownarrow$*-2APTA* $\mathcal{C}_{\mathbf{S},w}$ *that accepts a* $\Gamma_{\mathbf{S},w}$*-labeled tree* t *iff* t *is consistent. The number of states of* $\mathcal{C}_{\mathbf{S},w}$ *is constant, and* $\mathcal{C}_{\mathbf{S},w}$ *can be constructed in linear time in the size of* $\Gamma_{\mathbf{S},w}$.

PROOF SKETCH. Devising $\mathcal{C}_{\mathbf{S}}$ is pretty straightforward. We need to check two conditions. First, we need to ensure that no label of the input tree carries more than w constants. This can clearly be done by a single top-down pass along the tree with constantly many states. Second, we need to ensure that whenever $R_{\bar{a}}$ is a label of a node v, then also $[\bar{a}] \subseteq \mathsf{names}(v)$. This can as well be done by a top-down pass using only a constant number of states.

The construction time of $\mathcal{C}_{\mathbf{S}}$ is thus dominated by the size of the input alphabet $\Gamma_{\mathbf{S}}$, and we can write down the transition function in linear time in the size of $\Gamma_{\mathbf{S}}$. □

Now the crucial task is, given an OMQ $Q \in (\mathsf{G}, \mathsf{BAQ})$, to devise an automaton that accepts labeled trees which correspond to databases that make Q true:

Lemma 4.42. *Consider an OMQ $Q = (\mathbf{S}, \mathcal{O}, G) \in (\mathsf{G}, \mathsf{BAQ})$. There is a $\updownarrow$-2APTA $\mathcal{A}_Q$ such that, for every consistent $\Gamma_{\mathbf{S}}$-labeled tree t,*

$$t \in \mathcal{L}(\mathcal{A}_Q) \iff [\![t]\!] \models Q.$$

The number of states of $\mathcal{A}_Q$ is exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and linear in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. Moreover, $\mathcal{A}_Q$ can be constructed in doubly exponential time in the size of Q such that the second exponent of the runtime of this construction depends only on $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$.

Before providing a proof of Lemma 4.42, let us show how to prove Theorem 4.17 by employing Lemma 4.42:

Proof of Theorem 4.17. Fix two OMQs $Q_1 = (\mathbf{S}, \mathcal{O}_1, G_1)$, $Q_2 = (\mathbf{S}, \mathcal{O}_2, G_2)$ belonging to $(\mathsf{G}, \mathsf{BAQ})$. Let $w \coloneqq \mathsf{wd}(\mathbf{S})$ in case $\mathsf{wd}(\mathbf{S}) > 0$, and $w \coloneqq 1$ otherwise. We construct the automata (i) $\mathcal{C}_{\mathbf{S},w}$ from Lemma 4.41, (ii) $\mathcal{A}_{Q_1}$ and $\mathcal{A}_{Q_2}$ from Lemma 4.42. Let

$$\mathcal{A} \coloneqq \mathcal{C}_{\mathbf{S},w} \cap \mathcal{A}_{Q_1} \cap \overline{\mathcal{A}}_{Q_2}.$$

By Lemma 4.41, $\mathcal{A}$ accepts a $\Gamma_{\mathbf{S}}$-labeled input tree t iff (i) t is consistent, (ii) $t \in \mathcal{L}(\mathcal{A}_{Q_1})$, and (iii) $t \notin \mathcal{L}(\mathcal{A}_{Q_2})$. By Lemma 4.42, this means that $t \in \mathcal{L}(\mathcal{A})$ iff t is consistent and $[\![t]\!] \models Q_1$, but $[\![t]\!] \not\models Q_2$, whence Corollary 4.40 establishes that

$$\mathcal{L}(\mathcal{A}) = \varnothing \iff Q_1 \subseteq Q_2.$$

Thus, we can simply check whether $\mathcal{L}(\mathcal{A})$ is empty in order to decide $Q_1 \subseteq Q_2$. By Theorem 2.29 and Lemmas 4.41 and 4.42, we can thus perform the check $Q_1 \subseteq Q_2$ in 2ExpTime in general, and in ExpTime in case we assume that the maximum arity of the symbols used in Q_1 and Q_2 is bounded. □

It thus remains to be shown that Lemma 4.42 holds. Before doing so, we need one additional technical notion.

Derivation trees Let $Q = (\mathbf{S}, \mathcal{O}, G(\bar{x}))$ be an OMQ from $(\mathsf{G}, \mathsf{AQ})$, and let $\mathfrak{D}$ be an $\mathbf{S}$-database. A *derivation tree for $G(\bar{a})$ with respect to $\mathfrak{D}$ and Q* is a finite labeled tree $\mathcal{T}$, with $\mu_{\mathcal{T}}$ being the node labeling function that assigns a fact $R(\bar{a})$ to each node of $\mathcal{T}$, where $R \in \mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ and $[\bar{a}] \subseteq \mathsf{adom}(\mathfrak{D})$, such that the following conditions are satisfied:

(i) For the root node v of $\mathcal{T}$ we have that $\mu_{\mathcal{T}}(v) = G(\bar{a})$.

(ii) For each leaf node v of $\mathcal{T}$, we have $\mathfrak{D} \models \mu_{\mathcal{T}}(v)$ or $\mathcal{O} \models \mu_{\mathcal{T}}(v)$.

(iii) For each non-leaf node v of $\mathcal{T}$, with $u_1, \ldots, u_k$ being its children, we have that:

  (a) $\{\mu_{\mathcal{T}}(u_1), \ldots, \mu_{\mathcal{T}}(u_k)\}$ is *guarded*, i.e., it has an atom that contains all the terms of $\mathsf{adom}(\{\mu_{\mathcal{T}}(u_1), \ldots, \mu_{\mathcal{T}}(u_k)\})$, and

(b) $(\{\mu_{\mathcal{T}}(u_1), \ldots, \mu_{\mathcal{T}}(u_k)\}, \mathcal{O}) \models \mu_{\mathcal{T}}(v)$.

Intuitively, a derivation tree for $G(\bar{a})$ with respect to $\mathfrak{D}$ and Q specifies how to derive the fact $G(\bar{a})$ from $\mathfrak{D}$ and $\mathcal{O}$ by only using facts with arguments from $\mathsf{adom}(\mathfrak{D})$.

Remark 4.43. The case $\mathcal{O} \models \mu_{\mathcal{T}}(v)$ in the second item above is required to accommodate certain cases where we use $\top$ in the body of TGDs. Indeed, in our setting, there are OMQs that have a positive answer on the empty database, e.g., $Q = (\varnothing, \{\top \to G\}, G)$.

LEMMA 4.44. *For any OMQ* $Q = (\mathbf{S}, \mathcal{O}, G(\bar{x}))$ *from* $(\mathsf{G}, \mathsf{AQ})$ *and an* $\mathbf{S}$*-database* $\mathfrak{D}$*, it holds that* $\mathfrak{D} \models Q(\bar{a})$ *iff there exists a derivation tree for* $G(\bar{a})$ *with respect to* $\mathfrak{D}$ *and* Q*.*

PROOF. The direction from right to left is fairly standard, and we leave the details to the reader.

Suppose therefore that $\mathfrak{D} \models Q$. In [17] it is shown that Q can be rewritten into an equivalent guarded Datalog query whose rules only use symbols from $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$. Recall that a guarded Datalog rule is just a Datalog rule that is guarded, i.e., it has an atom in its body that has all the body variables as arguments (we allow that the body is empty in the sense that it is equivalent to $\top$). Accordingly, a guarded Datalog query is a Datalog query in which every rule is guarded. More precisely, there is a Datalog query $Q' = (\Pi, G)$ such that the extensional schema of Q' matches $\mathbf{S}$, and $Q' \equiv Q$. In fact, it is shown that we can take for Π the set[4]

$$\Pi := \{\tau \mid \mathcal{O} \models \tau, \text{ where } \tau \text{ is a guarded Datalog rule using symbols from } \mathbf{S} \cup \mathsf{sig}(\mathcal{O})\}.$$

An easy argument shows that one can assume that Π is finite, since there are, modulo renaming variables, only finitely many guarded Datalog rules that solely use symbols from $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$.

We can construct an appropriate derivation tree $\mathcal{T}$ for $G(\bar{a})$ w.r.t. $\mathfrak{D}$ and Q by employing the Datalog query Q'. Since $\mathfrak{D} \models Q(\bar{a})$, also $\mathfrak{D} \models Q'(\bar{a})$. Hence, there is an $\ell \geq 0$ such that $T^{\ell}_{Q'}(\mathfrak{D}) = T^{n}_{Q'}(\mathfrak{D})$ for all $n \geq \ell$. We are going to show that, for all $\alpha \in T^{\ell}_{Q'}(\mathfrak{D})$, there exists a derivation tree $\mathcal{T}_\alpha$ for α w.r.t. $\mathfrak{D}$ and Q'. We will then show that $\mathcal{T}_\alpha$ is also a derivation tree for α w.r.t. $\mathfrak{D}$ and Q, whence for the case $\alpha = G(\bar{a})$ the claim follows.

We proceed by induction on ℓ. Suppose first that $\ell = 0$, and let $\alpha \in T^{\ell}_{Q'}(\mathfrak{D}) = \mathfrak{D}$. Since $\mathfrak{D} \models \alpha$ we can take $\mathcal{T}_\alpha$ to be the tree with a single node whose label via the labeling function $\mu_{\mathcal{T}_\alpha}$ is α.

Assume now that $\ell > 0$ and let $\alpha \in T^{\ell}_{Q'}$. Recall that

$$T^{\ell}_{Q'}(\mathfrak{D}) := T_{Q'}(T^{\ell-1}_{Q'}(\mathfrak{D})),$$

where $T_{Q'}(\cdot)$ is the one-step operator for Q'. Let $\tau \in \Pi$ be a rule with head α and body atoms $\beta_1, \ldots, \beta_k$ such that $T^{\ell-1}_{Q'}(\mathfrak{D}) \models \beta_1 \wedge \cdots \wedge \beta_k$. By induction hypothesis, there are derivation trees $\mathcal{T}_{\beta_1}, \ldots, \mathcal{T}_{\beta_k}$ respectively for the atoms $\beta_1, \ldots, \beta_k$. We construct $\mathcal{T}_\alpha$ as the tree that has a root node labeled with α, and whose immediate subtrees are the trees

[4]Recall that we can view each Datalog rule as a logical sentence; see Subsection 2.3.3

$\mathcal{T}_{\beta_1}, \dots, \mathcal{T}_{\beta_k}$. It is clear that $\mathcal{T}_\alpha$ is a derivation tree for α w.r.t. $\mathfrak{D}$ and Q'. Notice that this also holds in case $k = 0$, since $\Pi \models \alpha$ in this case. This completes the induction step.

Now $\mathcal{T}_\alpha$ is easily seen to be a derivation tree for α w.r.t. $\mathfrak{D}$ and Q. Consider any node v of $\mathcal{T}_\alpha$, and suppose that either $\Pi \models \mu_\mathcal{T}(v)$ or v has children $v_1, \dots, v_k$ such that $(\{\mu_\mathcal{T}(v_1), \dots, \mu_\mathcal{T}(v_k)\}, \Pi) \models \mu_\mathcal{T}(v)$. Let τ be the rule from Π that has been used to derive $\mu_\mathcal{T}(v)$. By definition of Π, we have $\mathcal{O} \models \tau$, whence $(\{\mu_\mathcal{T}(v_1), \dots, \mu_\mathcal{T}(v_k)\}, \mathcal{O}) \models \mu_\mathcal{T}(v)$ follows (also for the case $k = 0$). Thus, $\mathcal{T}_\alpha$ is indeed a derivation tree for α with respect to $\mathfrak{D}$ and Q. □

We are now ready to prove Lemma 4.42.

Proof of Lemma 4.42. Let $Q = (\mathbf{S}, \mathcal{O}, G)$ be an OMQ from $(\mathsf{G}, \mathsf{BAQ})$. Consider a consistent $\Gamma_\mathbf{S}$-labeled tree t. We are going to devise a $\updownarrow$-2APTA $\mathcal{A}_Q$ that runs on $\Gamma_\mathbf{S}$-labeled input trees such that Eve has a winning strategy in $\mathcal{G}(\mathcal{A}_Q, t)$ if and only if there is a derivation tree $\mathcal{T}$ for G w.r.t. $[\![t]\!]$ and Q. By Lemma 4.44, this entails that $\mathcal{A}_Q$ accepts t if and only if $[\![t]\!] \models Q$.

Let $\mathcal{A}_Q = (S, \Gamma_\mathbf{S}, \{0, \updownarrow\}, \delta, \Omega, s_0)$. The remaining components of $\mathcal{A}_Q$ are specified in the following.

- *The state set S*: Let $U_\mathbf{S}$ be the finite set of names that is used for arguments in $\Gamma_\mathbf{S}$. The state set S consists of all atomic formulas $R(a_1, \dots, a_k)$, where $R/k \in \mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ and $a_1, \dots, a_k \in U_\mathbf{S}$. We set the initial state s_0 to be equal to G.
- *The priority function Ω*: We set $\Omega(s) \coloneqq 1$ for all $s \in S$. That is, only finite plays are winning for Eve.
- *The transition function δ*: We define δ as follows. Consider a symbol $\rho \in \Gamma_\mathbf{S}$, and let $R(a_1, \dots, a_k)$ be a state from S. We set $\bar{a} \coloneqq a_1, \dots, a_k$ and distinguish cases.

  (C1) If $\{a_1, \dots, a_k\} \not\subseteq \mathsf{names}(\rho)$, then we set $\delta(R(\bar{a}), \rho) \coloneqq \mathsf{false}$.

  In this case, Eve immediately loses the game, since she cannot ensure that the atom $R(a_1, \dots, a_k)$ has the same meaning in the current node as in the node she came from (the names $a_1, \dots, a_k$ may denote different elements of $[\![t]\!]$ in subsequent moves).

  (C2) Otherwise, if $R(\bar{a}) \in \rho$ then $\delta(R(\bar{a}), \rho) \coloneqq \mathsf{true}$.

  In this case, Eve immediately wins, since she can prove that $R(\bar{a})$ names a database fact.

  (C3) Otherwise, let

  $$\tau_1 \coloneqq \alpha_{1,1} \wedge \dots \wedge \alpha_{1,m_1}, \quad \dots, \quad \tau_l \coloneqq \alpha_{l,1} \wedge \dots \wedge \alpha_{l,m_l}$$

  be an enumeration of all (possibly empty) conjunctions of atoms present in S such that, for all $i = 1, \dots, l$,

  (i) $\{\alpha_{i,1}, \dots, \alpha_{i,m_i}\}$ is guarded with $\mathsf{adom}(\{\alpha_{i,1}, \dots, \alpha_{i,m_i}\}) \subseteq \mathsf{names}(\rho)$, and

  (ii) $(\{\alpha_{i,1}, \dots, \alpha_{i,m_i}\}, \mathcal{O}) \models R(\bar{a})$.

We let

$$\delta(R(\bar{a}), \rho) \coloneqq \langle\updownarrow\rangle R(\bar{a}) \vee \bigvee_{i=1}^{l} \bigwedge_{j=1}^{m_i} \langle 0\rangle \alpha_{i,j}.$$

Eve may choose between two possibilities here. Either she moves to some neighboring node in the tree while remaining in state $R(\bar{a})$, or she may decide to pick a guarded conjunction $\tau_i := \alpha_{i,1} \wedge \cdots \wedge \alpha_{i,m_i}$. In the latter case, Adam challenges Eve's choice by changing the state to one of the $\alpha_{i,j}$. This case thus amounts to the construction of a derivation tree, and Eve's claim in the second possibility is that the atoms named by $\alpha_{i,1}, \ldots, \alpha_{i,m_i}$ are feasible successors for the atom named by $R(\bar{a})$ in a derivation tree.

This concludes the construction of $\mathcal{A}_Q$. It is rather tedious, but not very interesting from a technical point of view, to show that the behavior of $\mathcal{A}$ is correct in the sense that $\mathcal{A}_Q$ accepts t iff there is a derivation tree for G w.r.t. $[\![t]\!]$ and Q. We leave this proof as an exercise to the reader.

Let us briefly comment on the size of $\mathcal{A}_Q$ and the time needed to construct it. It is clear that the number of states of $\mathcal{A}_Q$ is exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and linear in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. Moreover, the overall construction of $\mathcal{A}_Q$ takes doubly exponential time in the size of Q. The determining factor for this upper bound is the construction of $\delta(\cdot, \cdot)$ – more specifically, the case of item (C3). Modulo renaming variables, there are at most doubly exponentially many conjunctions of the form $\tau_i = \alpha_{i,1} \wedge \cdots \wedge \alpha_{i,m_i}$ that imply a given atomic fact under $\mathcal{O}$, and τ_i is of at most exponential size. Assuming relation symbols of bounded arity, the number of all τ_i modulo renaming variables is at most exponential. Moreover, checking whether an atomic fact is implied by a database and a set of guarded rules is feasible in 2ExpTime in combined complexity, and in PTime in data complexity (see Subsection 3.4.2). Furthermore, the combined complexity drops to ExpTime once we assume schemas of bounded arity. The transition function can therefore be constructed in 2ExpTime in the general case, and in ExpTime when we assume that Q is formulated over a schema of bounded width. □

4.3.2 From Conjunctive Queries to Atomic Queries

As announced in the introductory part of this section, we will now show how to use Theorem 4.17 in order to establish Theorem 4.16. For obtaining the upper bounds mentioned in Theorem 4.16, we need to show that

(i) $\mathsf{Cont}(\mathsf{FG}, \mathsf{UCQ})$ is in 2ExpTime, and

(ii) $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ})$ is in ExpTime, assuming schemas of bounded width.

The following lemma shows that we can focus on the variants of these problems that only consider Boolean atomic queries:

Lemma 4.45. (i) $\mathsf{Cont}(\mathsf{FG}, \mathsf{UCQ})$ *can be polynomially reduced to* $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$.

(ii) $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ})$ *can be polynomially reduced to* $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$.

Proof. We first prove item (i). Let

$$Q_1 = (\mathbf{S}, \mathcal{O}_1, q_1(x_1, \ldots, x_n)) \quad \text{and} \quad Q_2 = (\mathbf{S}, \mathcal{O}_2, q_2(x_1, \ldots, x_n))$$

be OMQs from $(\mathsf{FG}, \mathsf{UCQ})$, where

$$q_1(x_1, \ldots, x_n) = \bigvee_{i=1}^{k_1} q_{1,i}(x_1, \ldots, x_n), \quad q_2(x_1, \ldots, x_n) = \bigvee_{i=1}^{k_2} q_{2,i}(x_1, \ldots, x_n),$$

for some CQs $q_{1,i}(x_1, \ldots, x_n)$ and $q_{2,i}(x_1, \ldots, x_n)$. Let $A_1, \ldots, A_n$ be fresh unary predicates that do not occur in $\mathbf{S} \cup \mathsf{sig}(\mathcal{O}_1) \cup \mathsf{sig}(\mathcal{O}_2)$, and let $\mathbf{S}' \coloneqq \mathbf{S} \cup \{A_1, \ldots, A_n\}$. For $i \in \{1, 2\}$ and $j = 1, \ldots, k_i$, let $q'_{i,j}$ be the BCQ that results from $q_{i,j}(x_1, \ldots, x_n)$ by adding the atoms $A_1(x_1), \ldots, A_n(x_n)$ as conjuncts to its body and closing the resulting query off under existential quantifiers. Let G be a fresh 0-ary predicate that does not occur in $\mathbf{S} \cup \mathsf{sig}(\mathcal{O}_1) \cup \mathsf{sig}(\mathcal{O}_2)$. For $i \in \{1, 2\}$, we let

$$\begin{aligned} &Q'_i \coloneqq (\mathbf{S}', \mathcal{O}'_i, G), \text{where} \\ &\mathcal{O}'_i \coloneqq \mathcal{O}_i \cup \{q'_{i,j} \to G \mid j = 1, \ldots, k_i\}. \end{aligned}$$

It is not hard to check that Q'_i belongs to $(\mathsf{FG}, \mathsf{BAQ})$. We claim that $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$.

Indeed, if $Q'_1(\mathfrak{D}') \not\subseteq Q'_2(\mathfrak{D}')$ for some $\mathbf{S}'$-database $\mathfrak{D}'$, then there are $a_1, \ldots, a_n \in \mathsf{adom}(\mathfrak{D}')$ such that $\mathfrak{D}' \models A_1(a_1) \wedge \cdots \wedge A_n(a_n)$. Let $\mathfrak{D}$ be the $\mathbf{S}$-database that results from $\mathfrak{D}'$ by restricting $\mathfrak{D}'$ to facts over $\mathbf{S}$. It is easy to see that $\mathfrak{D} \models Q_1(a_1, \ldots, a_n)$ while $\mathfrak{D} \not\models Q_2(a_1, \ldots, a_n)$, whence $Q_1 \not\subseteq Q_2$ follows.

On the other hand, if $Q_1(\mathfrak{D}) \not\subseteq Q_2(\mathfrak{D})$, then $\mathfrak{D} \models Q_1(a_1, \ldots, a_n)$ for some $a_1, \ldots, a_n \in \mathsf{adom}(\mathfrak{D})$, yet $\mathfrak{D} \not\models Q_2(a_1, \ldots, a_n)$. Hence, for $\mathfrak{D}' \coloneqq \mathfrak{D} \cup \{A_1(a_1), \ldots, A_n(a_n)\}$ we have $\mathfrak{D}' \models Q'_1$ but $\mathfrak{D}' \not\models Q'_2$, whence $Q'_1 \not\subseteq Q'_2$ follows.

For item (ii) we proceed likewise. Let

$$Q_1 = (\mathbf{S}, \mathcal{O}_1, G_1(x_1, \ldots, x_n)) \quad \text{and} \quad Q_2 = (\mathbf{S}, \mathcal{O}_2, G_2(x_1, \ldots, x_n))$$

be OMQs from $(\mathsf{G}, \mathsf{AQ})$, and let $\mathbf{S}' \coloneqq \mathbf{S} \cup \{A_1, \ldots, A_n\}$, where $A_1, \ldots, A_n$ are fresh unary predicates as before. Pick a fresh 0-ary predicate G, and, for $i \in \{1, 2\}$, let

$$\begin{aligned} &Q'_i \coloneqq (\mathbf{S}', \mathcal{O}'_i, G), \text{where} \\ &\mathcal{O}'_i \coloneqq \mathcal{O}_i \cup \{G_i(x_1, \ldots, x_n), A_1(x_1), \ldots, A_n(x_n) \to G\}. \end{aligned}$$

Notice that $\mathcal{O}_i$ is indeed guarded. Then, as before, $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$. □

Notice that it follows from Lemma 4.45 and Theorem 4.17 that $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ})$ is feasible in ExpTime assuming schemas of bounded width. To establish Theorem 4.16 it therefore remains to be shown that $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$ is in 2ExpTime. To this end, we introduce some additional technical notions. These notions discussed here are mainly inspired from similar ones defined in [18, 20, 44], but differ in technical aspects.

Squid Decompositions

The following definition relaxes the notion of acyclicity:

Definition 4.46. Let $\mathfrak{A}$ be an $\mathbf{S}$-structure and $\mathfrak{C}$ be an induced substructure of $\mathfrak{A}$. We say that $\mathfrak{A}$ is a $\mathfrak{C}$-*tree* if there is a tree decomposition $\delta = (\mathcal{T}, (X_v)_{v \in T})$ of $\mathfrak{A}$ such that: (i) $\mathfrak{A}_\delta(\varepsilon) = \mathfrak{C}$, and (ii) every $v \in T \setminus \{\varepsilon\}$ is guarded.

Intuitively, if $\mathfrak{A}$ is a $\mathfrak{C}$-tree, then $\mathfrak{A}$ is acyclic, except for the substructure $\mathfrak{C}$ of $\mathfrak{A}$ which is allowed to be cyclic.

Lemma 4.47. *Suppose $\mathcal{O}$ is a set of frontier-guarded rules. For every $k \geq 0$, $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ is an $\mathfrak{F}$-tree, where $\mathfrak{F}$ is the substructure of $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ induced by $\mathsf{dom}(\mathfrak{A})$.*

Proof hint. This proof is similar to that of Lemma 4.20, but works with tree decompositions of $\mathfrak{C}$-trees rather than guarded tree decompositions. We omit it for brevity. □

Definition 4.48. Let $q(\bar{x})$ be a CQ over $\mathbf{S}$ and let $\mathbf{T} \supseteq \mathbf{S}$. A *squid decomposition of* $q(\bar{x})$ over $\mathbf{T}$ is a tuple $\theta = (q_0, p_1, \ldots, p_n)$, where

(i) q_0 is a CQ over $\mathbf{T}$, and $p_1, \ldots, p_n$ are strictly acyclic CQs over $\mathbf{T}$.

(ii) $|q_0| \leq |q|$ and $|\mathsf{var}(q_0)| \leq |\mathsf{var}(q)|$.

(iii) $|q_0| + |p_1| + \cdots + |p_n| \leq 3|q| + 4|\bar{x}|$.

(iv) $q_0 \wedge p_1 \wedge \cdots \wedge p_n$ has the same free variables as $q(\bar{x})$, and there exists a query among $q_0, p_1, \ldots, p_n$ that has exactly the variables $\bar{x}$ as answer variables.

(v) $q_0 \wedge p_1 \wedge \cdots \wedge p_n \models q(\bar{x})$.

Whenever it is unambiguous to do so, we will abuse notation and regard θ as the formula $\theta(\bar{x}) \coloneqq q_0 \wedge p_1 \wedge \cdots \wedge p_n$.

Intuitively, a squid decomposition of $q(\bar{x})$ describes a way how $q(\bar{x})$ can be homomorphically mapped to a $\mathfrak{C}$-tree. The CQ q_0 describes the cyclic part of $q(\bar{x})$, while the queries $p_1, \ldots, p_n$ represent the acyclic components. The term "squid decomposition" is taken from [44], and Definition 4.48 is inspired by a similar definition from [44], but differs in technical aspects. We can visualize θ as a "squid" whose "head" is q_0, and whose "tentacles" are given by $p_1, \ldots, p_n$. In the following, we are going to make the relationship between squid decompositions and $\mathfrak{C}$-trees more precise. Before that, let us analyze homomorphisms from queries to acyclic structures and $\mathfrak{C}$-trees more carefully.

A CQ $q(\bar{x})$ is *answer-guarded* if it contains an atom in its body that has all answer variables of $q(\bar{x})$ as arguments – such an atom is called an *answer guard* of $q(\bar{x})$, and we may speak about *the* answer guard by simply fixing one if there are multiple. Notice that every BCQ is trivially answer-guarded. Also notice that the body of any frontier-guarded rule can be seen as an answer-guarded CQ.

A result similar to the following is shown in [18], and a full proof of it can be found in Appendix B.2:

Lemma 4.49. *Suppose $\mathfrak{A}$ is an acyclic $\mathbf{T}$-structure and $q(\bar{x})$ an answer-guarded CQ over a schema $\mathbf{S} \subseteq \mathbf{T}$. Then the following are equivalent:*

(i) $\mathfrak{A} \models q(\bar{a})$.

(ii) *There is a strictly acyclic $p(\bar{x})$ over $\mathbf{T}$ that has the same answer variables as $q(\bar{x})$ such that*
   (a) $p(\bar{x}) \models q(\bar{x})$,
   (b) $\mathfrak{A} \models p(\bar{a})$, *and*
   (c) $|p| \leq 3|q| + |\bar{x}|$.

The following lemma highlights the relationship between squid decompositions and $\mathfrak{C}$-trees. A proof of it can again be found in Appendix B.2:

Lemma 4.50. *Suppose $\mathfrak{A}$ is a $\mathfrak{C}$-tree over a schema $\mathbf{T}$, and suppose $\delta = (\mathcal{T}, (X_v)_{v \in T})$ is a tree decomposition of $\mathfrak{A}$ witnessing this fact. Suppose further that $q(x_1, \ldots, x_n)$ is an answer-guarded CQ over a schema $\mathbf{S} \subseteq \mathbf{T}$. Then, for all tuples $a_1, \ldots, a_n$ over $\mathsf{dom}(\mathfrak{A})$, the following are equivalent:*

(i) $\mathfrak{A} \models q(a_1, \ldots, a_n)$.

(ii) *There is a squid decomposition $\theta = (q_0, p_1, \ldots, p_k)$ of $q(\bar{x})$ over $\mathbf{T}$ and homomorphisms $h_0, h_1, \ldots, h_k$ such that*
   (a) *h_0 is a homomorphism from q_0 to $\mathfrak{C}$.*
   (b) *For $i = 1, \ldots, k$, h_i is a homomorphism from p_i to $\bigcup_{v \in T \setminus \{\varepsilon\}} \mathfrak{A}_\delta(v)$.*
   (c) *For $i = 1, \ldots, n$ and $s, r \in \{0, 1, \ldots, k\}$, if $x_i \in \mathsf{dom}(h_s) \cap \mathsf{dom}(h_r)$, then $h_s(x_i) = h_r(x_i) = a_i$.*

Remark 4.51. We remark that the bounds on the combined number of atoms of the queries among $q_0, p_1, \ldots, p_k$ can be improved by performing a deeper analysis concerning the construction of the p_i. However, we prefer here to keep the analysis simple, and the stated bounds suffice for our purposes.

Treeification

We now proceed to present the reduction from $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$ to $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$. To this end, we aim at transforming a given $Q \in (\mathsf{FG}, \mathsf{BAQ})$ into a $Q' \in (\mathsf{G}, \mathsf{BAQ})$ so that Q and Q' are equivalent over acyclic databases, though they may not be equivalent over the class of all databases. It will turn out, given two $Q_1, Q_2 \in (\mathsf{FG}, \mathsf{BAQ})$ and their respective translations $Q_1', Q_2' \in (\mathsf{G}, \mathsf{BAQ})$, that $Q_1 \subseteq Q_2$ iff $Q_1' \subseteq Q_2'$ (Theorem 4.58). The translation from Q to Q' introduces auxiliary relation symbols and incurs an exponential blowup in the number of rules. Hence, the translation is in general exponential. However, it turns out that we only increase the maximum arity used in Q' only linearly in maximum number of variables that occurs in a rule body of Q. This suffices to establish that $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$ is indeed in 2ExpTime.

Definition 4.52. Given an answer-guarded CQ $q(\bar{x})$ over $\mathbf{S}$ and a schema $\mathbf{S} \subseteq \mathbf{T}$, the $\mathbf{T}$-*treeification of* $q(\bar{x})$ is the set $\Lambda_q^{\mathbf{T}}$ of all strictly acyclic CQs $q'(\bar{x})$ over $\mathbf{T}$, that have the same answer variables as $q(\bar{x})$, such that (i) q' implies q, that is, $q'(\bar{x}) \models q(\bar{x})$. (ii) q' is minimal in the sense that removing one atom from q' turns q' into a CQ that is either not strictly acyclic or does not imply q anymore.

It follows from Lemma 4.49 that all the CQs of $\Lambda_q^{\mathbf{T}}$ can be restricted as to contain only CQs of size at most $3|q| + |\bar{x}|$. Indeed, if one CQ in $\Lambda_q^{\mathbf{T}}$ were bigger than $3|q| + |\bar{x}|$, then Lemma 4.49 tells us that we can find one of size $3|q| + |\bar{x}|$, since we can view a CQ as a structure and vice versa. It is also not too hard to check that $\Lambda_q^{\mathbf{T}}$ is consists thus of at most exponentially many CQs each of whose size is linear in the size of q (cf. [18]). Hence, $\Lambda_q^{\mathbf{T}}$ can be seen as a UCQ that is of exponential size in the size of q, and we will often consider $\Lambda_q^{\mathbf{T}}$ as such a UCQ.

Notice that $q(\bar{x})$ is in general not equivalent to its treeification. However, $q(\bar{x})$ and $\Lambda_q^{\mathbf{T}}$ are equivalent over acyclic $\mathbf{T}$-structures [20]. The following lemma is an immediate consequence of Lemma 4.49 by considering that we can view CQs as structures:

Lemma 4.53. *For any acyclic* $\mathbf{T}$*-structure* $\mathfrak{A}$ *and any tuple* $\bar{a}$*,*

$$\mathfrak{A} \models q(\bar{a}) \iff \mathfrak{A} \models \Lambda_q^{\mathbf{T}}(\bar{a}).$$

Treeifying OMQs from (FG, BAQ) Let $Q = (\mathbf{S}, \mathcal{O}, G)$ from $(\mathsf{FG}, \mathsf{BAQ})$. The *width* of $\mathcal{O}$, denoted $\mathsf{wd}(\mathcal{O})$, is the maximum number of variables that appear in any body of a rule from $\mathcal{O}$. Fix a new relation symbol C of arity $\mathsf{wd}(\mathcal{O})$.

We are now going to describe a translation $\eta_C(Q)$ that takes Q and transforms it into an OMQ $\eta_C(Q)$ from $(\mathsf{G}, \mathsf{BAQ})$ with data schema $\mathbf{S} \cup \{C\}$. Firstly, we set

$$\eta_C(Q) \coloneqq \left(\mathbf{S} \cup \{C\}, \bigcup_{\tau \in \mathcal{O}} \eta_C^{\mathbf{S} \cup \mathsf{sig}(\mathcal{O})}(\tau), G \right),$$

where the definition of $\eta_C^{\mathbf{T}}(\tau)$, for $\tau \in \mathcal{O}$ and a schema $\mathbf{T}$, is as follows. Suppose τ is of the form $\varphi(\bar{x}, \bar{z}) \to \exists \bar{y}\, \beta(\bar{x}, \bar{y})$. Then we set

$$f_C^{\mathbf{T}}(\tau) \coloneqq \left\{ q(\bar{x}) \to \exists \bar{y}\, \beta(\bar{x}, \bar{y}) \mid q(\bar{x}) \in \Lambda_{\exists \bar{z}\, \varphi(\bar{x}, \bar{z})}^{\mathbf{T} \cup \{C\}} \right\}.$$

Notice though, strictly speaking, the rules $q(\bar{x}) \to \exists \bar{y}\, \beta(\bar{x}, \bar{y})$ may not be guarded. However, since $q(\bar{x})$ is strictly acyclic, we may unfold $q(\bar{x}) \to \exists \bar{y}\, \beta(\bar{x}, \bar{y})$ into linearly many guarded rules by using additional auxiliary predicates. The result of this unfolding will be denoted $\eta_C^{\mathbf{T}}(\tau)$.

We are going to describe this unfolding step in more detail in the following. Let $\chi(\bar{x})$ be a strictly guarded formula equivalent to $q(\bar{x})$ (cf. Lemma 4.24). During the unfolding step, we are going to introduce fresh auxiliary predicates of the form T_η/k, where η is a subformula of $\varphi(\bar{x})$ and k is the number of free variables of η. We shall treat these predicates modulo logical equivalence, i.e., we set $T_{\eta_1} = T_{\eta_2}$ iff $\eta_1 \equiv \eta_2$. We unfold the query $q(\bar{x})$ inductively according to the construction of $\chi(\bar{x})$.

- Suppose first that $\chi(\bar{x}) \equiv \alpha(\bar{x})$ for some relational atom $\alpha(\bar{x})$. We translate $q(\bar{x})$ to the rule

$$\alpha(\bar{x}) \rightarrow T_{\chi(\bar{x})}(\bar{x}).$$

- Suppose now that $\chi(\bar{x}) \equiv \exists \bar{y}\,(\gamma(\bar{x},\bar{y}) \wedge \eta)$, where $\gamma(\bar{x},\bar{y})$ is an atomic formula guarding η with free variables as indicated. Let $\eta_1(\bar{x}_1),\dots,\eta_k(\bar{x}_k)$ be strictly guarded formulas with free variables as indicated and whose conjunction is equivalent to η. Then we rewrite $q(\bar{x}) \rightarrow \exists \bar{y}\,\beta(\bar{x},\bar{y})$ into the rule

$$\gamma(\bar{x},\bar{y}), T_{\eta_1}(\bar{x}_1),\dots,T_{\eta_k}(\bar{x}_k) \rightarrow T_{\chi(\bar{x})}(\bar{x}),$$

  and, in addition, add the according translations for the formulas $\eta_1(\bar{x}_1),\dots,\eta_k(\bar{x}_k)$.

The *unfolding*[5] of the rule $q(\bar{x}) \rightarrow \exists \bar{y}\,\beta(\bar{x},\bar{y})$ is then the set of rules resulting from translating $\chi(\bar{x})$ plus the rule

$$T_{\chi(\bar{x})} \rightarrow \exists \bar{y}\,\beta(\bar{x},\bar{y}).$$

As mentioned above, we set

$$\eta_C^{\mathbf{T}}(\tau) \coloneqq \{\sigma \mid \sigma \text{ is a rule occurring in the unfolding of } \tau\}.$$

It is easy to see that the unfolding introduces at most linearly many new auxiliary predicates per rule. Thus, the total number of rules in $\bigcup_{\tau\in\mathcal{O}} \eta_C^{\mathbf{S}\cup\mathsf{sig}(\mathcal{O})}(\tau)$ is exponential in the number of rules of $\mathcal{O}$, and we thus may introduce an exponential number of new auxiliary relation symbols. The OMQ $\eta_C(Q)$ is called the *C-treeification of Q*.

Example 4.54. We provide an example that that illustrates the unfolding of rules as described above. Suppose $q(x) = \exists y, z\,(R(x,y) \wedge S(x,x) \wedge R(y,z))$ which is equivalent to the strictly guarded formula $\chi(x) \coloneqq S(x,x) \wedge \exists y\,(R(x,y) \wedge \exists z\,R(y,z))$. Suppose we want to unfold the frontier-guarded rule $q(x) \rightarrow O(x)$, where $O/1$ is a unary relation symbol. Then the unfolding described above yields the set of rules

$$\begin{aligned}
T_{\chi(x)}(x) &\rightarrow O(x),\\
S(x,x), T_{\exists y(R(x,y)\wedge\exists z R(y,z))}(x) &\rightarrow T_{\chi(x)}(x),\\
R(x,y), T_{\exists z R(y,z)}(y) &\rightarrow T_{\exists y(R(x,y)\wedge\exists z R(x,z))}(x),\\
R(y,z) &\rightarrow T_{\exists z R(y,z)}(y).
\end{aligned}$$

Notice that here we did not follow the exact translation, since we treat $\exists z\,R(y,z)$ as a strictly guarded formula, thereby invoking the fact that it is equivalent to the formula $\exists z\,(R(y,z) \wedge R(y,z))$. ⊣

[5] Of course, the notion of unfolding depends on the choice of $\chi(\bar{x})$. However, it is easily seen that we arrive at an equivalent set of rules, no matter which $\chi(\bar{x})$ equivalent to $q(\bar{x})$ is chosen.

For the following results, let us fix an OMQ $Q = (\mathbf{S}, \mathcal{O}, G)$ from $(\mathsf{FG}, \mathsf{BAQ})$, and a relation symbol C of arity *at least* $\mathsf{wd}(\mathcal{O})$ not occurring in Q. In the following, let $\mathbf{T} \coloneqq \mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ and let us write $\mathcal{O}^+$ for $\bigcup_{\tau \in \mathcal{O}} \eta_C^{\mathbf{T}}(\tau)$.

The following lemma lifts Lemma 4.53 to the case of OMQs from $(\mathsf{FG}, \mathsf{BAQ})$:

LEMMA 4.55. *For every acyclic* $(\mathbf{S} \cup \{C\})$*-structure* $\mathfrak{A}$ *it holds that*

$$\mathfrak{A} \restriction \mathbf{S} \models Q \iff \mathfrak{A} \models \eta_C(Q).$$

PROOF. Suppose first that $\mathfrak{A} \restriction \mathbf{S} \models Q$. Hence, $\mathsf{chase}(\mathfrak{A} \restriction \mathbf{S}, \mathcal{O}) \models G$, which entails that $\mathsf{chase}(\mathfrak{A}, \mathcal{O}) \models G$. Since $\mathfrak{A}$ is acyclic, Lemma 4.20 tells us that, for every $k \geq 0$, $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ is as well. We prove by induction on k that, for all $k \geq 0$, there is a homomorphism h_k from $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+)$ such that (i) h_{k+1} extends h_k for all $k \geq 0$, and (ii) $h \coloneqq \bigcup_{k \geq 0} h_k$ is a homomorphism from $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+)$.

The base case of this induction is trivial, as we can simply chose h_0 to be the identity on $\mathsf{dom}(\mathfrak{A})$. Consider now $\mathsf{chase}^{k+1}(\mathfrak{A}, \mathcal{O})$ and let $\beta(\bar{t}, \bar{\lambda})$ be the atom derived in the $(k+1)$-st chase step using a rule $\tau\colon q(\bar{x}) \to \exists \bar{y}\, \beta(\bar{x}, \bar{y})$ from $\mathcal{O}$ and a homomorphism h' such that $h'(\bar{x}) = \bar{t}$. Assume that $\bar{\lambda}$ is a sequence of labeled nulls that only appear in $\beta(\bar{t}, \bar{\lambda})$ and that do not appear in $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$. Since $\mathcal{O}$ is frontier-guarded, the variables $\bar{x}$ are the frontier-variables of τ, and hence $\bar{t}$ is a guarded tuple of $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$. From $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O}) \models q(\bar{t})$ and the fact that $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ is acyclic, Lemma 4.53 gives us $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O}) \models \Lambda_q^{\mathbf{T} \cup \{C\}}(\bar{t})$, that is, $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O}) \models p(\bar{t})$ for some $p(\bar{x}) \in \Lambda_q^{\mathbf{T} \cup \{C\}}$. By induction hypothesis, h_k is a homomorphism from $\mathsf{chase}^k(\mathfrak{A}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+)$, whence $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+) \models p(h_k(\bar{t}))$ follows. An easy subsidiary induction on the structure of the strictly guarded $\chi(\bar{x}) \equiv p(\bar{x})$ shows that $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+) \models p(h_k(\bar{t}))$ entails $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+) \models T_{\chi(\bar{x})}(h_k(\bar{t}))$ as well. Recall that $T_{\chi(\bar{x})}(\bar{x}) \to \exists \bar{y}\, \beta(\bar{x}, \bar{y})$ is a rule of $\mathcal{O}^+$, and thus we can extend h_k to a mapping h_{k+1} that maps the nulls $\bar{\lambda}$ to appropriate nulls in $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+)$ – these nulls must exist, since $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+) \models T_{\chi(\bar{x})}(h_k(\bar{t}))$ – so that h_{k+1} is a homomorphism from $\mathsf{chase}^{k+1}(\mathfrak{A}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+)$. This concludes the induction step. It is clear that $h \coloneqq \bigcup_{k \geq 0} h_k$ is a homomorphism from $\mathsf{chase}(\mathfrak{A}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{A}, \mathcal{O}^+)$. Hence, $(\mathfrak{A}, \mathcal{O}^+) \models G$, which means that $\mathfrak{A} \models \eta_C(Q)$, as desired.

Suppose now that $\mathfrak{A} \models \eta_C(Q)$, i.e., $(\mathfrak{A}, \mathcal{O}^+) \models G$. We have to show that $\mathfrak{A} \restriction \mathbf{S} \models Q$, that is, $(\mathfrak{A} \restriction \mathbf{S}, \mathcal{O}) \models G$. Let $\mathfrak{B} \supseteq \mathfrak{A} \restriction \mathbf{S}$ be a model of $\mathcal{O}$. We shall first expand $\mathfrak{B}$ to a model $\mathfrak{B}^+ \supseteq \mathfrak{B}$ of $\mathcal{O}^+$. To this end, it suffices to specify the interpretation of C plus the interpretations of the relation symbols of the form $T_{\varphi(\bar{x})}$, where $\varphi(\bar{x})$ is a strictly guarded formula, introduced as new symbols in $\mathcal{O}^+$ – more precisely, this means that $\varphi(\bar{x})$ appears as a subformula of a strictly guarded formula χ, and χ is equivalent to some query occurring in the treeification of some rule body of $\mathcal{O}$. For these symbols, we let

$$\mathfrak{B}^+ \models C(\bar{b}) \iff_{df} \mathfrak{A} \models C(\bar{b}), \text{ and}$$
$$\mathfrak{B}^+ \models T_{\varphi(\bar{x})}(\bar{b}) \iff_{df} \mathfrak{B} \cup \mathfrak{A} \restriction \{C\} \models \varphi(\bar{b}),$$

for all tuples $\bar{b}$ over $\mathsf{dom}(\mathfrak{B})$. For all other relations, we specify that $\mathfrak{B}^+$ agrees with $\mathfrak{B}$. Notice that the interpretation of $T_{\varphi(\bar{x})}$ in $\mathfrak{B}^+$ is indeed well-defined, since we identify the symbols $T_{\varphi(\bar{x})}$ modulo logical equivalence.

It remains to be shown that $\mathfrak{B}^+$ is indeed a model of $\mathcal{O}^+$. An easy induction on the structure of $\varphi(\bar{x})$ shows that, if $T_{\varphi(\bar{x})}(\bar{x})$ is the head atom of a rule $\tau \in \mathcal{O}^+$, then τ is satisfied in $\mathcal{O}^+$. Now let $T_{\varphi(\bar{x})} \to \psi(\bar{x})$ be a rule of $\mathcal{O}^+$ such that $\varphi(\bar{x}) \equiv p(\bar{x})$, where $p(\bar{x}) \in \Lambda_q^{\mathbf{T} \cup \{C\}}$ for some query $q(\bar{x})$ that is the rule body of some rule in $\mathcal{O}$. Observe that if $\mathfrak{B}^+ \models T_{\varphi(\bar{x})}(\bar{b})$ for some tuple $\bar{b}$, then, by the construction of $\mathfrak{B}^+$, also $\mathfrak{B} \cup \mathfrak{A} \restriction \{C\} \models \varphi(\bar{b})$ and thus $\mathfrak{B} \cup \mathfrak{A} \restriction \{C\} \models p(\bar{b})$. Hence, $\mathfrak{B} \models q(\bar{b})$ follows due to $p(\bar{x}) \models q(\bar{x})$ and the fact that C does not occur in $q(\bar{x})$. Since $\mathfrak{B}$ is a model of $\mathcal{O}$, we obtain $\mathfrak{B} \models \psi(\bar{b})$ and so $\mathfrak{B}^+ \models \psi(\bar{b})$, as $\mathfrak{B}^+$ extends $\mathfrak{B}$ only by the interpretation of new relation symbols, of which none occurs in $\psi(\bar{x})$. Therefore, $\mathfrak{B}^+$ is a model of $\mathcal{O}$.

Since $\mathfrak{B}^+ \supseteq \mathfrak{A}$ and $\mathfrak{B}^+ \models \mathcal{O}^+$, we immediately obtain $\mathfrak{B}^+ \models G$ by the assumption $(\mathfrak{A}, \mathcal{O}^+) \models G$. Hence, also $\mathfrak{B} \models G$, and thus $(\mathfrak{A} \restriction \mathbf{S}, \mathcal{O}) \models G$ as desired. □

In the following, given an $\mathbf{S}$-database $\mathfrak{D}$, we write $\mathfrak{D}_C$ for the $(\mathbf{S} \cup \{C\})$-database that extends $\mathfrak{D}$ by the facts

$$\{C(a_1, \ldots, a_w) \mid a_1, \ldots, a_w \in \mathsf{adom}(\mathfrak{D})\}.$$

Lemma 4.56. *Suppose* $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O}) \models q(\bar{t})$ *for some answer-guarded CQ* $q(\bar{x})$ *over* $\mathbf{T}$ *with* $|\mathsf{var}(q)| \leq \mathsf{wd}(\mathcal{O})$. *Then there exists a* $p(\bar{x}) \in \Lambda_q^{\mathbf{T} \cup \{C\}}$ *such that* $\mathsf{chase}^k(\mathfrak{D}_C, \mathcal{O}) \models p(\bar{t})$.

Proof. Let $\bar{t} = t_1, \ldots, t_m$ and $\bar{x} = x_1, \ldots, x_m$. Since $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O}) \models q(\bar{t})$, also $\mathsf{chase}^k(\mathfrak{D}_C, \mathcal{O}) \models q(\bar{t})$. Let us write $\mathfrak{J}$ for $\mathsf{chase}^k(\mathfrak{D}_C, \mathcal{O})$ and w for the arity of C in the following – recall that $w \geq \mathsf{wd}(\mathcal{O})$. By Lemma 4.47, we know that $\mathfrak{J}$ is an $\mathfrak{F}$-tree, where $\mathfrak{F}$ is the substructure of $\mathfrak{J}$ induced by $\mathsf{dom}(\mathfrak{D}_C) = \mathsf{adom}(\mathfrak{D})$. Let $\delta = (\mathcal{T}, (X_v)_{v \in T})$ be a tree decomposition witnessing this fact – in particular $X_\varepsilon = \mathsf{dom}(\mathfrak{D})$.

By Lemma 4.50, there is a squid decomposition $\theta = (q_0, p_1, \ldots, p_n)$ of $q(\bar{x})$ over $\mathbf{T} \cup \{C\}$ and homomorphisms $h_0, h_1, \ldots, h_n$ such that:

(i) h_0 is a homomorphism from q_0 to $\mathfrak{F}$.

(ii) For $i = 1, \ldots, n$, h_i is a homomorphism from p_i to $\bigcup_{v \in T \setminus \{\varepsilon\}} \mathfrak{J}_\delta(v)$.

(iii) All the homomorphisms $h_0, h_1, \ldots, h_n$ agree on all the variables among $x_1, \ldots, x_m$ that are in their domain and if x_i is in the domain of h_j, then $h_j(x_i) = t_i$.

Let us assume w.l.o.g. that the only variables two distinct queries among $q_0, p_1, \ldots, p_n$ share are from $\bar{x}$. For $i = 1, \ldots, n$, let p_i be of the form $\exists \bar{y}_i \, \varphi_i(\bar{x}_i, \bar{y}_i)$, where φ_i is a conjunction of atoms and $[\bar{x}_i] \subseteq [\bar{x}]$.

Since θ is a squid decomposition of $q(\bar{x})$, we know that $|\mathsf{var}(q_0)| \leq |\mathsf{var}(q)| \leq w$. Let q_0^- be the CQ that results from q_0 by stripping existential quantifiers off it, and let $u_1, \ldots, u_w$ be a sequence of variables such that $\{u_1, \ldots, u_w\} = \mathsf{var}(q_0^+)$. Hence, the sequence $u_1, \ldots, u_w$ consists of w variables, possibly with repetitions, that are all members of $\mathsf{var}(q_0^-)$. Let $\bar{v}$ be a sequence of variables such that $[\bar{v}] = \{u_1, \ldots, u_w\} \setminus [\bar{x}]$. Consider the formula

$$\chi \coloneqq \exists \bar{v} \, (C(u_1, \ldots, u_w) \wedge q_0^- \wedge p_1 \wedge \cdots \wedge p_n).$$

Obviously, χ is equivalent to the CQ

$$s \coloneqq \exists \bar{v}, \bar{y}_1, \ldots, \bar{y}_n \, (C(u_1, \ldots, u_w) \wedge q_0^- \wedge \varphi_1 \wedge \cdots \wedge \varphi_n).$$

Since θ is a squid decomposition of $q(\bar{x})$, one of the $q_0, p_1, \ldots, p_n$ has exactly the variables $\bar{x}$ as answer variables, hence s has exactly the variables $\bar{x}$ as answer variables. Moreover, for $i = 1, \ldots, n$, p_i is equivalent to a strictly guarded formula χ_i. This means that χ is actually equivalent to a strictly guarded formula, whence it follows by Lemma 4.24 that χ (and thus s) is equivalent to a strictly acyclic query.

Recall that h_0 maps q_0^- to $\mathfrak{F}$ and notice that all the w-tuples over $\mathsf{dom}(\mathfrak{F}) = \mathsf{dom}(\mathfrak{D})$ are elements of the relation $C^{\mathfrak{D}_C}$. Thus, the homomorphism $h \coloneqq (h_0 \cup h_1 \cup \cdots \cup h_n)$ witnesses that $\mathfrak{J} \models s(\bar{t})$. Notice that h is well-defined, since we assumed that two distinct queries among $q_0, p_1, \ldots, p_n$ only share variables that are among $\bar{x}$.

Now since $\theta(\bar{x}) \models q(\bar{x})$ and $s(\bar{x}) \models \theta(\bar{x})$, we obtain that $s(\bar{x}) \models q(\bar{x})$. Moreover, since $s(\bar{x})$ is strictly acyclic, by the definition of $\Lambda_q^{\mathbf{T} \cup \{C\}}$ there must be a strictly acyclic $p(\bar{x}) \in \Lambda_q^{\mathbf{T} \cup \{C\}}$ that results from $s(\bar{x})$ by only removing atoms. Since $p(\bar{x})$ differs from $s(\bar{x})$ by only removing atoms, it follows that $s(\bar{x}) \models p(\bar{x})$, whence $\mathfrak{J} \models s(\bar{t})$ entails $\mathfrak{J} \models p(\bar{t})$ as desired. □

Lemma 4.57. *For any* $\mathbf{S}$*-database* $\mathfrak{D}$*, if* $\mathfrak{D} \models Q$ *then* $\mathfrak{D}_C \models \eta_C(Q)$.

Proof. We shall prove that there is a family $\{h_k\}_{k \geq 0}$ of homomorphisms, each of which is the identity on $\mathsf{dom}(\mathfrak{D})$, such that (i) h_k is a homomorphism from $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+)$, (ii) h_{k+1} extends h_k, and (iii) $h \coloneqq \bigcup_{k \geq 0} h_k$ is a homomorphism from $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+)$.

We proceed by induction on k. The base case is entirely trivial, as we can take the identity on $\mathsf{dom}(\mathfrak{D})$ for h_0.

For the induction step, suppose that h_k is a homomorphism from $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+)$, and consider the structure $\mathsf{chase}^{k+1}(\mathfrak{D}, \mathcal{O})$. Let $\beta(\bar{t}, \bar{\lambda})$ be the atom derived in the $(k+1)$-st chase step by the application of some frontier-guarded rule with body $q(\bar{x})$ and head $\exists \bar{y} \, \beta(\bar{x}, \bar{y})$ using the homomorphism h' with $h(\bar{x}) = \bar{t}$. Assume further that $\bar{\lambda}$ is a sequence of labeled nulls that does not occur in $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$. Since $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O}) \models q(\bar{t})$, Lemma 4.56 tells us that there is a $p(\bar{x}) \in \Lambda_q^{\mathbf{T} \cup \{C\}}$ such that $\mathsf{chase}^k(\mathfrak{D}_C, \mathcal{O}) \models p(\bar{t})$. Due to the fact that h_k is a homomorphism from $\mathsf{chase}^k(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+)$, it follows that $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+) \models p(h_k(\bar{t}))$. We can thus extend h_k to a mapping h_{k+1} by mapping the nulls $\bar{\lambda}$ to appropriate nulls in $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+)$ so that h_{k+1} is a homomorphism from $\mathsf{chase}^{k+1}(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+)$. This concludes the induction step.

It is clear that $\{h_k\}_{k \geq 0}$ is a family of homomorphisms with the desired properties. In particular, it is not hard to check that $h \coloneqq \bigcup_{k \geq 0} h_k$ is a homomorphism from $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ to $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+)$. Since $\mathsf{chase}(\mathfrak{D}, \mathcal{O}) \models G$ (in other symbols, $\mathfrak{D} \models Q$), it thus follows that $\mathsf{chase}(\mathfrak{D}_C, \mathcal{O}^+) \models G$, that is, $\mathfrak{D}_C \models \eta_C(Q)$ as required. □

We are now ready to prove the main result that is used to reduce $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$ to $\mathsf{Cont}(\mathsf{G}, \mathsf{BAQ})$:

Theorem 4.58. *Suppose* $Q_1 = (\mathbf{S}, \mathcal{O}_1, G_1)$ *and* $Q_2 = (\mathbf{S}, \mathcal{O}_2, G_2)$ *are OMQs that fall into* $(\mathsf{FG}, \mathsf{BAQ})$*, and suppose* C *is a relation symbol of arity* $\max\{\mathsf{wd}(\mathcal{O}_1), \mathsf{wd}(\mathcal{O}_2)\}$ *that does neither occur in* Q_1 *nor in* Q_2*. Then,*

$$Q_1 \subseteq Q_2 \iff \eta_C(Q_1) \subseteq \eta_C(Q_2).$$

Proof. Suppose first that $Q_1 \not\subseteq Q_2$. Then there is an $\mathbf{S}$-database $\mathfrak{D}$ such that $\mathfrak{D} \models Q_1$ but $\mathfrak{D} \not\models Q_2$. By Lemma 4.57, $\mathfrak{D}_C \models \eta_C(Q_1)$, and thus by Lemma 4.36 there is an acyclic $(\mathbf{S} \cup \{C\})$-database $\mathfrak{B}$ such that $\mathfrak{B} \models \eta_C(Q_1)$ and a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}_C$. Obviously, $\mathfrak{D}_C \not\models Q_2$, and since Q_2 is closed under weak homomorphisms, we obtain $\mathfrak{B} \not\models Q_2$ as well. Lemma 4.55 yields $\mathfrak{B} \not\models \eta_C(Q_2)$, whence $\eta_C(Q_1) \not\subseteq \eta_C(Q_2)$ follows.

Suppose now that $\eta_C(Q_1) \not\subseteq \eta_C(Q_2)$. By Theorem 4.34, there is an acyclic $(\mathbf{S} \cup \{C\})$-database $\mathfrak{D}$ such that $\mathfrak{D} \models \eta_C(Q_1)$ but $\mathfrak{D} \not\models \eta_C(Q_2)$, whence Lemma 4.55 yields that $\mathfrak{D} \restriction \mathbf{S} \models Q_1$ and $\mathfrak{D} \restriction \mathbf{S} \not\models Q_2$. Hence, $Q_1 \not\subseteq Q_2$ as required. □

Putting Everything Together

Now let $Q_1 = (\mathbf{S}, \mathcal{O}_1, G_1)$ and $Q_2 = (\mathbf{S}, \mathcal{O}_2, G_2)$ be OMQs from $(\mathsf{FG}, \mathsf{BAQ})$. We show that $Q_1 \subseteq Q_2$ can be decided in 2ExpTime. Let C be a fresh relation symbol of arity $\max\{\mathsf{wd}(\mathcal{O}_1), \mathsf{wd}(\mathcal{O}_2)\}$ that does neither occur in Q_1 nor in Q_2. We construct $\eta_C(Q_1)$ and $\eta_C(Q_2)$, and each of these constructions is feasible in exponential time in the size of the respective OMQ. Notice that $\eta_C(Q_1)$ and $\eta_C(Q_2)$ may introduce exponentially many rules and exponentially many fresh predicates. However, by Theorem 4.17, we can decide $\eta_C(Q_1) \subseteq \eta_C(Q_2)$ in doubly exponential time in the combined sizes of Q_1 and Q_2, since in the construction of $\eta_C(Q_1)$ and $\eta_C(Q_2)$, we only increased the maximum arity used in any of the two OMQs Q_1, Q_2 linearly (the relation symbol of maximum arity in any of the two is C). By Theorem 4.58, we know that $Q_1 \subseteq Q_2$ iff $\eta_C(Q_1) \subseteq \eta_C(Q_2)$. Hence, we can decide $Q_1 \subseteq Q_2$ in 2ExpTime in the combined size of Q_1 and Q_2. This proves that $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$ is in 2ExpTime, and Theorem 4.16 follows.

4.3.3 On the Use of Constants

Let us now, as promised in the introductory part of this section, explain how to extend the results of this section to the case where the use of constants is allowed in rules and queries. It turns out that similar results as those in this section can be proved for this case, but it involves more complicated concepts from a technical point of view, which is why we focused on the more specific constant-free case in the first place. The following aspects need to be adapted in order to accommodate the use of constants:

▶ The tree witness property stated in Theorem 4.34 needs to be adjusted. We cannot focus on acyclic structures anymore, but we need to be more liberal in the sense that some part of the tree witness may be cyclic. In fact, having constants in rules destroys the tree witness property, as constants may query cycles – for example, the rule $R(a, b), R(b, c), R(c, a) \to G$, with a, b, c being constants, is guarded, but its body has no acyclic model over the schema $\{R\}$. However, the cycles we must allow in witnessing

databases turn out to be very mild. In fact, instead of working with acyclic structures, we can work with $\mathfrak{C}$-trees instead, where $\mathfrak{C}$ only depends on the constants used in the OMQ. Thus, OMQs having constants still exhibit a tree witness property, but the notion of "being tree-like" is more liberal in the sense that one part of the structure may have cycles. A version of Theorem 4.34 can then be established that works with $\mathfrak{C}$-trees instead of acyclic structures.

▶ The encoding and decoding of trees needs to also capture $\mathfrak{C}$-trees. This is not hard to establish at all, as $\mathfrak{C}$-trees – once $\mathfrak{C}$ is appropriately bounded in size – are also structures of bounded tree-width. Apart from the adjustment toward $\mathfrak{C}$-trees, the encoding of them also needs to have special names for constants in order to ensure that constants denote the same element in the entire encoding. This adds additional elements to the alphabet $\Gamma_{\mathbf{S}}$, but poses no real issue. The consistency criteria for $\Gamma_{\mathbf{S}}$-labeled trees are then adjusted to be sure that these names for constants are used appropriately (similar encodings can be found in [28]).

▶ For Lemma 4.44, recall that we heavily exploited the fact that OMQs from $(\mathsf{G}, \mathsf{AQ})$ can be equivalently rewritten into guarded Datalog queries. Moreover, we use the construction of such rewritings as provided in [17]. The proof of Lemma 4.44 can be carried out without modifications, since in [17] the case where constants are allowed is explicitly taken into account.

▶ The automata $\mathcal{C}_{\mathbf{S},w}$ and $\mathcal{A}_Q$ from, respectively, Lemmas 4.41 and 4.42 can be devised in the same manner, as the crucial modifications required to make their constructions work out for this case have been done made in the previous points.

▶ Finally, the material from Subsection 4.3.2 has to be adjusted to accommodate the use of constants. In particular, acyclic queries are now defined *modulo* the use of constants, so the sustain the equivalence between strictly acyclic queries and strictly guarded formulas that may use constants. For example, the formula $q \coloneqq R(a, b) \wedge R(b, c) \wedge R(c, a)$, with a, b, c being constants, is strictly guarded in this sense as well, and thus is considered to be acyclic as well – in fact, the treeification of q equals q itself. The definitions of treeification, etc. are along the established lines, but employ this definition of acyclicity modulo the use of constants. Once these concepts are set up in that way, one can proceed with the results of Subsection 4.3.2 in an according fashion, which at the end cumulates in the 2ExpTime-membership of $\mathsf{Cont}(\mathsf{FG}, \mathsf{BAQ})$.

4.4 Combining Languages

In the previous three sections, we studied the containment problem with the focus on a single language $\mathcal{L}$, i.e., both OMQs fall into $\mathcal{L}$. However, it is natural to consider the version of the problem where the involved OMQs fall into different languages. This is the goal of this section. Our analysis proceeds by considering two cases. In the first case, the left-hand side query falls into a UCQ-rewritable OMQ language, while in the second case, it is guarded or frontier-guarded. As we shall see below, we will mostly exploit the machinery developed in Sections 4.2 and 4.3 with some notable exceptions.

4.4.1 The Left-Hand Side Query is UCQ-Rewritable

As an immediate corollary of Theorem 4.8, we obtain that $\mathsf{Cont}((\mathcal{C}_1, \mathsf{CQ}), (\mathcal{C}_2, \mathsf{CQ}))$, for $\mathcal{C}_1 \neq \mathcal{C}_2$, $\mathcal{C}_1 \in \{\mathsf{L}, \mathsf{NR}, \mathsf{S}\}$ and $\mathcal{C}_2 \in \{\mathsf{L}, \mathsf{NR}, \mathsf{S}, \mathsf{FG}, \mathsf{G}\}$, is decidable. By exploiting the algorithm underlying Theorem 4.8, we establish optimal upper bounds for all the problems at hand with the only exception of $\mathsf{Cont}((\mathsf{S}, \mathsf{CQ}), (\mathsf{NR}, \mathsf{CQ}))$. For the latter, we obtain a coNExpTime$^{\mathrm{NP}}$ upper bound by providing a similar analysis as for $\mathsf{Cont}(\mathsf{NR}, \mathsf{CQ})$, while a NExpTime lower bound is inherited from query evaluation by exploiting Proposition 4.1. It is rather tedious – and not very interesting from a technical point of view – to go through all the containment problems in question and explain in detail how the exact upper bounds are obtained.

Regarding the matching lower bounds, in most of the cases they are inherited from query evaluation or its complement by exploiting Propositions 4.1 and 4.2, respectively. There are, however, some exceptions:

- $\mathsf{Cont}((\mathsf{S}, \mathsf{CQ}), (\mathsf{L}, \mathsf{CQ}))$, in the case of unbounded arity, is coNExpTime-hard, assuming that the sets of TGDs are allowed to use two constants. This is shown by a reduction from the standard tiling problem for the exponential grid. A proof of this fact can be found in Appendix B.2 in the proof of Theorem 4.15.
- $\mathsf{Cont}((\mathsf{L}, \mathsf{CQ}), (\mathsf{S}, \mathsf{CQ}))$ and $\mathsf{Cont}((\mathsf{S}, \mathsf{CQ}), (\mathsf{L}, \mathsf{CQ}))$ in the case of schemas of bounded width, where both problems are Π_2^{P}-hard even for constant-free TGDs. This result is implicit in [35], where containment for OMQ languages based on different description logics is considered.

4.4.2 The Left-Hand Side Query is (Frontier-)Guarded

We proceed with the case where the left-hand side query is (frontier-)guarded, and we show the following result:

Theorem 4.59. *The problem* $\mathsf{Cont}((\mathcal{L}, \mathsf{CQ}), (\mathcal{C}, \mathsf{CQ}))$*, where* $\mathcal{L} \in \{\mathsf{G}, \mathsf{FG}\}$*, is*

(i) 2ExpTime*-complete in case* $\mathcal{C} \in \{\mathsf{L}, \mathsf{S}\}$*, and*

(ii) 3ExpTime*-complete in case* $\mathsf{C} = \mathsf{NR}$*.*

The lower bounds hold even for the case where we use schemas of bounded width.

Lower Bounds

Recall that in [23] it is shown that containment of guarded Datalog queries in acyclic CQs is 2ExpTime-hard. This immediately implies that $\mathsf{Cont}((\mathsf{G}, \mathsf{CQ}), (\mathcal{C}, \mathsf{CQ}))$ is 2ExpTime-hard for $\mathcal{C} \in \{\mathsf{L}, \mathsf{S}\}$.

We only describe briefly how the 3ExpTime-hardness of $\mathsf{Cont}((\mathsf{G}, \mathsf{CQ}), (\mathsf{NR}, \mathsf{CQ}))$ is obtained and omit the rather tedious details. One can establish this matching lower bound by refining techniques from [55], where it is shown that containment of Datalog queries in UCQs is 2ExpTime-complete, while containment of Datalog queries in non-recursive Datalog queries is 3ExpTime-complete. The lower bounds hold for predicates of bounded

arity and constant-free rules. Interestingly, the left-hand side query in these proofs can be transformed into a Datalog query such that each rule has a body-atom that contains all the variables, i.e., one that is guarded. This is achieved by increasing the arity of some predicates in order to have enough positions for all the body variables. However, for each rule, the number of unguarded variables that we need to guard is constant, and thus the arity of the schema remains constant. Providing details for this reduction requires to revisit the (rather lengthy) proofs in [55] which is why we omit details here. This allows us to conclude that $\mathsf{Cont}((\mathsf{G},\mathsf{CQ}),(\mathsf{NR},\mathsf{CQ}))$ is 3ExpTime-hard.

Upper Bounds

For $\mathcal{C} = \mathsf{L}$, membership in 2ExpTime is an immediate corollary of Theorem 4.16, since every set of linear TGDs is also guarded. This is not true when $\mathcal{C} \in \{\mathsf{NR},\mathsf{S}\}$, since the right-hand side query is not guarded. But in this case, since $(\mathsf{NR},\mathsf{CQ})$ and (S,CQ) are UCQ-rewritable, one can rewrite the right-hand side query as a UCQ, and then apply the machinery developed in Section 4.3 for solving containment among (frontier-)guarded OMQs. Before going into details, let us point out that it is an immediate corollary of Theorem 4.16 that the problem of deciding whether a $Q \in (\mathsf{FG},\mathsf{CQ})$ is contained in a CQ q is feasible in 2ExpTime.

Now given OMQs $Q_1 \in (\mathsf{FG},\mathsf{CQ})$ and $Q_2 \in (\mathcal{C},\mathsf{CQ})$, where $\mathcal{C} \in \{\mathsf{NR},\mathsf{S}\}$, it holds that $Q_1 \subseteq Q_2$ iff $Q_1 \subseteq q$, where q is a UCQ-rewriting of Q_2. Thus, an immediate decision procedure, which exploits the algorithm $\mathsf{XRewrite}$, is the following:

(i) Let $q := \mathsf{XRewrite}(Q_2)$.

(ii) If $Q_1 \not\subseteq q'$ for some disjunct q' of q, then REJECT; otherwise ACCEPT.

The above procedure runs in triply exponential time. The first step is feasible in doubly exponential time [82]. Now, for a single CQ q' that is a disjunct of q, the check whether $Q_1 \subseteq q'$ can be done by using the machinery developed in Theorem 4.16, which reduces our problem to checking whether the language of a suitable 2APTA is empty. It should not be forgotten that q' is of exponential size, and thus membership in 3ExpTime follows.

The case of sticky OMQs Although the above algorithm establishes an optimal upper bound for OMQs based on non-recursive sets of TGDs, a more refined analysis is needed for those based on sticky sets of TGDs. In fact, we need a more refined complexity analysis for the problem $\mathsf{Cont}((\mathsf{FG},\mathsf{CQ}),\mathsf{UCQ})$, that is, to decide whether a frontier-guarded OMQ is contained in a UCQ. To this end, we provide an automata construction that takes the shapes of the rewritings of sticky OMQs into account. This allows us to establish a refined complexity upper bound for the problem in question. To this end, let us introduce some auxiliary notation. Let us for simplicity again assume that all the sets of TGDs and CQs we encounter are constant-free.

Given a BCQ q and $k \geq 1$, we write $\mathsf{var}_{\geq k}(q)$ for the set of variables of $\mathsf{var}(q)$ that occur in more than k distinct relational body atoms of q. Accordingly, $\mathsf{var}_{\leq k}(q)$ denotes the set of variables among $\mathsf{var}(q)$ that appear in at most k distinct atoms of q. It turns

out that we can devise more fine-grained automata for CQs when taking that measure into account:

Lemma 4.60. *Let q be a BCQ over a schema $\mathbf{S}$. There exists a $\updownarrow$-2APTA $\mathfrak{A}_q$ such that, for every consistent $\Gamma_{\mathbf{S}}$-labeled tree t,*

$$t \in \mathcal{L}(\mathcal{A}_q) \iff [\![t]\!] \models q.$$

Its number of states is exponential in $|\mathsf{var}_{\geq 2}(q)|$, and polynomial in $|\mathsf{var}_{\leq 1}(q)| + \mathsf{wd}(\mathbf{S})$. Furthermore, we can construct $\mathcal{A}_q$ in exponential time.

A proof of Lemma 4.60 can be found in Appendix B.2. We can employ Lemma 4.60 to show that $\mathsf{Cont}((\mathsf{FG}, \mathsf{CQ}), (\mathsf{S}, \mathsf{CQ}))$ is in 2ExpTime as follows. It is not hard to show that $\mathsf{Cont}((\mathsf{FG}, \mathsf{CQ}), (\mathsf{S}, \mathsf{CQ}))$ can be reduced in polynomial time to $\mathsf{Cont}((\mathsf{FG}, \mathsf{BAQ}), (\mathsf{S}, \mathsf{BCQ}))$ – a proof of this fact can be carried out *mutatis mutandis* to the proof of Lemma 4.45 and is left as an exercise for the reader.

Let $Q_1 = (\mathbf{S}, \mathcal{O}_1, q_1)$ from $(\mathsf{FG}, \mathsf{BAQ})$ and $Q_2 = (\mathbf{S}, \mathcal{O}_2, q_2)$ from $(\mathsf{S}, \mathsf{BCQ})$. Let $q \coloneqq \mathsf{XRewrite}(Q_2)$ be the UCQ obtained from applying $\mathsf{XRewrite}$ to Q_2, and assume that $q = \bigvee_{i=1}^{n} q'_i$ for BCQs $q'_1, \ldots, q'_n$.[6] It holds that (see [82]):

(i) The UCQ q consists of at most doubly exponentially many CQs.

(ii) Each disjunct of q is of at most exponential size.

(iii) For each disjunct q' of q, the set $\mathsf{var}_{\geq 2}(q')$ is a subset of the variables of the CQ q_2.

Let C be a fresh predicate of arity $\mathsf{wd}(\mathcal{O}_1)$. We then construct $\eta_C(Q_1)$ in exponential time as detailed in Subsection 4.3.2. Recall that $\eta_C(Q_1)$ falls into $(\mathsf{G}, \mathsf{BAQ})$, and that the maximum arity of any relation symbol occurring in $\eta_C(Q_1)$ is bounded by the arity of C. Let $\mathcal{B}$ be the intersection of (i) the $\updownarrow$-2APTA from Lemma 4.41 that accepts exactly the consistent $\Gamma_{\mathbf{S}}$-labeled trees, and (ii) the $\updownarrow$-2APTA $\mathcal{A}_{\eta_C(Q_1)}$ from Lemma 4.42. Again, although $\eta_C(Q_1)$ is of exponential size in the size of Q_1, the automaton $\mathcal{B}$ has at most exponentially many states in the size of Q_1. Now for each disjunct q'_i of q, we construct the automaton $\mathcal{A}_{q'_i}$ according to Lemma 4.60. Due to item (ii) above, each of these automata has at most exponentially many states in the size of Q_2. Now we put

$$\mathcal{A} \coloneqq \mathcal{B} \cap \overline{(\mathcal{A}_{q'_1} \cup \cdots \cup \mathcal{A}_{q'_n})}.$$

Observe that $\mathcal{A}$ still has only exponentially many states in the combined sizes of Q_1 and Q_2, since there are only exponentially many disjuncts in q, and union of 2APTA is feasible in polynomial time (see Proposition 2.31).

We claim that $Q_1 \subseteq Q_2$ iff $\mathcal{L}(\mathcal{A}) = \varnothing$. The "only if" direction is immediate by Lemmas 4.41, 4.42 and 4.60. Suppose therefore that $Q_1 \not\subseteq Q_2$, that is, $Q_1 \not\subseteq q'_i$ for some $i \in \{1, \ldots, n\}$. Hence, there is an $\mathbf{S}$-database $\mathfrak{D}$ such that $\mathfrak{D} \models Q_1$, yet $\mathfrak{D} \not\models q'_i$. By

[6] Actually, we need to convert Q_2 into normal form first, as required by $\mathsf{XRewrite}$ (see Appendix A). According to this normal form, each TGD of Q_2 has at most one atom in the head and at most one existential variable in the head. It is not hard to show that putting Q_2 into this normal form is feasible in polynomial time [47].

Lemma 4.36, there is an acyclic $\mathbf{S}$-database $\mathfrak{D}'$ such that (i) $\mathfrak{D}' \models Q_1$, and (ii) there is a weak homomorphism from $\mathfrak{D}'$ to $\mathfrak{D}$. Since satisfaction of constant-free CQs is preserved under weak homomorphisms, we must have $\mathfrak{D}' \not\models q_i'$. Now we can encode $\mathfrak{D}'$ as a finite and consistent $\Gamma_{\mathbf{S}}$-labeled tree t as detailed in Section 4.3 such that $[\![t]\!] \models Q_1$ and $[\![t]\!] \not\models q_i'$ (cf. Lemma 4.39). By Lemmas 4.41, 4.42 and 4.60, we obtain $t \in \mathcal{L}(\mathcal{A})$, as required.

4.5 Summary

We investigated the query containment problem for OMQ languages based on prominent classes of TGDs. We established the relationship between query evaluation and query containment, and showed that the decidability of the latter depends crucially on the decidability of the former. The solutions to query containment depend on the question whether the involved languages are UCQ-rewritable. For the case where both languages are UCQ-rewritable, we solved containment by exploiting the construction of UCQ-rewritings for the OMQs at hand. Nearly all the problems were solved optimally, with the notable exception of containment of OMQs based on non-recursive sets of TGDs, where we established $\textsc{PTime}^{\textsc{NExpTime}}$-hardness and membership in $\textsc{coNExpTime}^{\textsc{NP}}$. For (frontier-)guarded OMQs, which are not UCQ-rewritable, we established a tree-like witness property and resorted to automata techniques. Finally, we also considered cases where we combined different languages, i.e., we considered the containment problem where the left-hand side query and the right-hand side query fall into different languages. It turned out that interesting cases arise when the left-hand side query is (frontier-)guarded, and we developed specifically tailored decision procedures for these cases.

CHAPTER 5

First-Order Rewritability for Guarded-Based OMQs

In Subsection 3.4.2 we called a set of $\mathcal{O}$ TGDs *first-order rewritable*, if, for every UCQ q, one can effectively construct a first-order query $\varphi_{q,\mathcal{O}}$ such that $\varphi_{q,\mathcal{O}}(\mathfrak{D}) = \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$ for every database $\mathfrak{D}$. In Section 4.3 we lifted the notion of rewritability to the level of OMQs and called an OMQ *UCQ-rewritable*, if we can effectively construct a UCQ equivalent to it. Likewise, we call an OMQ *first-order rewritable* if we can effectively construct a first-order query equivalent to it – in fact, we will see that an OMQ is first-order rewritable iff it is UCQ-rewritable.

Having a first-order rewritable Q at hand allows us to solve the evaluation problem for Q using standard database technology – indeed, given Q and a database $\mathfrak{D}$, we can compute $Q(\mathfrak{D})$ by first rewriting Q into an equivalent first-order query φ, and then pass φ to a relational database management system in order to compute $\varphi(\mathfrak{D}) = Q(\mathfrak{D})$. Therefore, it is not surprising that the task of checking first-order rewritability – and finding appropriate rewritings – is of paramount interest in the context of ontology-mediated querying.

As we have already observed in Subsection 3.4.2 and Section 4.2, for OMQs based on linear, non-recursive, and sticky sets of TGDs, first-order rewritings are always guaranteed to exist. Therefore, the question whether such OMQs are first-order rewritable trivializes. The situation looks quite different for OMQs whose ontology is (frontier-)guarded. As we shall see, there are (frontier-)guarded OMQs that are inherently recursive, and thus not expressible as a first-order queries. This brings us to our main question of this chapter: *can we check whether a (frontier-)guarded OMQ is first-order rewritable, and, if yes, pinpoint its exact complexity?* Notice that, for OMQs based on more expressive classes of TGDs that capture Datalog, the answer to the above question is negative, since checking whether a Datalog query is first-order rewritable is an undecidable problem. Actually, we know that a Datalog query is first-order rewritable iff it is *bounded* [3], while the boundedness problem for Datalog is undecidable [78].

The above question has been studied in [34] for OMQ languages based on Horn-DLs, including those from the $\mathcal{EL}$-family, which are, as already mentioned, special cases of OMQs based on guarded TGDs. More precisely, in [34] first-order rewritability is semantically characterized by a locality property on tree-shaped ABoxes (i.e., databases), which in turn allows the authors to pinpoint the complexity of first-order rewritability by employing automata-based procedures. As usual in the context of description logics, schemas consist only of unary and binary relations. However, in our setting we have to deal with relations of higher arity. This entails, as we shall see, that the techniques devised for checking first-order rewritability for DL-based OMQs cannot be directly applied to OMQs based on (frontier-)guarded rules. We therefore develop new semantic characterizations and procedures that are significantly different from those for OMQs based on description logics.

Outline and contributions Our plan of attack and contributions can be summarized as follows:

▶ In Section 5.1, we introduce the problem setting formally, and we provide some background on it. We point out that, toward a solution of first-order rewritability for frontier-guarded OMQs, we first focus on the simpler OMQ language based on guarded TGDs and Boolean atomic queries, i.e., the language $(\mathsf{G}, \mathsf{BAQ})$. This is reminiscent of the solution of the containment problem for OMQs based on (frontier-)guarded TGDs in Section 4.3.

▶ In Section 5.2 we provide a semantic characterization of first-order rewritability that forms the basis for applying tree automata techniques.

▶ We then exploit, in Section 5.3, standard two-way alternating parity tree automata. In particular, we reduce first-order rewritability to the problem of checking whether the language accepted by a certain automaton is finite. The reduction relies on a refined version of the characterization of first-order rewritability established in Section 5.2. This provides a transparent solution to our problem based on standard tools, but unfortunately does not lead to an optimal result in terms of complexity.

▶ Toward an optimal complexity result, in Section 5.4 we use a more sophisticated automata model, known as *cost automata* [31, 58, 60]. This allows us to show that first-order rewritability for OMQs from $(\mathsf{G}, \mathsf{BAQ})$ is in 2ExpTime, and in ExpTime for schemas of bounded width. Our application of cost automata is quite transparent, which again relies on a refined version of the characterization of first-order rewritability established in Section 5.2. However, the complexity analysis relies on an intricate result on the boundedness problem for a certain class of cost automata from [31, 59].

▶ Finally, in Section 5.5 we obtain our main results. We show that first-order rewritability is 2ExpTime-complete for OMQs based on guarded TGDs and on frontier-guarded TGDs, no matter whether the actual queries are conjunctive queries, unions thereof, or simply atomic queries. This remains true when the width of the underlying schema is bounded, with the exception of guarded TGDs and atomic queries, for which the complexity then drops to ExpTime-completeness. A proof of this result relies again on the treeification technique, which we have already encountered in Section 4.3.

5.1 Problem Statement

We remind the reader again that, in Subsection 3.4.1, we called a set of TGDs $\mathcal{O}$ a first-order rewritable set if, for every UCQ q, one can effectively construct a first-order query $\varphi_{q,\mathcal{O}}$ such that $\varphi_{q,\mathcal{O}}(\mathfrak{D}) = \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$ for every database $\mathfrak{D}$. We observed in Subsection 3.4.2 that linear, non-recursive, and sticky sets of TGDs are always first-order rewritable, and the procedure XRewrite from Section 4.2 revealed that actually their rewritings are always UCQs (this is no coincidence, as we shall see below in Lemma 5.5). Thus, first-order rewritability was defined for a set of TGDs and the existence of first-order rewritings is demanded universally for every UCQ. However, the rewriting $\varphi_{q,\mathcal{O}}$ may depend on q, while it is important to notice that the rewriting $\varphi_{q,\mathcal{O}}$ is uniform for all input databases.

The following definition lifts the notion of first-order rewritability to the level of OMQs:

Definition 5.1. An OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ is *first-order rewritable*, if there exists a first-order query $\varphi_Q(\bar{x})$ over $\mathbf{S}$ that is equivalent to Q, i.e., for every $\mathbf{S}$-database $\mathfrak{D}$ it holds that

$$Q(\mathfrak{D}) = \varphi_Q(\mathfrak{D}).$$

We call $\varphi_Q(\bar{x})$ a *first-order rewriting of* Q.

A fundamental static analysis task for an OMQ language $(\mathcal{L}, \mathcal{Q})$, where $\mathcal{L}$ is a class of TGDs and $\mathcal{Q}$ is a class of queries, is deciding first-order rewritability:

Problem:	$\mathsf{FO}^{\Leftarrow}(\mathcal{L}, \mathcal{Q})$
Input:	An OMQ $Q \in (\mathcal{L}, \mathcal{Q})$.
Question:	Is it the case that Q is first-order rewritable?

Obviously, if $\mathcal{O}$ is a first-order rewritable set, then any OMQ of the form $Q = (\mathbf{S}, \mathcal{O}, q)$ is first-order rewritable. For a class of first-order rewritable sets $\mathcal{L}$, $\mathsf{FO}^{\Leftarrow}(\mathcal{L}, \mathsf{CQ})$ is thus trivial. On the other hand, it is easy to check that here are very simple OMQs that are not first-order rewritable:

Example 5.2. Consider the OMQ $Q = (\{E/2\}, \mathcal{O}, T(x, y))$, where $\mathcal{O}$ consists of the two rules

$$E(x, y) \rightarrow T(x, y),$$
$$T(x, y), T(y, z) \rightarrow T(x, z).$$

Q can be viewed as a Datalog query that computes the transitive closure of the input relation E. Hence, for an $\{E\}$-database $\mathfrak{D}$, we have that $\mathfrak{D} \models Q(a, b)$ iff b is reachable from a via a directed E-path in $\mathfrak{D}$. It is well-known that Q is an example of a Datalog query that has no first-order rewriting, since reachability in a graph is not first-order expressible (see, e.g., [71, 102] for this classical result).

As another example, take the OMQ $Q' = (\{A/1, E/2\}, \mathcal{O}', B(x))$, where $\mathcal{O}'$ consists of the two rules

$$A(x) \to B(x),$$
$$B(x), E(x, y) \to B(y).$$

Q' again corresponds to a Datalog query. Notice however that, unlike Q above, Q' is guarded and even *monadic*, that is, all its intensional predicates are unary.[1] For any $\{A, E\}$-database $\mathfrak{D}$, $\mathfrak{D} \models Q'(b)$ iff b can be reached via a directed E-path from some element $a \in \mathsf{adom}(\mathfrak{D})$ such that $\mathfrak{D} \models A(a)$. Thus, Q' also encodes some form of reachability when viewing $\mathfrak{D}$ as a directed graph. We will see later that Q' is not first-order rewritable either.

On the other hand, consider the OMQ $Q'' = (\{A/1, E/2, B/1\}, \mathcal{O}'', G(x, y))$ taken from [34], where $\mathcal{O}''$ consists of the three rules

$$A(x), E(x, y) \to A(y),$$
$$B(x), E(x, y) \to A(y),$$
$$B(x), E(x, y), A(y) \to G(x, y).$$

Then $\varphi(x, y) \coloneqq B(x) \wedge E(x, y)$ is a first-order rewriting of Q''. ⊣

(Un)decidability results Let us elaborate on the existing literature on related results on first-order rewritability here. Considering the case of Datalog queries, it turns out that a Datalog query $Q = (\Pi, G)$ is first-order rewritable iff it is *bounded* [3], i.e., iff there exists a uniform $k \geq 0$ such that, for every input database $\mathfrak{D}$ and every tuple $\bar{a}$, it holds that $\mathfrak{D} \models Q(\bar{a})$ iff $T_Q^k(\mathfrak{D}) \models G(\bar{a})$. Intuitively, a Datalog query is thus first-order rewritable iff its least fixed-point can be reached within a fixed number of k steps such that is uniform for all input databases.

Unfortunately, it turns out that the problem of deciding boundedness – and thus that of deciding first-order rewritability – is undecidable [78]. This has led to quite some activity in the research community to identify Datalog fragments for which boundedness is decidable. As in the case of query containment, it turns out that fragments of Datalog that exhibit some form of tree-model property enjoy a decidable boundedness problem. To wit, in [62] it is shown that boundedness for monadic Datalog queries is decidable. In [38] it is shown that decidability of the boundedness problem over the class of all trees extends to monadic least fixed-point recursion based on positive monadic second-order formulas. Roughly speaking, given a formula $\varphi(\bar{x}, X)$ with a second-order variable X that only occurs under an even number of negations, the *boundedness problem* asks whether the least-fixed point induced by $\varphi(\bar{x}, X)$ is reached within some uniform bound that is independent of the structure at hand. Notice that this extends the notion of boundedness of Datalog queries to arbitrary formulas from least fixed-point logic (LFP); see, e.g., [102] for more details on LFP vs. Datalog. The authors of [38] extend their results to the case of guarded second-order logic (GSO) over classes of structures of fixed,

[1] Notice that monadic Datalog queries are also frontier-guarded.

finite tree-width. These results entail a number of decidability results for the boundedness problems of Datalog fragments, e.g., it follows that boundedness is decidable for guarded and frontier-guarded Datalog as well. Moreover, they also imply that boundedness of guarded least fixed-point logic (GFP) and guarded negation fixed-point logic (GNFP) is decidable, since both of them can be naturally expressed as formulas of guarded second-order logic.

Benedikt et al. [31] study the problem of boundedness for guarded negation formulas in depth, and they show that boundedness of formulas $\varphi(\bar{x}, X)$ is decidable in elementary time when $\varphi(\bar{x}, X)$ is a guarded formula or a guarded negation formula. In fact, they even show that this problem is 2ExpTime-complete, and that 2ExpTime-completeness extends to the case of GNFP-formulas. To this end, they exploit cost automata models and show that boundedness of guarded negation and GNFP-formulas can naturally be reduced to the question of whether (the cost function defined by) a suitable cost automaton is bounded uniformly across its inputs. We will heavily use the machinery developed in [31] for our purposes in Section 5.4.

In the context of ontology-mediated querying, first-order rewritability is investigated in [34, 36] for description logics between $\mathcal{EL}$ and *Horn-*$\mathcal{SHIF}$. In particular, it is shown that first-order rewritability is 2ExpTime-complete for any OMQ-Language between $(\mathcal{ELI}, \mathsf{CQ})$ and $(\textit{Horn-}\mathcal{SHIF}, \mathsf{CQ})$, while it is ExpTime-complete for $(\mathcal{EL}, \mathsf{AQ})$. These results are obtained by showing that first-order rewritability for these classes can be characterized by the fact that any tree-like database $\mathfrak{D}$ satisfying such an OMQ Q at hand already satisfies Q when one restricts $\mathfrak{D}$ to a certain depth that only depends on Q (see below for more details). Decision procedures for first-order rewritability are then obtained by employing automata on trees. We will see below in Section 5.2 that the methods developed in [34, 36] cannot be extended to our setting due to the use of predicates of arity greater than two.

First-order rewritability for the class (FG, UCQ) As mentioned in the introductory part of this chapter, our goal here is to investigate the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{UCQ})$. More specifically, our main result reads as follows:

Theorem 5.3. (i) *For* $\mathcal{Q} \in \{\mathsf{UCQ}, \mathsf{CQ}, \mathsf{AQ}\}$*, the problem* $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathcal{Q})$ *is* 2ExpTime*-complete, and this holds even for schemas of bounded width.*

(ii) *For* $\mathcal{Q} \in \{\mathsf{UCQ}, \mathsf{CQ}\}$*, the problem* $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathcal{Q})$ *is* 2ExpTime*-complete, and this holds even for schemas of bounded width.*

(iii) $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{AQ})$ *is* 2ExpTime*-complete, and* ExpTime*-complete assuming schemas of bounded width.*

Lower bounds Let us first explain how to obtain the lower bounds stated in Theorem 5.3. The 2ExpTime-hardness in items (i) and (ii) is inherited from [34], where it is shown that deciding first-order rewritability for OMQs from $(\mathcal{ELI}, \mathsf{CQ})$ is 2ExpTime-hard. For the 2ExpTime-hardness in item (iii), we exploit the fact that containment for OMQs from $(\mathsf{G}, \mathsf{BAQ})$ is 2ExpTime-hard, even if the right-hand side query is first-order

rewritable – this is implicit in [23] has already been discussed in Section 4.3. A proof of the fact that $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ is 2ExpTime-hard is given in Appendix B.3. Finally, the ExpTime-hardness in the third item is inherited from [36], where it is shown that deciding first-order rewritability for OMQs from $(\mathcal{EL}, \mathsf{BAQ})$ is ExpTime-hard.

Upper bounds For obtaining the upper bounds stated in Theorem 5.3, we proceed in a similar fashion as we did for containment of guarded-based OMQs in Section 4.3. We first show the following result:

Theorem 5.4. *The problem* $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ *is decidable in* 2ExpTime *such that the second exponent on the runtime only depends on the width of the underlying schema. Hence,* $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ *is in* ExpTime *for predicates of bounded arity.*

We then use Theorem 5.4 for solving $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{UCQ})$ as follows:

(i) We first show that we can reduce $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{UCQ})$ to $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{UBCQ})$ in polynomial time.

(ii) Since Boolean UCQs correspond to frontier-guarded rules we can in turn reduce $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{UBCQ})$ to $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{BAQ})$.

(iii) As in the case of containment, in Section 5.5 we provide an exponential reduction from the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{BAQ})$ to $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$. This reduction again relies on the treeification technique introduced in Subsection 4.3.2. Although this reduction is exponential, it will, as in the case of containment, turn out that this reduction still yields a decision procedure for $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{BAQ})$ that runs in 2ExpTime.

For obtaining the ExpTime upper bound in item (iii) of Theorem 5.3 we simply reduce $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{AQ})$ to $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$.

First-order rewritability for the class (G, BAQ) The main focus of this chapter is thus on establishing Theorem 5.4. A first guess on how to prove that $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ is indeed in 2ExpTime, is to extend the methods developed in [34, 36] for languages from the $\mathcal{EL}$-family to the setting where the use of relation symbols of arity greater than two is allowed. However, as mentioned above and as we shall show in Section 5.2, this is not possible since we lose a crucial property of first-order rewritable OMQs when passing to the setting of higher-arity relations. Hence, the methods developed in [34, 36] do not seem to lead us to a solution for $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$.

Our plan of attack toward a proof of Theorem 5.4 can be summarized as follows:

▶ We provide a simple, yet useful, semantic criterion that characterizes the first-order rewritable OMQs within $(\mathsf{G}, \mathsf{BAQ})$. This characterization differs from the characterizations provided in [34, 36], as these refer to trees over a schema consisting of unary and binary predicates only, while ours is "coarser" in the sense that it accounts for more general structures. On the other hand, it is not immediate how this semantic characterization can be employed toward a decision procedure for $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$.

▶ Therefore, we refine the mentioned semantic characterization in Section 5.3 in order to make it accessible to tree automata techniques. In particular, we provide a decision

procedure for $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ that is based on two-way alternating parity automata. This procedure reduces $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ to the question whether the language accepted by a suitable automaton is finite. It turns out, however, that this procedure is not optimal as it runs in 3ExpTime. We nevertheless find this procedure to be appealing in the sense that it provides useful insights on the nature of the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$.

▶ We observe that the reason why the mentioned procedure based on classical automata techniques is not optimal is because classical automata lack a natural concept of attaching a quantitative measure to input structures. Moreover, the procedure from Section 5.3 crucially uses a minimization step in the construction of automata, and this minimization step turns out to be computationally expensive. We attack this problem by revising the semantic characterization devised in Section 5.2 so that it accounts for a quantitative measure assigned to input databases. Roughly speaking, given an OMQ Q and an input database $\mathfrak{D}$, we would like to measure the length of a minimal rewriting of Q into a CQ that witnesses that $\mathfrak{D} \models Q$. It turns out that if we can uniformly bound the length of such rewritings, we can conclude that Q is indeed first-order rewritable. To obtain a tight 2ExpTime upper bound, we then reduce $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ to the question whether (the cost function defined by) a certain cost automaton is uniformly bounded over all inputs. We then invoke a result from [31, 59] concerning the decidability of the boundedness problem for cost automata like ours. From this we obtain a 2ExpTime upper bound for $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ in the general case, and an ExpTime upper bound for the case of schemas of bounded width.

FO-rewritability vs. UCQ-rewritability Before diving into the technical details of this chapter, we want to elaborate on the relationship of first-order rewritability and UCQ-rewritability of rule-based OMQs. It is no coincidence that all OMQ languages we have encountered so far actually always exhibit UCQ-rewritings whenever they exhibit first-order rewritings. This is the content of the following lemma:

Lemma 5.5. *An OMQ Q from $(\mathsf{TGD}, \mathsf{UCQ})$ is first-order rewritable if and only if it is UCQ-rewritable.*

Proof. The "if" direction is immediate since every UCQ is obviously a first-order query. Suppose that $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ is first-order rewritable, with $\varphi_Q(\bar{x})$ being a first-order rewriting of it. Recall from Proposition 3.18 that Q is closed under homomorphisms, and thus so is φ. By the celebrated homomorphism preservation theorem (over finite structures) due to Rossman [123], it follows that $\varphi_Q(\bar{x})$ is equivalent to a UCQ $q'(\bar{x})$. □

Hence, deciding first-order rewritability for the languages of our interest amounts to deciding whether an according UCQ-rewriting exists. We will exploit this fact throughout this chapter without further citing the contents of Lemma 5.5.

Let us also point out that, throughout this chapter, we assume that all sets of TGDs and all queries are constant-free. The methods devised here can be extended to the case where constants are allowed in a similar fashion as in the case of containment among (frontier-)guarded OMQs (cf. Subsection 4.3.3).

5.2 Semantic Characterization

As mentioned in the previous section, our goal here is to state the semantic characterization of the first-order rewritable OMQs among $(\mathsf{G}, \mathsf{BAQ})$. This characterization is simple, yet quite useful for the decision procedures devised in this chapter. Before presenting it, let us before revisit the semantic characterizations provided in [34, 36] for OMQs formulated using description logics from the $\mathcal{EL}$-family.

Locality vs. FO-rewritability Let $Q = (\mathbf{S}, \mathcal{O}, A(x))$ be an OMQ where $A(x)$ is an atomic query and $\mathcal{O}$ is formulated using a member of the $\mathcal{EL}$-family. Roughly speaking, we can imagine that $\mathcal{O}$ is guarded and uses only unary and binary predicates (see [10, 11] for more details on description logics belonging to the $\mathcal{EL}$-family). The characterizations in [34, 36] essentially state that Q is first-order rewritable iff there is a uniform bound $k \geq 0$ such that, whenever the root of a tree-shaped $\mathbf{S}$-database $\mathfrak{D}$ is an answer to Q, then this already holds for the restriction of $\mathfrak{D}$ up to depth k. Notice that – since we are dealing here with databases formulated using unary and binary predicates only – the notion of *tree-shaped* can be taken literally in the sense of graphs (unlike in the case of arbitrary structures, where one resorts to tree decompositions to capture tree-likeliness).

Intuitively, the semantic characterizations from [34, 36] are based on a *locality* argument. To formalize this properly, let us provide some technical definitions. Given an $\mathbf{S}$-structure $\mathfrak{A}$, the *Gaifman graph of* $\mathfrak{A}$ is the undirected graph $\mathbb{G}(\mathfrak{A}) = (V, E)$, where $V \coloneqq \mathsf{dom}(\mathfrak{A})$ and

$$E \coloneqq \{\{a, b\} \mid a \neq b \text{ and there is an } R \in \mathbf{S} \text{ and a tuple } \bar{c} \in R^{\mathfrak{A}} \text{ such that } a, b \in [\bar{c}]\}.$$

Given elements $a, b \in \mathsf{dom}(\mathfrak{A})$, we write $d_{\mathfrak{A}}(a, b)$ for the distance between a and b in $\mathbb{G}(\mathfrak{A})$ provided they are reachable from each other (otherwise, we set $d_{\mathfrak{A}}(a, b) \coloneqq \infty$), and we call $d_{\mathfrak{A}}(a, b)$ the *Gaifman distance* between a and b. For a tuple $\bar{a} = a_1, \ldots, a_n$ over $\mathsf{dom}(\mathfrak{A})$ and $\ell \geq 0$, we let

$$B^{\mathfrak{A}}_{\ell}(\bar{a}) \coloneqq \{b \mid \min_{1 \leq i \leq n} d_{\mathfrak{A}}(a_i, b) \leq \ell\},$$
$$\mathfrak{B}^{\mathfrak{A}}_{\ell}(\bar{a}) \coloneqq \mathfrak{A} \restriction B^{\mathfrak{A}}_{\ell}(\bar{a}).$$

Hence, $\mathfrak{B}^{\mathfrak{A}}_{\ell}(\bar{a})$, called the *$\ell$-ball around $\bar{a}$*, is the substructure of $\mathfrak{A}$ induced by the set of elements consisting of those that have Gaifman distance at most ℓ from some a_i.

Now consider again the OMQ $Q = (\mathbf{S}, \mathcal{O}, A(x))$ from above, where $\mathcal{O}$ is formulated using a member of the $\mathcal{EL}$-family, and $A(x)$ is a unary atomic query. In [34, 36] it is shown that the following are equivalent:

(i) Q is first-order rewritable.

(ii) There exists a $k \geq 0$ such that, for every tree-shaped database $\mathfrak{D}$ with root a, if $\mathfrak{D} \models Q(a)$ then also $\mathfrak{B}^{\mathfrak{D}}_{k}(a) \models Q(a)$.

Let us call Q *local* if it satisfies the property stated in item (ii). The proof of the (contrapositive of the) "only if" direction of this statement establishes that, if there is no

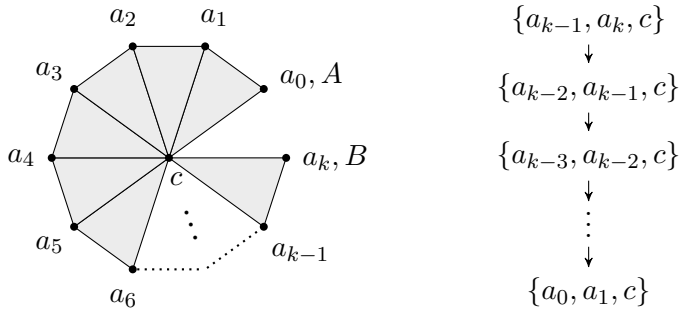

Figure 5.1: Illustration of the database $\mathfrak{D}_k$ from Example 5.6 (to the left), including a (guarded) tree decomposition of width two of $\mathfrak{D}_k$ (to the right).

such bound $k \geq 0$, then this is witnessed by an infinite sequence of deeper and deeper tree-shaped databases that establish the non-locality of Q. Moreover, it turns out that the locality of Q is beneficial when one aims to devise procedures for deciding first-order rewritability that are based on tree automata. Roughly speaking, the reason for this is that an automaton can easily keep track of the Gaifman distances of the elements required to satisfy Q.

When passing from OMQs that only use predicates of arity at most two to OMQs that use predicates of higher arity, one needs to resort to a different notion of tree-likeliness in order to make reasoning about these OMQs accessible to tree automata techniques. To wit, recall that in Section 4.3 we resorted to databases of bounded tree-width – where the bound on the tree-width only depended on the OMQ at hand – in order to devise tree automata procedures for OMQs from $(\mathsf{G}, \mathsf{AQ})$. A natural question arising in the context of first-order rewritability for $(\mathsf{G}, \mathsf{AQ})$ is whether the equivalence between locality and first-order rewritability also holds for OMQs from $(\mathsf{G}, \mathsf{AQ})$ when assuming that "being tree-like" stands for "being of bounded tree-width." Unfortunately, this is not the case, as the following example shows:

Example 5.6. Let $Q = (\mathbf{S}, \mathcal{O}, G(x))$ be from $(\mathsf{G}, \mathsf{AQ})$, where $\mathbf{S} = \{T/3, A/1, B/1\}$, and $\mathcal{O}$ consists of the rules

$$\begin{aligned} T(x,y,z), A(z) &\to R(x,z), \\ T(x,y,z), R(x,z) &\to R(x,y), \\ T(x,y,z), R(x,z), B(y) &\to G(y). \end{aligned}$$

We will prove below that Q is not first-order rewritable. For $k \geq 1$, consider the $\mathbf{S}$-database $\mathfrak{D}_k$ given by

$$\mathfrak{D}_k \coloneqq \{T(c, a_i, a_{i-1}) \mid i = 1, \ldots, k\} \cup \{A(a_0), B(a_k)\}.$$

Notice that $\mathfrak{D}_k$ is acyclic and of tree-width two. An illustration of $\mathfrak{D}_k$ together with a guarded tree decomposition of it can be found in Figure 5.1. Observe that $\mathfrak{D}_k \models Q(a_k)$.

We recursively define a family of formulas $\{\varphi_n\}_{n\geq 0}$ as follows:

$$\varphi_0(x,y,z) \coloneqq T(x,y,z) \wedge A(z),$$
$$\varphi_{n+1}(x,y,z) \coloneqq T(x,y,z) \wedge \exists v_n\, \varphi_n(x, y/z, z/v_n).$$

Let $\chi_n(y) \coloneqq \exists x, z\,(B(y) \wedge \varphi_n(x,y,z))$. It is not hard to check that, for all $n \geq 0$, χ_n is (equivalent to) a strictly guarded formula. Hence, by Lemma 4.24, χ_n is equivalent to a strictly acyclic query $p_n(y)$. Moreover, the infinite disjunction

$$q(y) \coloneqq \bigvee_{n\geq 0} p_n(y)$$

is easily seen to be equivalent to Q.

Suppose $\mathfrak{D}$ is an arbitrary $\mathbf{S}$-database such that $\mathfrak{D} \models Q(a)$ for some $a \in \mathsf{adom}(\mathfrak{D})$. Hence, $\mathfrak{D} \models p_k(a)$ for some $k \geq 0$, and thus there is a homomorphism h from $p_k(y)$ to $\mathfrak{D}$ with $h(y) = a$. Observe that p_k can be viewed just as the CQ corresponding to $\mathfrak{D}_k$, whence it follows that there is a weak homomorphism h_k from $\mathfrak{D}_k$ to $\mathfrak{D}$ such that $h_k(a_k) = a$. Moreover, $\mathfrak{B}_2^{\mathfrak{D}_k}(a_k)$ is isomorphic to $\mathfrak{D}_k$, and it is easy to prove that, by the construction of χ_k, $\mathfrak{B}_2^{\mathfrak{D}}(a) \models p_k(a)$ must hold as well.[2] Hence, $\mathfrak{B}_k^{\mathfrak{D}}(a) \models Q(a)$. This proves that Q is local but, as we shall prove below, Q is not first-order rewritable.[3] ⊣

Intuitively, Example 5.6 shows that locality notions used for first-order rewritability in the context of description logics from the $\mathcal{EL}$-family do not yield a sufficient condition for first-order rewritability for OMQs that fall into $(\mathsf{G}, \mathsf{AQ})$. This becomes clear when one inspects the tree decompositions of the databases $\mathfrak{D}_k$ depicted in Figure 5.1. Indeed, it turns out that, in terms of Gaifman distance, all the elements of $\mathfrak{D}_k$ are close to each other – the maximum Gaifman distance between any two elements equals two –, while any the underlying trees of suitable tree decompositions of $\mathfrak{D}_k$ have to grow in depth with increasing k.

Our semantic characterization We have seen above that a semantic characterization based on locality notions is unlikely to yield a useful characterization of the first-order rewritable OMQs from $(\mathsf{G}, \mathsf{AQ})$. Instead of enforcing a uniform bound on the Gaifman distance of the elements used to satisfy Q, we uniformly bound the number of facts required to satisfy Q.

This is the content of the following statement. In the following, given a schema $\mathbf{S}$, let us write $\mathsf{wd}^*(\mathbf{S})$ for $\max\{0, \mathsf{wd}(\mathbf{S}) - 1\}$.

Proposition 5.7. *Consider an OMQ $Q \in (\mathsf{G}, \mathsf{AQ})$ with data schema $\mathbf{S}$. The following are equivalent:*

(i) *Q is first-order rewritable.*

[2] Formally, one uses the fact here that p_k is a *connected* CQ [33, 34], i.e., the Gaifman graph of the structure corresponding to p_k consists of a single connected component.

[3] Notice that, in fact, we did not assume anything about the structure of $\mathfrak{D}$. Thus, Q is not only local for "tree-like" input databases, but local with respect to the class of all $\mathbf{S}$-databases.

(ii) *There is a* $k \geq 0$ *such that, for every* $\mathbf{S}$*-database* $\mathfrak{D}$ *of tree-width at most* $\mathsf{wd}^*(\mathbf{S})$*, if* $\mathfrak{D} \models Q(\bar{a})$*, then there is a* $\mathfrak{D}' \subseteq \mathfrak{D}$ *with at most* k *facts such that* $\mathfrak{D}' \models Q(\bar{a})$*.*

Proof. Assume first that $Q = (\mathbf{S}, \mathcal{O}, G(\bar{x}))$ is first-order -rewritable. Then there is a first-order query $\varphi_Q(\bar{x})$ which is equivalent over all $\mathbf{S}$-databases to Q. By Lemma 5.5 we actually know that Q is equivalent to a UBCQ $q_Q(\bar{x}) = \bigvee_{i=1}^{n} p_i(\bar{x})$. Let $k \coloneqq \max\{|p_i| : i = 1, \ldots, n\}$. We claim that k is the bound we are looking for in item (ii). Indeed, if $\mathfrak{D} \models Q(\bar{a})$, for a database of tree-width at most $\mathsf{wd}^*(\mathbf{S})$, then also $\mathfrak{D} \models q_Q(\bar{a})$ and so there is a homomorphism h that maps some p_i to $\mathfrak{D}$ such that $h(\bar{x}) = \bar{a}$. The image of p_i under h is a database $\mathfrak{D}'$ with at most k facts that satisfies q_Q, i.e., $\mathfrak{D}' \models q_Q(\bar{a})$. Since Q is equivalent to q_Q, we infer that $\mathfrak{D}' \models Q(\bar{a})$, as required.

Suppose now that there is a $k \geq 0$ such that, for every $\mathbf{S}$-database $\mathfrak{D}$ of tree-width at most $\mathsf{wd}^*(\mathbf{S})$, if $\mathfrak{D} \models Q(\bar{a})$, then there is a $\mathfrak{D}' \subseteq \mathfrak{D}$ with at most k facts such that $\mathfrak{D}' \models Q(\bar{a})$. Let $\mathcal{S}$ be the class of all tuples $(\mathfrak{D}, \bar{a})$, such that

(i) $\mathfrak{D}$ is an $\mathbf{S}$-database and $\bar{a}$ a tuple over $\mathsf{adom}(\mathfrak{D})$ of length $|\bar{x}|$,

(ii) $\mathfrak{D}$ contains at most k facts, and

(iii) $\mathfrak{D} \models Q(\bar{a})$.

Let us say that two tuples $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}, \bar{b})$ are *isomorphic* if there exists a weak isomorphism χ from $\mathfrak{D}$ to $\mathfrak{D}'$ such that $\chi(\bar{a}) = \bar{b}$. Consider $\mathcal{S}$ factorized modulo isomorphisms, and observe that $\mathcal{S}$ is thus finite. Let

$$q_Q \coloneqq \bigvee_{(\mathfrak{D}, \bar{a}) \in \mathcal{S}} q_{\mathfrak{D}, \bar{a}},$$

where $q_{\mathfrak{D}, \bar{a}}$ is the CQ that is obtained by simultaneously replacing each constant c in the database $\mathfrak{D}$ with the variable x_c, and existentially quantifying over all x_c for which $c \notin [\bar{a}]$ holds. We claim that q_Q is a UCQ equivalent to Q, and thus a first-order rewriting of Q.

Suppose first that $\mathfrak{D} \models Q(\bar{a})$ for some $\mathbf{S}$-database $\mathfrak{D}$. By Lemma 4.36, there is an acyclic $\mathbf{S}$-database $\mathfrak{B}$ – hence it has tree-width at most $\mathsf{wd}^*(\mathbf{S})$ – such that $\mathfrak{B} \models Q(\bar{a})$ and there is a weak homomorphism h from $\mathfrak{B}$ to $\mathfrak{D}$ that is the identity on $\bar{a}$. By assumption, there is a database $\mathfrak{D}' \subseteq \mathfrak{B}$ with at most k facts such that $\mathfrak{D}' \models Q(\bar{a})$. It follows that some isomorphic representative of $(\mathfrak{D}', \bar{a})$ is contained in $\mathcal{S}$. Therefore, $\mathfrak{D}' \models q_Q(\bar{a})$ and, since $\mathfrak{D}' \subseteq \mathfrak{B}$, also $\mathfrak{B} \models q_Q(\bar{a})$, i.e., there is a homomorphism h_Q from some CQ p of q_Q to $\mathfrak{B}$ that maps the output variables of p to $\bar{a}$. Therefore, $h_Q \circ h$ is a homomorphism from p to $\mathfrak{D}$ mapping the output variables of p to $\bar{a}$, from which we conclude that $\mathfrak{D} \models q_Q(\bar{a})$.

Suppose now that $\mathfrak{D} \models q_Q(\bar{a})$. Then there is some $(\mathfrak{D}', \bar{a}) \in \mathcal{S}$ such that $\mathfrak{D} \models q_{\mathfrak{D}', \bar{a}}(\bar{a})$, i.e., there is a homomorphism from $q_{\mathfrak{D}', \bar{a}}$ to $\mathfrak{D}$ that maps the output variables of $q_{\mathfrak{D}', \bar{a}}$ to $\bar{a}$, or, equivalently, a weak homomorphism from $\mathfrak{D}'$ to $\mathfrak{D}$ that is the identity on $\bar{a}$. By the construction of $\mathcal{S}$, it is also the case that $\mathfrak{D}' \models Q(\bar{a})$. Since Q is closed under (weak) homomorphisms (cf. Proposition 3.18), we obtain $\mathfrak{D} \models Q(\bar{a})$ as desired. □

Example 5.8. Reconsider the OMQ $Q = (\mathbf{S}, \mathcal{O}, G(x))$ from Example 5.6, and, for $k \geq 1$, let $\mathfrak{D}_k$ denote the $\mathbf{S}$-database from Example 5.6. As promised before, we now argue that Q is indeed not first-order rewritable. To this end, recall that $\mathfrak{D}_k \models Q(a_k)$, and notice

that $\mathfrak{D}_k$ has tree-width two (in fact, it is acyclic), and that $\mathfrak{D}_k$ consists of $k+2$ facts. However, for every $\mathfrak{D}' \subset \mathfrak{D}$ that has at most $k+1$ facts we clearly have $\mathfrak{D}' \not\models Q(a_k)$. Hence, by Proposition 5.7, we can conclude that Q is not first-order rewritable. ⊣

Example 5.9. Reconsider now the OMQ $Q' = (\{A/1, E/2\}, \mathcal{O}', B(x))$ from Example 5.2, where $\mathcal{O}'$ consists of the two rules

$$A(x) \to B(x),$$
$$B(x), E(x,y) \to B(y).$$

For $k \geq 0$, let

$$\mathfrak{D}_k \coloneqq \{A(a_0), E(a_0,a_1), \ldots, E(a_{k-1}, a_k)\}.$$

Obviously, $\mathfrak{D}_k \models Q'(a_k)$ for every $k \geq 0$. Notice further that $\mathfrak{D}_k$ has tree-width one, as it is simply a directed path. Now $\mathfrak{D}_k$ consists of $k+1$ atoms, but for every $\mathfrak{D}' \subset \mathfrak{D}$ of at most k atoms, it holds that $\mathfrak{D}' \not\models Q'(a_k)$, which by Proposition 5.7 entails that Q' is not first-order rewritable. ⊣

5.3 Alternating Automata Approach

In this section, we exploit the semantic characterization from Section 5.2 and 2APTA running on finite trees of bounded branching degree to prove that $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ can be solved in elementary time, more specifically, in 3ExpTime. Although our result is not optimal in terms of complexity, our construction provides a transparent solution to $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ based on standard tools. This is in contrast to existing results on guarded logics, in which all elementary bounds heavily rely on the intricate use of results on cost automata [31, 38]. We are also going to employ cost automata later, but only to pinpoint the exact complexity of $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$.

Notice that the decidability of $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ in elementary time already follows from the fact that OMQs from $(\mathsf{G}, \mathsf{BAQ})$ can be equivalently expressed as guarded Datalog queries of elementary size, while guarded Datalog queries can in turn be expressed using sentences of GFP [31]. Since boundedness for GFP is decidable in 2ExpTime [19, 31], and boundedness of guarded Datalog queries is equivalent to their first-order rewritability, an elementary upper bound follows. Again, the obtained upper bound is far from optimal, and we prefer here to provide an alternative procedure which, though not optimal, yields interesting insights into the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ from a conceptual point of view.

The main result of this section reads as follows:

Theorem 5.10. *The problem* $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ *is solvable in* 3ExpTime*, and in* 2ExpTime *if we assume predicates of bounded arity.*

The main lemma toward a proof of Theorem 5.10 will be the following:

LEMMA 5.11. *Let $Q = (\mathbf{S}, \mathcal{O}, G)$ be an OMQ from $(\mathsf{G}, \mathsf{BAQ})$. There is an m-2APTA $\mathcal{B}_Q$ on trees of branching degree at most $m := 2^{\mathsf{wd}(\mathbf{S})}$ such that*

$$Q \text{ is first-order rewritable} \iff \mathcal{L}(\mathcal{B}_Q) \text{ is finite.}$$

The state set of $\mathcal{B}_Q$ is of doubly exponential size in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$, and of exponential size in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. Furthermore, $\mathcal{B}_Q$ can be constructed in triply exponential time in the size of Q, where the third exponent depends only on $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$.

Before explaining how to establish Lemma 5.11, let us show how to infer Theorem 5.10 from Lemma 5.11.

PROOF OF THEOREM 5.10. Let $Q = (\mathbf{S}, \mathcal{O}, G)$ be an OMQ from $(\mathsf{G}, \mathsf{BAQ})$, and let $\mathcal{B}_Q$ be the m-2APTA as stated in Lemma 5.11. By Theorem 2.28, there is a 1NTA $\mathcal{B}'_Q$ that accepts the same trees as $\mathcal{B}_Q$, and whose number of states is exponential in the number of states of $\mathcal{B}_Q$, i.e., triply exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$, and doubly exponential in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. Moreover, $\mathcal{B}'_Q$ can be constructed in exponential time in the size of $\mathcal{B}_Q$. In [131] it is shown that the problem of deciding whether a 1NTA accepts a finite language is feasible in polynomial time. Thus, we simply check whether $\mathcal{L}(\mathcal{B}'_Q)$ is finite, whence the claimed upper bounds stated in Theorem 5.10 follow. □

The remainder of this section is devoted to establish Lemma 5.11. To this end, we proceed as follows:

▶ We provide a semantic characterization of the first-order rewritable OMQs in $(\mathsf{G}, \mathsf{BAQ})$ that refines Proposition 5.7. Intuitively, this semantic characterization encodes some minimality criterion for the class of all input databases of bounded tree-width.

▶ We then devise an m-2APTA that only accepts (encodings of) input databases that are feasible candidates for the "minimal" databases, as singled out by the refined semantic characterization.

▶ To actually implement the minimality criterion, we use projection on m-2APTA. The use of projection explains why the upper bounds stated in Theorem 5.10 are not optimal – indeed, we will see that projection is applied upon an m-2APTA whose state set is already exponential, and projection incurs an additional exponential blowup to its state set (cf. Proposition 2.32).

5.3.1 A Refined Semantic Characterization

For introducing a semantic characterization that refines the one given in Proposition 5.7, we need some additional technical notions.

Let $\mathfrak{D}$ be an $\mathbf{S}$-database and $\delta = (\mathcal{T}, (X)_{v \in T})$ be a tree decomposition of $\mathfrak{D}$. Recall that an *adornment* of the pair $(\mathfrak{D}, \delta)$ is a function $\eta \colon T \to 2^{\mathfrak{D}}$ such that

(i) $\eta(v) \subseteq \mathfrak{D} \restriction X_v$ for all $v \in T$, and

(ii) for every fact α of $\mathfrak{D}$, there is some $v \in T$ such that $\alpha \in \eta(v)$.

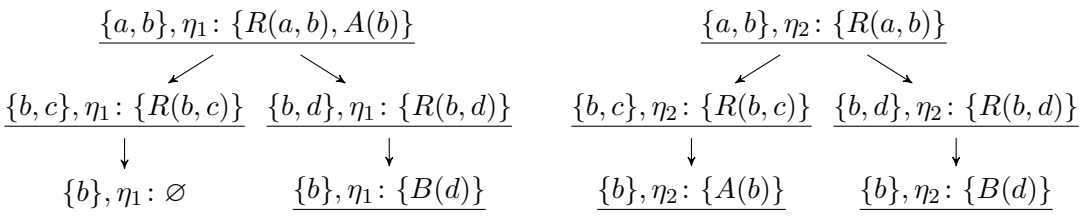

Figure 5.2: Both figures depict a tree decomposition together with an adornment η_i whose values are specified for each node of the respective tree decomposition. For a node v, a label of the form $X_v, \eta_i \colon Y_v$ specifies that X_v is the bag of v in the tree decomposition at hand, and that $\eta_i(v) = Y_v$. Nodes that are black with respect to η_i are underlined. The tree decomposition in (a) is neither η_1-well-colored nor η_1-simple, while the one in (b) is both η_2-well-colored and η_2-simple.

Hence, the pair (δ, η) can be viewed as a representation of the database $\mathfrak{D}$ along with a tree decomposition of it. For the intended characterization, it is important that this representation is free of redundancies, which we formalize in the following.

We say that δ is *η-simple* if the following conditions hold:

(i) $|\eta(v)| \leq 1$ for all $v \in T$.

(ii) Non-empty η-labels are unique, that is, $\eta(v) \neq \eta(w)$ for all distinct $v, w \in T$ with $\eta(v)$ and $\eta(w)$ non-empty.

Nodes $v \in T$ where $\eta(v)$ is empty are called *white* (w.r.t. η) from now on. These white nodes are required since we might not have a (unique!) fact available to label them. Note, though, that white nodes v are still associated with a non-empty set of constants from $\mathfrak{D}$ via X_v. All nodes that are not white are called *black* (w.r.t. η). While δ being η-simple avoids redundancies that are due to a fact occurring in the label of multiple black nodes, additional redundancies may arise from the inflationary use of white nodes.

We say that a node $v \in T$ is *η-well-colored* if it is black, or it has at least two successors and all its successors are η-well-colored. We say that δ is *η-well-colored* if every node in T is η-well-colored. For example, δ is not η-well-colored if it has a white leaf, or if it has a white node and its single successor is also white. Informally, requiring δ to be η-well-colored makes it impossible to blow up the tree by introducing white nodes without introducing black nodes.

Example 5.12. Consider the database

$$\mathfrak{D} \coloneqq \{R(a, b), A(b), R(b, c), R(b, d), B(d)\}.$$

In Figure 5.2 we depict two tree decompositions of $\mathfrak{D}$ (of width one) together with adornments η_i ($i \in \{1, 2\}$) for them. The first one is neither η_1-simple nor η_1-well-colored, while the second one is both η_2-simple and η_2-well-colored. ⊣

For $i \in \{1, 2\}$, let $\mathfrak{D}_i$ be a database, δ_i a tree decomposition of $\mathfrak{D}_i$, and η_i an adornment of $(\mathfrak{D}_i, \delta_i)$. We say that $(\mathfrak{D}_1, \delta_1, \eta_1)$ and $(\mathfrak{D}_2, \delta_2, \eta_2)$ are *isomorphic* if the latter can be obtained from the former by consistently renaming constants in $\mathfrak{D}_1$ and tree nodes in δ_1.

We are now ready to refine the characterization of first-order rewritability for OMQs from $(\mathsf{G}, \mathsf{BAQ})$ given in Proposition 5.7:

PROPOSITION 5.13. *Consider an OMQ $Q \in (\mathsf{G}, \mathsf{BAQ})$ with data schema $\mathbf{S}$. The following are equivalent:*

(i) *Q is first-order rewritable.*

(ii) *There are finitely many non-isomorphic triples $(\mathfrak{D}, \delta, \eta)$, where $\mathfrak{D}$ is an $\mathbf{S}$-database, δ a tree decomposition of $\mathfrak{D}$ of width at most $\mathsf{wd}^*(\mathbf{S})$, and η an adornment of $(\mathfrak{D}, \delta)$, such that*

  (a) *δ is η-simple and η-well-colored,*

  (b) *$\mathfrak{D} \models Q$, and*

  (c) *for every $\alpha \in \mathfrak{D}$, it is the case that $\mathfrak{D} \setminus \{\alpha\} \not\models Q$.*

The proof of Proposition 5.13 depends on several lemmas stated in the following.

LEMMA 5.14. *If $\mathfrak{D}$ has tree-width w, then there is a tree decomposition δ of $\mathfrak{D}$ of width w and an adornment η for $(\mathfrak{D}, \delta)$ such that δ is η-simple and η-well-colored.*

PROOF. Let $\delta = (\mathcal{T}, (X_v)_{v \in T})$ be a tree decomposition of $\mathfrak{D}$ of width w. Let $\delta' = (\mathcal{T}', (X'_v)_{v \in T'})$ be a tree decomposition of $\mathfrak{D}$ defined as follows. Initially, we define that δ' equals δ. In a second step, we add additional nodes to δ'. For any $v \in T$, if $|\mathfrak{D} \restriction X_v| = n$ (for $n \geq 2$) then we add $n - 1$ copies of v to δ' that become children of v in δ'. Let $v \in T$ and suppose $\mathfrak{D} \restriction X_v = \{\alpha_1, \ldots, \alpha_n\}$. Let $v_1, \ldots, v_{n-1}$ be the copies of v in δ'. We then set $\eta(v) \coloneqq \{\alpha_1\}$, and $\eta(v_i) \coloneqq \{\alpha_{i+1}\}$ for $i = 1, \ldots, n-1$. Clearly, η is an adornment for $(\mathfrak{D}, \delta')$ satisfying $|\eta(v)| = 1$, for all $v \in T'$. By construction, observe that δ' is also η-simple.

Now we show how we can modify δ' in order to become η-well-colored. Let B denote the set of black nodes of $\mathcal{T}'$. Let T^* be the smallest set such that

(i) $B \subseteq T^*$, and

(ii) if v is the greatest common ancestor of some $T_0 \subseteq T^*$, then also $v \in T^*$.

Hence, T^* is the closure of B under greatest common ancestors. Let $\delta^* \coloneqq (\mathcal{T}^*, (Y_v)_{v \in T^*})$, where $\mathcal{T}^*$ is a tree with node set T^* and, for $v, w \in T^*$, we have that w is a successor of v in $\mathcal{T}^*$ iff $v \prec_{\mathcal{T}'} w$. Notice that δ^* is a tree decomposition of $\mathfrak{D}$ that has width w, as it contains all black nodes of T' with respect to η. It is now easy to check that δ^* is also η-well-colored. □

LEMMA 5.15. *Suppose η is an adornment for $(\mathfrak{D}, \delta)$ and that δ is η-simple. Then $\mathfrak{D}$ contains at least as many facts as δ contains black nodes with respect to η.*

PROOF. By induction on the number n of black nodes of $\delta = (\mathcal{T}, (X_v)_{v \in T})$ with respect

to η. If $n = 1$ the claim is trivial. Suppose δ has $n + 1$ black nodes with respect to η. There must be a black node $v \in T$ such that v has no descendant that is also black. Let $\delta' = (\mathcal{T}', (X_v)_{v \in T'})$ be the tree decomposition that arises from δ by removing the subtree rooted at v. Let $\mathfrak{D}' \coloneqq \bigcup_{v \in T'} \eta(v)$. Then δ' is a tree decomposition of $\mathfrak{D}'$ and δ' has n black nodes with respect to η. Now, if $\mathfrak{D}' = \mathfrak{D}$ then it must be the case that $\eta(v) = \eta(w)$ for some $w \neq v$. Hence, δ cannot be simple. Therefore, $\mathfrak{D}' \subset \mathfrak{D}$. By the induction hypothesis, $|\mathfrak{D}'| \geq n$ and we thus obtain $|\mathfrak{D}| \geq n + 1$. □

Lemma 5.16. *Suppose $\mathfrak{D}$ is a database that has at least one fact. If δ is an η-well-colored tree decomposition of $\mathfrak{D}$, then the number of white nodes of δ is strictly less than the number of black nodes of δ with respect to η.*

Proof. Let b_δ (w_δ, respectively) denote the number of black (white, respectively) nodes of $\delta = (\mathcal{T}, (X_v)_{v \in T})$ with respect to η. We proceed by induction on the depth of $\mathcal{T}$, i.e., the maximum length of a branch leading from the root node to a leaf node. If $\mathcal{T}$ consists only of a single node and δ is η-well-colored, this single node must be a black node – recall that $\mathfrak{D}$ has at least one fact – and so the claim holds trivially. Assume that $\mathcal{T}$ is of depth $n + 1$ and δ is η-well-colored. Let $\mathcal{T}_1, \ldots, \mathcal{T}_k$ enumerate the subtrees of $\mathcal{T}$ rooted at the child nodes of the root of $\mathcal{T}$, and let δ_i ($i = 1, \ldots, k$) be the tree decomposition that arises from δ if we restrict $\mathcal{T}$ to $\mathcal{T}_i$. If the root of $\mathcal{T}$ is black, the claim is again trivial. Otherwise, if it is white, we see that $k \geq 2$ since δ is η-well-colored. For $i = 1, \ldots, k$, let b_{δ_i} (w_{δ_i}, respectively) denote the number of black (white, respectively) nodes of $\mathcal{T}_i$ with respect to η. Using the induction hypothesis, we conclude that $w_\delta = w_{\delta_1} + \cdots + w_{\delta_k} + 1 < b_{\delta_1} + \cdots + b_{\delta_k} = b_\delta$. □

Proof of Proposition 5.13. Assume that item (ii) does not hold. That is, there are infinitely many non-isomorphic triples $(\mathfrak{D}, \delta, \eta)$ that satisfy conditions (a) to (c). Let $\mathcal{S}$ be the set of all these triples and let $\mathcal{S}'$ be $\mathcal{S}$ factorized modulo our notion of isomorphism, i.e., $\mathcal{S}'$ contains a representative for every isomorphism type of $\mathcal{S}$. Let

$$\Phi \coloneqq \{\mathcal{T} \mid (\mathfrak{D}, \delta \coloneqq (\mathcal{T}, (X_v)_{v \in T}), \eta) \in \mathcal{S}'\}$$

be the set of trees on which the tree decompositions of the elements of $\mathcal{S}'$ are based, and consider Φ factorized modulo usual tree isomorphism. Notice that Φ must be infinite as well. Since all the tree decompositions on which the trees in Φ are based on are of bounded width, it must be that case that, for every $k \geq 0$, Φ contains a tree that has at least k nodes. Thus, by Lemma 5.16, for every $k \geq 0$, we can find a $(\mathfrak{D}_k, \delta_k, \eta_k) \in \mathcal{S}'$, where $\delta_k = (\mathcal{T}_k, (X_v)_{v \in T_k})$, such that $\mathcal{T}_k$ has at least k black nodes with respect to η_k. Thus, by Lemma 5.15, $\mathfrak{D}_k$ has at least k facts. Now $\mathfrak{D}_k \models Q$ by assumption, but $\mathfrak{D}_0 \not\models Q$ for every $\mathfrak{D}_0 \subset \mathfrak{D}_k$. This shows that, for every k, we can find a database $\mathfrak{D}$ of tree-width at most $\mathsf{wd}^*(\mathbf{S})$ (namely, $\mathfrak{D}_k$) such that $\mathfrak{D} \models Q$, but for every $\mathfrak{D}_0 \subseteq \mathfrak{D}$ of at most k atoms we have $\mathfrak{D}_0 \not\models Q$. Hence, item (i) does not hold.

Suppose now that Q is not first-order rewritable. That is, by Proposition 5.7, for every $k \geq 0$, there is a database $\mathfrak{D}_k$ of tree-width at most $\mathsf{wd}^*(\mathbf{S})$ such that $\mathfrak{D} \models Q$, yet for every $\mathfrak{D}_0 \subset \mathfrak{D}$ with at most k facts we have $\mathfrak{D}_0 \not\models Q$. Let $\mathcal{S}$ be the set of all

$\mathbf{S}$-databases $\mathfrak{D}$ of tree-width at most $\mathsf{wd}^*(\mathbf{S})$ such that $\mathfrak{D} \models Q$, yet for any $\mathfrak{D}_0 \subset \mathfrak{D}$ we have $\mathfrak{D}_0 \not\models Q$. Let $\mathcal{S}'$ be $\mathcal{S}$ factorized modulo weak isomorphisms. It is easy to check that $\mathcal{S}'$ must be infinite as well (cf. the proof of Proposition 5.7 for a similar argument). By Lemma 5.14, for every $\mathfrak{D} \in \mathcal{S}'$, there is a tree decomposition $\delta_{\mathfrak{D}}$ and an adornment $\eta_{\mathfrak{D}}$ of $(\mathfrak{D}, \delta_{\mathfrak{D}})$ such that $\delta_{\mathfrak{D}}$ is $\eta_{\mathfrak{D}}$-well-colored and $\eta_{\mathfrak{D}}$-simple. Now for two distinct $\mathfrak{D}, \mathfrak{D}' \in \mathcal{S}'$ it must be the case that $(\mathfrak{D}, \delta_{\mathfrak{D}}, \eta_{\mathfrak{D}})$ and $(\mathfrak{D}', \delta_{\mathfrak{D}'}, \eta_{\mathfrak{D}'})$ are non-isomorphic, for otherwise $\mathfrak{D}$ and $\mathfrak{D}'$ would be weakly isomorphic as well, which is absurd as we assume that $\mathcal{S}'$ is factorized modulo weak isomorphisms. For $\mathfrak{D} \in \mathcal{S}'$ we know that, by the construction of $\mathcal{S}'$, $\mathfrak{D} \setminus \{\alpha\} \not\models Q$ for all $\alpha \in \mathfrak{D}$. Hence, the class

$$\{(\mathfrak{D}, \delta_{\mathfrak{D}}, \eta_{\mathfrak{D}}) \mid \mathfrak{D} \in \mathcal{S}'\}$$

is a class of infinitely many, pairwise non-isomorphic triples such that conditions (a) to (c) of item (ii) are satisfied. Thus, item (ii) does not hold either. □

5.3.2 Devising Automata

Let us outline how the 2APTA announced in Lemma 5.11 is constructed. Throughout this subsection, we fix an OMQ $Q = (\mathbf{S}, \mathcal{O}, G)$ from $(\mathsf{G}, \mathsf{BAQ})$. Our goal is to devise an automaton $\mathcal{B}_Q$ whose language is finite iff item (ii) from Proposition 5.13 is satisfied. By Propositions 5.7 and 5.13, Q is then first-order rewritable iff $\mathcal{L}(\mathcal{B}_Q)$ is finite.

The 2APTA $\mathcal{B}_Q$ will be the intersection of several automata that check the properties stated in item (ii) of Proposition 5.13. As in the case of Section 4.3, our automata will work on $\Gamma_{\mathbf{S}}$-labeled trees, and we point the reader to Section 4.3 for the definition of $\Gamma_{\mathbf{S}}$. For our automata constructions that follow, it will be convenient to work on $\Gamma_{\mathbf{S}}$-labeled trees whose branching degree can be bounded by the constant $m_{\mathbf{S}} \coloneqq 2^{\mathsf{wd}(\mathbf{S})}$, so that we can work with $m_{\mathbf{S}}$-2APTA. The following statement shows that we can always assume this without loss of generality.

Lemma 5.17. *Suppose $\mathfrak{D}$ is an $\mathbf{S}$-database and δ a tree decomposition of $\mathfrak{D}$. Then there exists a tree decomposition δ' of $\mathfrak{D}$ such that δ' has the same width as δ, yet the branching degree of the tree δ' is based on is at most $m_{\mathbf{S}} = 2^{\mathsf{wd}(\mathbf{S})}$.*

Proof. Let $\delta = (\mathcal{T}, (X_v)_{v \in T})$ be a tree decomposition of $\mathfrak{D}$ of width at most $\mathsf{wd}^*(\mathbf{S})$. For $v \in T$, let $b_{\delta,v}$ be the branching degree of v in $\mathcal{T}$ and let $d_{\delta,v}$ be the length of the longest path that leads from v to some leaf node in $\mathcal{T}$. Moreover, let

$$b_\delta \coloneqq \sum \{d_{\delta,v}(b_{\delta,v} - m_{\mathbf{S}}) \mid b_{\delta,v} > m_{\mathbf{S}}, v \in T\}.$$

We are going to prove the statement of the lemma by induction on b_δ. Moreover, we will show that δ' results from "reorganizing" nodes of δ, and that we can view any adornment η of $(\mathfrak{D}, \delta)$ also as an adornment of $(\mathfrak{D}, \delta')$.

For $b_\delta = 0$ the claim holds trivially. Suppose therefore that $b_\delta = n + 1$. Then there is a node $v \in T$ such that $b_v > m_{\mathbf{S}}$. Assume v is chosen such that it has, among all nodes of branching degree greater than $m_{\mathbf{S}}$, maximal distance to the root. Let $v_1, \ldots, v_k$ enumerate all children of v, and assume for the sake of simplicity that $k = m_{\mathbf{S}} + 1$ –

the case where $k > m_{\mathbf{S}} + 1$ is treated analogously. For $i = 1, \dots, k$, let $Y_i \coloneqq X_v \cap X_{v_i}$. Hence, $Y_i \subseteq X_v$, and since there are at most $m_{\mathbf{S}} = 2^{\mathsf{wd}(\mathbf{S})}$ subsets of X_v, it must be the case that $Y_i = Y_j$ for some $i \neq j$. Let δ' be the tree decomposition that arises from δ by removing the subtree rooted at v_j from $\mathcal{T}$, while inserting it below v_i so that v_j becomes a child node of v_i. Notice that, in δ', v_i has branching degree at most $m_{\mathbf{S}} + 1$ by the choice of v. Moreover, δ' is still a tree decomposition of $\mathfrak{D}$ of width at most $\mathsf{wd}^*(\mathbf{S})$, since connectivity is clearly ensured. Observe that

$$\begin{aligned}
b_{\delta'} &= \sum\{d_{\delta',w}(b_{\delta',w} - m_{\mathbf{S}}) \mid b_{\delta',w} > m_{\mathbf{S}}, w \in T\} \\
&\leq \sum\{d_{\delta,w}(b_{\delta,w} - m_{\mathbf{S}}) \mid b_{\delta,w} > m_{\mathbf{S}}, w \in T \setminus \{v, v_i, v_j\}\} + d_{\delta',v_i}(b_{\delta',v_i} - m_{\mathbf{S}}) \\
&< \sum\{d_{\delta,w}(b_{\delta,w} - m_{\mathbf{S}}) \mid b_{\delta,w} > m_{\mathbf{S}}, w \in T \setminus \{v, v_i, v_j\}\} + d_{\delta,v}(b_{\delta,v} - m_{\mathbf{S}}) \\
&= b_{\delta}.
\end{aligned}$$

(Notice that $d_{\delta',v_i} < d_{\delta,v}$ and $b_{\delta',v_i} \leq b_{\delta,v}$.) Now an application of the induction hypothesis yields the claim. □

By Lemma 5.17 we can thus always assume that the encoding of an $\mathbf{S}$-database together with an associated tree decomposition and an adornment has branching degree at most $m_{\mathbf{S}} = 2^{\mathsf{wd}(\mathbf{S})}$.

We now proceed with the automata constructions to prove Lemma 5.11. Firstly, we would like to recall the automaton $\mathcal{C}_{\mathbf{S}}$ from Lemma 4.41:

Lemma 5.18. *There is an $m_{\mathbf{S}}$-2APTA $\mathcal{C}_{\mathbf{S}}$ that accepts an $m_{\mathbf{S}}$-ary $\Gamma_{\mathbf{S}}$-labeled tree t iff t is consistent. The number of states of $\mathcal{C}_{\mathbf{S}}$ is constant. Moreover, $\mathcal{C}_{\mathbf{S}}$ can be constructed in polynomial time in the size of $\Gamma_{\mathbf{S}}$.*

Proof. The automaton $\mathcal{C}_{\mathbf{S}}$ is equally constructed as the automaton from Lemma 4.41. Actually, the only difference between $\mathcal{C}_{\mathbf{S}}$ and the one from Lemma 4.41 is the fact that the $\mathcal{C}_{\mathbf{S}}$ constructed here works on $m_{\mathbf{S}}$-ary trees instead of amorphous trees. However, the construction proceeds similar to the one of Lemma 4.41 and we omit details. □

Just as a version of $\mathcal{C}_{\mathbf{S}}$ that works on $m_{\mathbf{S}}$-ary trees can be constructed, the same holds accordingly for the automaton $\mathcal{A}_Q$ from Lemma 4.42:

Lemma 5.19. *There is an $m_{\mathbf{S}}$-2APTA $\mathcal{A}_Q$ that accepts a consistent $\Gamma_{\mathbf{S}}$-labeled tree t iff $[\![t]\!] \models Q$. The number of states of $\mathcal{A}_Q$ is exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and linear in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. We can construct $\mathcal{A}_Q$ in doubly exponential time in the size of Q, and the second exponent of the runtime of this construction only depends on $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$.*

For a $\Gamma_{\mathbf{S}}$-labeled tree t, recall that δ_t denotes the standard tree decomposition of t, and η_t denotes the standard adornment of t. Since t thus incorporates the information about a tree decomposition and an adornment, the notions of being well-colored and simple can be naturally defined for t as well. Formally, we say that t is *simple* (respectively, *well-colored*) if δ_t is η_t-simple (respectively, η_t-well-colored).

LEMMA 5.20. *There is am $m_{\mathbf{S}}$-2APTA $\mathcal{R}_{\mathbf{S}}$ that accepts a consistent $\Gamma_{\mathbf{S}}$-labeled tree iff it is well-colored and simple. The number of states of $\mathcal{R}_{\mathbf{S}}$ is exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and linear in $|\mathbf{S}|$. Moreover, $\mathcal{R}_{\mathbf{S}}$ can be constructed in polynomial time in the size of $\Gamma_{\mathbf{S}}$.*

PROOF. We can devise $\mathcal{R}_{\mathbf{S}}$ as the intersection of two separate $m_{\mathbf{S}}$-2APTA, $\mathcal{R}_{1,\mathbf{S}}$ and $\mathcal{R}_{2,\mathbf{S}}$, where the former checks whether the input tree t is simple and the latter checks whether t is well-colored. Recall from Proposition 2.31 that building the intersection of two $m_{\mathbf{S}}$-2APTAs is feasible in polynomial time.

The automaton $\mathcal{R}_{1,\mathbf{S}}$. Let us first explain how the automaton $\mathcal{R}_{1,\mathbf{S}}$ is constructed. In order to check whether t is simple, we have to check two conditions:

(i) $|\eta_t(v)| \leq 1$, for each node v in t, and

(ii) $\eta_t(v) \neq \eta_t(w)$, for all black nodes v, w in t with $v \neq w$.

The first condition is easy to check with an $m_{\mathbf{S}}$-2APTA (respecting the stated size bounds) in a single top-down pass. We describe how the check the second one, assuming that the input tree satisfies the first condition – the final automaton $\mathcal{R}_{1,\mathbf{S}}$ is again the intersection of the two separate automata.

We shall describe the game $\mathcal{G}(\mathcal{R}_{1,\mathbf{S}}, t)$. Adam will have a winning strategy in the game $\mathcal{G}(\mathcal{R}_{1,\mathbf{S}}, t)$ iff $\eta_t(v) = \eta_t(w)$ for some black nodes $v \neq w$. Adam first navigates to an arbitrary black node v for which he wants to prove that there is some other $w \neq v$ such that $\eta_t(v) = \eta_t(w)$. He then selects the one and only atom $R_{\bar{a}} \in t(v)$ and guesses the path to the node w for which he thinks that $\eta_t(v) = \eta_t(w)$. While navigating to w, Adam must ensure that the tuple $\bar{a}$ appears at all nodes along his path of navigation, for otherwise, the label $R_{\bar{a}}$ he finds will denote an atom distinct from the one in $\eta_t(v)$. If he finds such a node, he wins. During navigation to w, he must remember the atom $R_{\bar{a}}$ in the states and also the last direction he came from. He must remember the directional information due to the fact that we require $v \neq w$. Thus, the number of states of $\mathcal{R}_{1,\mathbf{S}}$ also depends linearly on the branching degree $m_{\mathbf{S}}$, which still allows us the respect the stated size bounds as $m_{\mathbf{S}} = 2^{\mathsf{wd}(\mathbf{S})}$. While navigating to w, Adam is not allowed to traverse the tree backwards in the direction he came from. For storing this information, we need exponentially many states in $\mathsf{wd}(\mathbf{S})$ and linearly many in $|\mathbf{S}|$ and $m_{\mathbf{S}}$.

The automaton $\mathcal{R}_{2,\mathbf{S}}$. Recall that a node v in t is well-colored iff it is either black, or it has at least two successor nodes which are both well-colored. Having this definition in place, devising $\mathcal{R}_{2,\mathbf{S}}$ becomes quite easy. In $\mathcal{G}(\mathcal{R}_{2,\mathbf{S}}, t)$, Adam's tasks is to prove that some node v is not well-colored. Adam's strategy to do so is as follows. Adam navigates to a node v that is not well-colored and that has a maximum distance from the root. Since v is not well-colored, v is white and it has less than two successors that are well-colored. Moreover, since v has maximum distance from the root, v must have either no successors or it has a single successor that is black. (Two black successors would turn v into a well-colored node, while one black and a white successor – both not well-colored – would turn the white successor into a node which is not well-colored, but has larger distance to the root.) Therefore, all Adam has to do is to challenge Eve to show the existence of that second successor. Adam will win if Eve cannot point to such a second successor.

Notice that the size of the state set of this automaton is independent from $\mathbf{S}$. In fact, $\mathcal{R}_{2,\mathbf{S}}$ has constantly many states. □

Remark 5.21. Let us remark here that the reason why we resort to $m_{\mathbf{S}}$-2APTA in this section (rather than $\updownarrow$-2APTA) lies in the construction of $\mathcal{R}_{\mathbf{S}}$, since Adam needs to be in control of the direction in which he moves.

The crucial task is to check condition (ii)(c) of Proposition 5.13, which states the key minimality criterion. Unfortunately, this involves an extra exponential blowup:

Lemma 5.22. *There is an $m_{\mathbf{S}}$-2APTA $\mathcal{M}_Q$ that accepts a consistent $\Gamma_{\mathbf{S}}$-labeled tree t iff $[\![t]\!] \setminus \{\alpha\} \not\models Q$ for all $\alpha \in [\![t]\!]$. The state set of $\mathcal{M}_Q$ is of doubly exponential size in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and of exponential size in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. Furthermore, $\mathcal{M}_Q$ can be constructed in triply exponential time in the size of Q, where the third exponent depends only on $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$.*

Proof. Firstly, we define an auxiliary alphabet $\Lambda_{\mathbf{S}}$ that is the cross product of $\Gamma_{\mathbf{S}}$ together with an alphabet $\Xi_{\mathbf{S}}$ which carries information on "tagged" facts. Formally, we set

$$\Xi_{\mathbf{S}} \coloneqq 2^F, \quad \text{where } F \coloneqq \{R^\sharp_{a_1,\dots,a_n} \mid R/n \in \mathbf{S}, a_1, \dots, a_n \in U_{\mathbf{S}}\},$$

and

$$\Lambda_{\mathbf{S}} \coloneqq \Gamma_{\mathbf{S}} \times \Xi_{\mathbf{S}}.$$

Hence, the alphabet $\Lambda_{\mathbf{S}}$ contains symbols of the form (ρ, ξ), where $\rho \in \Gamma_{\mathbf{S}}$ and ξ is a set of facts of the form $R^\sharp_{\bar{a}}$ which we will call *tagged* in the following. Intuitively, a tagged fact $R^\sharp_{\bar{a}}$ specifies that the minimization procedure – which is to be implemented in $\mathcal{M}_Q$ in the following – should aim to satisfy Q without the need of $R_{\bar{a}}$.

Given a $\Lambda_{\mathbf{S}}$-labeled tree t, recall that $\pi_{\Gamma_{\mathbf{S}}}(t)$ (respectively, $\pi_{\Xi_{\mathbf{S}}}(t)$) denotes the $\Gamma_{\mathbf{S}}$-projection (respectively, $\Xi_{\mathbf{S}}$-projection) of t. We say that t is *consistent* if

(i) $\pi_{\Gamma_{\mathbf{S}}}(t)$ is consistent.

(ii) There exists at least one node $v \in \mathsf{dom}(t)$ such that $\pi_{\Xi_{\mathbf{S}}}(t) \neq \varnothing$.

(iii) For all $v \in \mathsf{dom}(t)$, if $R^\sharp_{\bar{a}} \in \pi_{\Xi_{\mathbf{S}}}(t)$, then also $R_{\bar{a}} \in \pi_{\Gamma_{\mathbf{S}}}(t)$.

If t is a consistent $\Lambda_{\mathbf{S}}$-labeled tree, we define

$$[\![t]\!] \coloneqq [\![\pi_{\Gamma_{\mathbf{S}}}(t)]\!] \setminus \{R([v]_{a_1}, \dots, [v]_{a_n}) \mid v \in \mathsf{dom}(t), (\rho, \xi) \in t(v), R^\sharp_{a_1,\dots,a_n} \in \xi\}.$$

That is, $[\![t]\!]$ removes those facts from $[\![\pi_{\Gamma_{\mathbf{S}}}(t)]\!]$ which are tagged.

Lemma 5.23. *There is an $m_{\mathbf{S}}$-2APTA $\mathcal{N}_Q$ that accepts a $\Lambda_{\mathbf{S}}$-labeled tree t if and only if*

(i) *t is consistent, and*

(ii) *$[\![t]\!] \models Q$.*

The number of states of $\mathcal{N}_Q$ is exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and linear in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. Moreover, $\mathcal{N}_Q$ can be constructed in doubly exponential time in the size of Q, where the second exponent only depends on $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$.

Proof sketch. The $m_{\mathbf{S}}$-2APTA $\mathcal{N}_Q$ can be constructed as the intersection of two 2APTA, where one checks consistency and the other ensures that $[\![t]\!] \models Q$. The former can be constructed in a similar fashion as the one from Lemma 5.18 by additionally checking that (i) the input tree contains at least one tagged fact, and (ii) if a tagged fact occurs in a label, then also its non-tagged version does. Doing these checks is easily achieved by a single top-down pass. The latter can be constructed in a similar spirit as $\mathcal{A}_Q$ from Lemma 5.19 so that $\mathcal{N}_Q$ and the time needed to construct it respect the same bounds. □

Having $\mathcal{N}_Q$ from Lemma 5.23 in place, we are now going to construct $\mathcal{M}_Q$. The $m_{\mathbf{S}}$-2APTA $\mathcal{M}_Q$ will accept a consistent $\Gamma_{\mathbf{S}}$-labeled input tree t if and only if there is no $\Lambda_{\mathbf{S}}$-labeled t' such that

(i) $\pi_{\Gamma_{\mathbf{S}}}(t') = t$,

(ii) t' is consistent, and

(iii) $[\![t']\!] \models Q$.

Equivalently, $\mathcal{M}_Q$ will accept a consistent $\Gamma_{\mathbf{S}}$-labeled t iff there is no non-empty set of facts $A \subseteq [\![t]\!]$ such that $[\![t]\!] \setminus A \models Q$.

Constructing $\mathcal{M}_Q$ is easy using the machinery from Section 2.4. According to Proposition 2.32, from $\mathcal{N}_Q$ we can construct an $m_{\mathbf{S}}$-2APTA $\mathcal{N}_{Q,\Gamma}$ such that $\mathcal{L}(\mathcal{N}_{Q,\Gamma}) = \pi_\Gamma(\mathcal{N}_Q)$. The number of states of $\mathcal{N}_{Q,\Gamma}$ is exponential in the number of states of $\mathcal{N}_Q$, and the time needed to construct it is exponential in the size of $\mathcal{N}_Q$. Now $\mathcal{M}_Q$ is just defined as the complement of $\mathcal{N}_{Q,\Gamma}$. Building the complement is feasible in polynomial time (cf. Proposition 2.31), whence the claimed upper bounds on the size of the state set of $\mathcal{M}_Q$ and the time needed to construct it follow from Lemma 5.22 and Proposition 2.31. □

We are now ready to prove Lemma 5.11:

Proof of Lemma 5.11. We can obtain $\mathcal{B}_Q$ by intersecting the respective $m_{\mathbf{S}}$-2APTA from Lemmas 5.18 to 5.20 and 5.22. It is clear $\mathcal{B}_Q$ has doubly exponentially many states in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and, moreover, $\mathcal{B}_Q$ can be constructed in triply exponential time as claimed in the statement of Lemma 5.11. Thus, $\mathcal{B}_Q$ accepts a $\Gamma_{\mathbf{S}}$-labeled tree t if and only if

- t is consistent,
- t is well-colored and simple,
- $[\![t]\!] \models Q$, and
- $[\![t]\!] \setminus \{\alpha\} \not\models Q$ for all $\alpha \in [\![t]\!]$.

For a proof of Lemma 5.11, it thus remains to be shown that the language of $\mathcal{B}_Q$ is

infinite iff Q is not first-order rewritable.

Suppose first that $\mathcal{L}(\mathcal{B}_Q)$ is infinite. Since the branching degree of the input trees is bounded (recall that we run $\mathcal{B}_Q$ on $m_{\mathbf{S}}$-ary trees), the $m_{\mathbf{S}}$-2APTA $\mathcal{B}_Q$ accepts trees of arbitrary height. Hence, there are infinitely many trees $t_0, t_1, \ldots, t_k, \ldots$ and natural numbers $h_0, h_1, \ldots, h_k, \ldots$ such that, for $i \geq 0$,

- t_i has height h_i,
- $[\![t_i]\!] \models Q$,
- t_i is well-colored and simple, and
- $[\![t_i]\!] \models Q$, while $[\![t_i]\!] \setminus \{\alpha\} \not\models Q$ for all $\alpha \in [\![t_i]\!]$.

Moreover, we can assume that $i \neq j$ implies $h_i \neq h_j$ (otherwise we simply drop t_j). For $i \geq 0$, consider the standard tree decomposition δ_{t_i} and the standard adornment η_{t_i} of t_i. It is clear that δ_{t_i} is η_{t_i}-well-colored and η_{t_i}-simple as well. Moreover, for $i \neq j$, the triples $([\![t_i]\!], \delta_{t_i}, \eta_{t_i})$ must be non-isomorphic, since the height of t_i and t_j differ, i.e., $h_i \neq h_j$. We thus obtain by Proposition 5.13 that Q cannot be first-order rewritable.

Suppose now that Q is not first-order rewritable. By Proposition 5.13 there is an infinite class $\mathcal{S}$ of pairwise non-isomorphic triples $(\mathfrak{D}, \delta, \eta)$ – where $\mathfrak{D}$ is an $\mathbf{S}$-database, δ a tree decomposition of $\mathfrak{D}$ of width at most $\mathsf{wd}^*(\mathbf{S})$, and η an adornment of $(\mathfrak{D}, \delta)$ – such that

- δ is η-well-colored and η-simple,
- $\mathfrak{D} \models Q$, and
- for every $\alpha \in \mathfrak{D}$ it holds that $\mathfrak{D} \setminus \{\alpha\} \not\models Q$.

Recall that we can encode every such triple $\gamma = (\mathfrak{D}, \delta, \eta)$ as a $\Gamma_{\mathbf{S}}$-labeled tree t_γ. Considering the encoding, for two non-isomorphic triples $\gamma, \gamma' \in \mathcal{S}$ we must have $t_\gamma \neq t_{\gamma'}$. By construction we then have $\{t_\gamma \mid \gamma \in \mathcal{S}\} \subseteq \mathcal{L}(\mathcal{B}_Q)$. Hence, $\mathcal{L}(\mathcal{B}_Q)$ must be infinite since $\mathcal{S}$ is. This completes the proof of Lemma 5.11. □

5.4 Cost Automata Approach

We proceed to study $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ using the more sophisticated model of cost automata. This allows us to improve the complexity of $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ as stated in Theorem 5.4. That is, we show that $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ is in 2ExpTime, and in ExpTime for schemas of bounded width.

As in the previous approach, we develop a semantic characterization that relies on a minimality criterion for trees accepted by cost automata. The extra features provided by cost automata allow us to deal with such a minimality criterion in a more efficient way than standard 2APTA. While our application of cost automata is transparent, the complexity analysis relies on an intricate result on the boundedness problem for a certain class of cost automata from [31, 59]. Before we proceed further, let us first introduce the cost automata model.

5.4.1 Cost Automata

Cost automata [31, 58, 60] extend traditional automata by providing counters that can be manipulated at each transition. Instead of assigning a Boolean value to each input structure (indicating whether it is accepted or not), these automata assign a value from $\mathbb{N}_\infty := \mathbb{N} \cup \{\infty\}$ to each input – thus, cost automata define *cost functions*.

Here, we focus on cost automata that work on finite trees of unbounded degree, and allow for two-way movements. In fact, the automata we need are those that extend 2APTA over finite trees with a *single* counter. The operation of such an automaton $\mathcal{A}$ on each input t will be viewed as a two-player *cost game* $\mathcal{G}(\mathcal{A}, t)$ between the players Eve and Adam. Recall that the acceptance of an input tree for a conventional 2APTA can be formalized via a two-player game as well. However, instead of sequences of priority values, plays in the cost game between Eve and Adam will be assigned costs, and the cost automaton specifies via an *objective* whether Eve's goal is to minimize or maximize that cost.

We remark that in this section we are going to work on amorphous labeled trees, i.e., that have an arbitrary branching degree. This is not necessary for a technical reason, but it simplifies the presentation of our constructions.

Objectives An *objective* is a triple $\mathsf{Obj} = (\mathsf{Act}, f, \mathsf{goal})$, where Act is a finite set of *actions*, f an *objective function*, which assigns values from $\mathbb{N}_\infty$ to sequences of actions, and $\mathsf{goal} \in \{\min, \max\}$. We shall consider a run of a (two-way) alternating cost automaton as a two-player game with players Eve and Adam, where goal specifies whether Eve's aim is to minimize or maximize the objective function.

An example for an objective can be given by the well-known parity acceptance condition which we also used for plain 2APTA. This condition can be recast into a *parity objective* $\mathsf{parity} = (P, \mathsf{cost}_{\mathsf{parity}}, \mathsf{goal})$, where P is a finite set of *priorities* and $\mathsf{cost}_{\mathsf{parity}}$ is specified as follows: if $\mathsf{goal} = \min$ ($\mathsf{goal} = \max$, respectively) then $\mathsf{cost}_{\mathsf{parity}}$ maps a sequence of priorities to 0 (∞, respectively), if the least priority that occurs infinitely often is even, and to ∞ (0, respectively) otherwise.

The cost automata model In the following, given a set X, we denote by $\mathbb{B}^+(X)$ the set of all positive Boolean formulas without the use of true and false. Let Γ be an alphabet. A *two-way alternating cost automaton* $\mathcal{A}$ that *runs on* Γ-labeled trees is a tuple $(S, \Gamma, \mathsf{Dir}, s_0, \mathsf{Obj}, \delta)$, where

- S is a finite set of *states*.
- Obj is an objective.
- s_0 is the *initial state*.
- Dir, as in the case of 2APTA, describes the possible directions; in our case, we will always have $\mathsf{Dir} = \{0, \updownarrow\}$.
- $\delta \colon S \times \Gamma \to \mathbb{B}^+(\mathsf{tran}(\mathcal{A}))$ the *transition function*, where

$$\mathsf{tran}(\mathcal{A}) := \{\langle d \rangle(s, c), [d](s, c) \mid s \in S, c \in \mathsf{Act}, d \in \mathsf{Dir}\}.$$

To emphasize the objective that is used, we often call an automaton in the form of $\mathcal{A}$ an Obj-*automaton*, and we call the class of all Γ-labeled trees its *input trees*. Notice that each transition also carries information on the action $c \in \mathsf{Act}$ that is to be performed when transitioning to a new state. We will present the concrete actions available to our automata model below. We remark also that we excluded the formulas true and false from the possible transitions only for the sake of simpler definitions.

Cost game semantics Fix an Obj-automaton $\mathcal{A} = (S, \Gamma, \mathsf{Dir}, s_0, \mathsf{Obj}, \delta)$. As in the case of 2APTA, we assign *owners* to each formula from $\mathbb{B}^+(\mathsf{tran}(\mathcal{A}))$ in the expected manner. That is, conjunctions are owned by Adam, disjunctions are owned by Eve, atomic formulas of the form $[d](s, c)$ are owned by Adam, while those of the form $\langle d \rangle(s, c)$ are owned by Eve.

Let t be a Γ-labeled tree. We define a two-player *cost game* $\mathcal{G}(\mathcal{A}, t)$ *for* $\mathcal{A}$ *and* t. The arena of the game is $\mathbb{B}^+(\mathsf{tran}(\mathcal{A})) \times \mathsf{dom}(t)$, and the notion of possible choices for a position (χ, v) in the game is defined as in the case of 2APTA, that is:

- If $\chi = \chi_1 \wedge \chi_2$ or $\chi = \chi_1 \vee \chi_2$, then the possible choices are $\{(\chi_1, v), (\chi_2, v)\}$.
- If $\chi = [d](s, a)$ or $\chi = \langle d \rangle(s, a)$, then the possible choices are given by
$$\{(\delta(s, t(w)), w) \mid w \in d_t(v)\}.$$

Let $\chi_0 \coloneqq \delta(s_0, t(\varepsilon))$. The initial position of the game is (χ_0, ε), and, from any position (χ, v), the game proceeds as follows:

- The player who owns χ selects a (χ', w) from the possible choices of (χ, v), provided there is one, and if so then
- the game continues from position (χ', w).

The transition from (χ, v) to (χ', w) is a *move*. If χ is of the form $\langle d \rangle(s, c)$ or $[d](s, c)$, then we say that the *output* of (χ, v) is c. Otherwise, we simply say that the move has *no* output. A *play* in $\mathcal{G}(\mathcal{A}, t)$ is a (finite or infinite) sequence $(\chi_0, \varepsilon), (\chi_1, v_1), (\chi_2, v_2), \ldots$ of positions arising from successive moves, and a *strategy* for one of the players is a function that, given the history of the play, returns the next choice for that player. Again, fixing a strategy for both players uniquely determines a play in $\mathcal{G}(\mathcal{A}, t)$. A play π is *consistent* with a strategy ξ, if there is a strategy ξ' for the other player such that ξ and ξ' yield π. The *output* of a play $\pi = (\chi_0, v_0), (\chi_1, v_1), \ldots$ is the sequence of actions $c_{i_0}, c_{i_1}, \ldots$ such that (i) $0 \leq i_0 < i_1 < \cdots$, (ii) c_{i_j} is the output of (χ_{i_j}, v_{i_j}), for all $j \geq 0$, and (iii) if $i_j < \ell < i_{j+1}$ or $0 \leq \ell < i_0$, then (χ_ℓ, v_ℓ) has no output.

Suppose $\mathsf{Obj} = (\mathsf{Act}, f, \mathsf{goal})$. The *cost* of a play π with respect to Obj is the value of f on the output of that play. If $\mathsf{goal} = \min$ ($\mathsf{goal} = \max$, respectively) then an n-*winning strategy* for Eve is a strategy such that the cost of any play consistent with that strategy is at most n (at least n, respectively) with respect to Obj. We define

$$[\![\mathcal{A}]\!] \colon t \longmapsto \mathrm{op}\{n \mid \text{Eve has an } n\text{-winning strategy in } \mathcal{G}(\mathcal{A}, t)\},$$

where $\mathrm{op} = \inf$ (respectively, $\mathrm{op} = \sup$) if $\mathsf{goal} = \min$ (respectively, $\mathsf{goal} = \max$). We say that $[\![\mathcal{A}]\!]$ is the *cost function defined by* $\mathcal{A}$.

Counter actions As in [31], we are interested in objectives that are based on *counters*. We use the elementary actions *increment & check* $\mathtt{ic}$, *reset* $\mathtt{r}$, and *no change* ε. Let γ be a counter. Its initial value is 0 and afterwards it can take values from $\mathbb{N}$ according to a sequence $\bar{u}$ of actions from $\{\mathtt{ic}, \mathtt{r}, \varepsilon\}$. The meanings of $\mathtt{r}$ and ε are clear. The operation $\mathtt{ic}$ increments the counter value (the *increment*) and, at the same time, indicates that we are interested in the current value of the counter (the *check*). Let $C_\gamma(\bar{u})$ denote the set of values at the moment(s) in the sequence $\bar{u}$ when γ is checked, i.e., when the operation $\mathtt{ic}$ occurs. For example, $C_\gamma(\mathtt{icicricricicic}) = \{2, 1, 3\}$ and $C_\gamma(\mathtt{icicricic}) = \{2\}$.

Distance and parity automata The *distance objective* is

$$\mathsf{dist} \coloneqq (\{\mathtt{ic}, \mathtt{r}, \varepsilon\}, \mathsf{cost}_{\mathsf{dist}}, \min),$$

and we assume that dist only has a single counter, say γ. The function $\mathsf{cost}_{\mathsf{dist}}$ maps a sequence $\bar{u}$ of counter actions over $\{\mathtt{ic}, \mathtt{r}, \varepsilon\}$ to $\sup C_\gamma(\bar{u})$. We will be interested in an objective that combines dist with the parity acceptance condition parity. Formally, given two objectives $O_1 = (\mathsf{Act}_1, f_1, \min), O_2 = (\mathsf{Act}_2, f_2, \min)$, we denote by $O_1 \wedge O_2$ the objective $(\mathsf{Act}_1 \times \mathsf{Act}_2, \max\{f_1, f_2\}, \min)$.

We concentrate on $\mathsf{dist} \wedge \mathsf{parity}$-*automata* [31], which are cost automata with a single counter, and where Eve's objective is to minimize the cost while satisfying the parity condition.

Boundedness A key property of cost functions defined by cost automata is *boundedness*. We say that $\mathcal{A}$ is *bounded* over a set $\mathcal{C}$ of input trees, if there exists an $n \in \mathbb{N}$ such that $[\![\mathcal{A}]\!](t) \leq n$ for every $t \in \mathcal{C}$. Moreover, we say that $\mathcal{A}$ is *bounded* if it is bounded over the set of all input trees. The following result establishes that, for $\mathsf{dist} \wedge \mathsf{parity}$-automata, the boundedness problem is decidable in singly exponential time.

Theorem 5.24. *There is a polynomial f such that, for every $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathcal{A}$ using priorities $\{0, 1\}$ for the parity acceptance condition, boundedness for $\mathcal{A}$ is decidable in time $\|\mathcal{A}\|^{f(m)}$, where m is the number of states of $\mathcal{A}$.*

Remark 5.25. This result is formally stated in [31]. The authors of [31] rely on an unpublished result from [57]. This result has subsequently been published in [59].

Our goal is to reduce $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ to the boundedness problem for $\mathsf{dist} \wedge \mathsf{parity}$-automata. To this end, we first devise a semantic characterization of the first-order rewritable OMQs from $(\mathsf{G}, \mathsf{BAQ})$ that is suitable for the use of cost automata. We then devise a cost automaton that exploits this semantic characterization.

5.4.2 A Refined Semantic Characterization

Let $Q = (\mathbf{S}, \mathcal{O}, G)$ be an OMQ from $(\mathsf{G}, \mathsf{BAQ})$, and $\mathfrak{D}$ an $\mathbf{S}$-database. Recall the notion of a derivation tree for G with respect to $\mathfrak{D}$ and Q from Subsection 4.3.1. Given such a derivation tree $\mathcal{T}$, the *height* of $\mathcal{T}$ is the maximum length of a path that leads from the root node to a leaf node plus one, i.e., the maximum number of nodes of a branch.

Notice that the height of a derivation tree is defined such that every derivation tree has a height of at least one. We say that $\mathcal{T}$ is *k-ary* if its branching degree is at most k.

Lemma 5.26. *For every $\mathbf{S}$-database $\mathfrak{D}$, the following are equivalent:*

(i) $\mathfrak{D} \models Q$.

(ii) *There exists a k_Q-ary derivation tree for G w.r.t. $\mathfrak{D}$ and Q, where*

$$k_Q \coloneqq |\mathbf{S} \cup \mathsf{sig}(\mathcal{O})| \cdot \mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))^{\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))}.$$

Proof sketch. The direction from (ii) to (i) is immediate by Lemma 4.44. The other direction result essentially already follows from Lemma 4.44 by further analyzing the derivation trees constructed in the proof of Lemma 4.44. More precisely, one observes that the maximum number of non-equivalent bodies of guarded Datalog rules that one can construct using symbols from $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ is bounded by k_Q. This allows one to restrict to such rules, whence the branching degree of the constructed derivation tree is bounded by k_Q. □

We define

$$\mathsf{cost}(\mathfrak{D}, Q) \coloneqq \min\{n \mid \text{there is a } k_Q\text{-ary derivation tree for } G \text{ w.r.t. } \mathfrak{D} \text{ and } Q \text{ of height } n\},$$

while the *cost of Q* is defined as

$$\mathsf{cost}(Q) \coloneqq \sup\{\mathsf{cost}(\mathfrak{D}, Q) \mid \mathfrak{D} \models Q,\ \mathfrak{D} \text{ is an } \mathbf{S}\text{-database with } \mathsf{tw}(\mathfrak{D}) \leq \mathsf{wd}^*(\mathbf{S})\}.$$

In other words, the cost of Q is the least upper bound of $\mathsf{cost}(\mathfrak{D}, Q)$ over all $\mathbf{S}$-databases $\mathfrak{D}$ of tree-width at most $\mathsf{wd}^*(\mathbf{S})$ such that $\mathfrak{D} \models Q$. If there is no such a database, then the cost of Q is zero since $\sup \varnothing \coloneqq 0$. Actually, $\mathsf{cost}(Q) = 0$ indicates that Q is unsatisfiable, since $\mathfrak{D} \models Q$ implies that $\mathsf{cost}(\mathfrak{D}, Q) \geq 1$.

If Q is unsatisfiable, then Q is trivially first-order rewritable. More generally, it turns out that Q is first-order rewritable iff its cost is bounded. This is the content of the following statement:

Proposition 5.27. *For every OMQ $Q \in (\mathsf{G}, \mathsf{BAQ})$, the following are equivalent:*

(i) *Q is first-order rewritable.*

(ii) $\mathsf{cost}(Q) < \infty$.

Proof. Let $Q = (\mathbf{S}, \mathcal{O}, G)$. Throughout the proof, we let $w \coloneqq \mathsf{wd}^*(\mathbf{S})$. Suppose first that Q is first-order rewritable. By Proposition 5.7, this means that there is a $k \geq 0$ such that, for every $\mathbf{S}$-database $\mathfrak{D}$ of tree-width at most w, if $\mathfrak{D} \models Q$, then there is a $\mathfrak{D}' \subseteq \mathfrak{D}$ with at most k facts such that $\mathfrak{D}' \models Q$. We show that $\mathsf{cost}(Q) < \infty$. Let $\mathcal{S}$ be the class of all $\mathbf{S}$-databases $\mathfrak{D}$ with at most k facts such that $\mathfrak{D} \models Q$, and consider $\mathcal{S}$ to be factorized modulo weak isomorphisms. Clearly, $\mathcal{S}$ must be finite. For each $\mathfrak{D} \in \mathcal{S}$, let $\mathcal{T}_\mathfrak{D}$ be a k_Q-ary derivation tree for G w.r.t. $\mathfrak{D}$ and Q. Let $n < \infty$ be the maximum

height among all the trees $\mathcal{T}_{\mathfrak{D}}$. We claim that $\mathsf{cost}(Q) \leq n$. Indeed, suppose that $\mathfrak{D}$ is an $\mathbf{S}$-database of tree-width at most w such that $\mathfrak{D} \models Q$. By assumption, we can find a $\mathfrak{D}' \subseteq \mathfrak{D}$ of at most k atoms such that $\mathfrak{D}' \models Q$. Hence, (some weakly isomorphic copy of) $\mathfrak{D}'$ is contained in $\mathcal{S}$. Consider a k_Q-ary derivation tree $\mathcal{T}$ for G w.r.t. $\mathfrak{D}$ and Q of minimal height. In case the height of $\mathcal{T}$ is greater that n, we know that $\mathcal{T}_{\mathfrak{D}'}$ is also a derivation tree for G w.r.t. $\mathfrak{D}$ and Q due to the fact that $\mathfrak{D}' \subseteq \mathfrak{D}$. In this case, $\mathcal{T}$ is not the minimal one, which is absurd by assumption. Therefore, the height of $\mathcal{T}$ is at most n, and so $\mathsf{cost}(Q) \leq n$ as required.

Suppose now that $\mathsf{cost}(Q) = n$ for some $n \in \mathbb{N}$. Let $\mathfrak{D}$ be an $\mathbf{S}$-database of tree-width at most $\mathsf{wd}^*(\mathbf{S})$. Given a k_Q-ary derivation tree $\mathcal{T}$ for G w.r.t. $\mathfrak{D}$ and Q, we define a BCQ $q_{\mathcal{T}}$ as follows. Let

$$\varphi_{\mathcal{T}} \coloneqq \bigwedge \{\alpha \mid \alpha \text{ is an } \mathbf{S}\text{-fact and the label of a leaf node of } \mathcal{T}\}.$$

We define $q_{\mathcal{T}}$ to be the BCQ that results from $\varphi_{\mathcal{T}}$ by renaming all constants to variables and closing the resulting formula off under existential quantifiers. Notice that if $\varphi_{\mathcal{T}}$ is empty, then $q_{\mathcal{T}} \equiv \varphi_{\mathcal{T}} \equiv \top$. Now let

$$\mathcal{S} \coloneqq \{q_{\mathcal{T}} \mid \mathcal{T} \text{ is a } k_Q\text{-ary derivation tree for } G \text{ w.r.t. } \mathfrak{D} \text{ and } Q \\ \text{of minimal height, where } \mathsf{tw}(\mathfrak{D}) \leq \mathsf{wd}^*(\mathbf{S})\},$$

and let $\mathcal{S}'$ be the set $\mathcal{S}$ factorized modulo homomorphic equivalence. We claim that

$$q \coloneqq \bigvee \mathcal{S}'$$

is a UBCQ equivalent to Q, and hence a first-order rewriting of Q. Observe first that q is clearly a finite disjunction, since the number of atoms of each query in $\mathcal{S}$ is clearly uniformly bounded due to the bound on the height and the branching degree of the derivation trees. Hence, up to homomorphic equivalence, there are only finitely many BCQs in $\mathcal{S}$.

Suppose first that $\mathfrak{D} \models Q$ for some $\mathbf{S}$-database $\mathfrak{D}$. By Lemma 5.26 there exists a k_Q-ary derivation tree $\mathcal{T}$ for G w.r.t. $\mathfrak{D}$ and Q. Hence, (some query homomorphically equivalent to) $q_{\mathcal{T}}$ is present in $\mathcal{S}'$, whence $\mathfrak{D} \models q$ follows immediately by construction.

Suppose now to the contrary that $\mathfrak{D} \models q$ for some $\mathbf{S}$-database $\mathfrak{D}$, i.e., $\mathfrak{D} \models q_{\mathcal{T}}$ for some $q_{\mathcal{T}}$ present in $\mathcal{S}'$. A straightforward induction on the height of $\mathcal{T}$ shows that $\mathfrak{D} \models Q$ holds as well. □

5.4.3 Devising automata

Consider an OMQ $Q = (\mathbf{S}, \mathcal{O}, G)$ from $(\mathsf{G}, \mathsf{BAQ})$. Our goal is to devise a $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathcal{B}_Q$ such that $\mathcal{B}_Q$ is bounded iff $\mathsf{cost}(Q)$ is finite. Therefore, by Proposition 5.27, to check whether Q is first-order rewritable, we simply need to check if $[\![\mathcal{B}_Q]\!]$ is bounded, which, by Theorem 5.24, can be done in exponential time in the number of states of $\mathcal{B}_Q$.

The input trees to our automata will be over the same alphabet $\Gamma_{\mathbf{S}}$ that is used to encode tree-like $\mathbf{S}$-databases in Sections 4.3 and 5.3. Recall that a $\mathsf{dist} \wedge \mathsf{parity}$-automaton

$\mathcal{A}$ is bounded over a certain class $\mathcal{C}$ of input trees, if there is an $n \in \mathbb{N}$ such that $[\![\mathcal{A}]\!](t) \leq n$ for every tree $t \in \mathcal{C}$. The following lemma is the main ingredient toward a proof of Theorem 5.4:

Lemma 5.28. *There is a* $\mathsf{dist} \wedge \mathsf{parity}$*-automaton* $\mathcal{H}_Q$ *using priorities* $\{0,1\}$ *such that*

$$\mathcal{H}_Q \text{ is bounded over consistent } \Gamma_{\mathbf{S}}\text{-labeled trees} \iff \mathsf{cost}(Q) < \infty.$$

The number of states of $\mathcal{H}_Q$ *is exponential in* $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ *and linear in* $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$*. Moreover,* $\mathcal{H}_Q$ *can be constructed in doubly exponential time such that the second exponent of the runtime only depends on* $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$*.*

Proof. Let $Q = (\mathbf{S}, \mathcal{O}, G)$ be an OMQ from $(\mathsf{G}, \mathsf{BAQ})$. Consider a consistent $\Gamma_{\mathbf{S}}$-labeled tree t. We are going to devise a $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathcal{H}_Q$ running on $\Gamma_{\mathbf{S}}$-labeled trees such that Eve has an n-winning strategy in $\mathcal{G}(\mathcal{H}_Q, t)$ if and only if there is a derivation tree $\mathcal{T}$ for G w.r.t. $[\![t]\!]$ and Q of height at most n.

Let $\mathcal{H}_Q = (S, \Gamma_{\mathbf{S}}, \{0, \updownarrow\}, s_0, \mathsf{dist} \wedge \mathsf{parity}, \delta)$. For the parity condition, we shall only use the priorities $\{0,1\}$. The remaining components of $\mathcal{H}_Q$ are specified in the following.

- *The state set* S: Let $U_{\mathbf{S}}$ be the finite set of constants that is used for arguments in $\Gamma_{\mathbf{S}}$. The state set S consists of all atomic formulas $R(a_1, \ldots, a_k)$, where $R/k \in \mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ and $a_1, \ldots, a_k \in U_{\mathbf{S}}$. We set the initial state s_0 to be equal to G. For technical reasons, we include an additional sink state, denoted sink.
- *The transition function* δ: We define δ as follows. Consider a symbol $\rho \in \Gamma_{\mathbf{S}}$. Firstly, we set

  $$\delta(\mathsf{sink}, \rho) \coloneqq \langle 0 \rangle (\mathsf{sink}, \varepsilon, 0).$$

  Secondly, let $R(a_1, \ldots, a_k)$ be a state different from sink. We set $\bar{a} \coloneqq a_1, \ldots, a_k$ and distinguish three cases:

  (C1) If $\{a_1, \ldots, a_k\} \not\subseteq \mathsf{names}(\rho)$, then

  $$\delta(R(\bar{a}), \rho) \coloneqq \langle 0 \rangle (R(\bar{a}), \mathtt{ic}, 1).$$

  In this case, Eve will lose the game as she loops in this state while incrementing the counter and producing an infinite run whose least priority that occurs infinitely often (i.e., the priority 1) is odd.

  (C2) Otherwise, if $\mathcal{O} \models R(\bar{a})$, then

  $$\delta(R(\bar{a}), \rho) \coloneqq \langle 0 \rangle (\mathsf{sink}, \mathtt{ic}, 1).$$

  In this case, the atomic $R(\bar{a})$ is already entailed by $\mathcal{O}$ alone, and Eve wins unconditionally by switching again to the sink state. Notice that, since we assume constant-free rules, this is only possible when $[\bar{a}] = \varnothing$.

  (C3) Otherwise, if $R(\bar{a}) \in \rho$ then

  $$\delta(R(\bar{a}), \rho) \coloneqq \langle 0 \rangle (\mathsf{sink}, \mathtt{ic}, 1).$$

Eve will win the game as she first changes to the sink state (incrementing the counter once), and then she loops in the sink state while not incrementing the counter. In the sink state, she thus produces an infinite play whose least priority that occurs infinitely often (the priority 0) is even.

(C4) Otherwise, let

$$\tau_1 \coloneqq \alpha_{1,1} \wedge \cdots \wedge \alpha_{1,m_1},\ \ldots,\ \tau_l \coloneqq \alpha_{l,1} \wedge \cdots \wedge \alpha_{l,m_l}$$

be an enumeration of all non-empty guarded conjunctions of atomic facts from S for which it holds that $(\{\alpha_{i,1}, \ldots, \alpha_{i,m_i}\}, \mathcal{O}) \models R(\bar{a})$, for all $i = 1, \ldots, l$. Notice that $l \geq 1$ since $(\{R(\bar{a})\}, \mathcal{O}) \models R(\bar{a})$. We let

$$\delta(R(\bar{a}), \rho) \coloneqq \langle\updownarrow\rangle(R(\bar{a}), \varepsilon, 1) \vee \bigvee_{i=1}^{l} \bigwedge_{j=1}^{m_i} \langle 0\rangle(\alpha_{i,j}, \mathtt{ic}, 1).$$

Eve may choose between two possibilities here. Either she moves to some neighboring node in the tree while remaining in state $R(\bar{a})$, or she may decide to pick a guarded conjunction $\tau_i \coloneqq \alpha_{i,1} \wedge \cdots \wedge \alpha_{i,m_i}$. In the latter case, Adam challenges Eve's choice by changing the state to one of the $\alpha_{i,j}$ while incrementing the counter. Notice that this case corresponds to the unfolding of a (series of) rules and thus to the built-up of a derivation tree.

This completes the construction of $\mathcal{H}_Q$. Notice also that the construction is reminiscent of the one provided in the proof of Lemma 4.42. We briefly comment on the size of $\mathcal{H}_Q$ and the time required to construct the same.

It is clear that the number of states of $\mathcal{H}_Q$ is exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(\mathcal{O}))$ and linear in $|\mathbf{S} \cup \mathsf{sig}(\mathcal{O})|$. Moreover, the overall construction of $\mathcal{H}_Q$ takes doubly exponential time in the size of Q. An analysis of this is performed as in the proof of Lemma 4.42. The determining factor for this upper bound is again the construction of $\delta(\cdot, \cdot)$.

It is technically rather tedious, but not very interesting, to show that the behavior of $\mathcal{H}_Q$ is indeed correct, that is, if t is a consistent $\Gamma_{\mathbf{S}}$-labeled tree, then Eve has an n-winning strategy in $\mathcal{G}(\mathcal{H}_Q, t)$ iff there is a derivation tree $\mathcal{T}$ for G w.r.t. $[\![t]\!]$ and Q of height at most n. We leave this technical result as an exercise to the reader.

Hence, for any consistent $\Gamma_{\mathbf{S}}$-labeled tree t, we have that $[\![\mathcal{H}_Q]\!](t) = n$ if and only if n is the minimal n_0 such that there is a derivation tree for G w.r.t. $[\![t]\!]$ and Q of height n_0. Therefore, for all $n \in \mathbb{N}$, it is the case that $[\![\mathcal{H}_Q]\!](t) = n$ iff $\mathsf{cost}([\![t]\!], Q) = n$. We thus obtain that $\mathcal{H}_Q$ is bounded over consistent $\Gamma_{\mathbf{S}}$-labeled trees iff $\mathsf{cost}(Q) < \infty$. By Proposition 5.27, this means that $\mathcal{H}_Q$ is bounded over all consistent $\Gamma_{\mathbf{S}}$-labeled trees iff Q is first-order rewritable. This concludes the proof of Lemma 5.28. □

We are now in a position to prove Theorem 5.4:

Proof of Theorem 5.4. The desired $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathcal{B}_Q$ is defined as $\mathcal{C}'_{\mathbf{S}} \cap \mathcal{H}_Q$, where $\mathcal{C}'_{\mathbf{S}}$ is similar to the 2APTA $\mathcal{C}_{\mathbf{S}}$ of Lemma 4.41 that checks for consistency of $\Gamma_{\mathbf{S}}$-labeled trees. Notice that $\mathcal{C}'_{\mathbf{S}}$ is essentially a $\mathsf{dist} \wedge \mathsf{parity}$-automaton that assigns

zero (respectively, ∞) to input trees that are consistent (respectively, inconsistent). One can take build the intersection of two cost automata with the objectives O_1 and O_2 in polynomial time similarly as in the case of ordinary 2APTA (cf. Proposition 2.31) so that the resulting cost automaton has the objective $O_1 \wedge O_2$. It is easy to check that $\mathcal{C}'_{\mathbf{S}} \cap \mathcal{H}_Q$ is again a $\mathsf{dist} \wedge \mathsf{parity}$-automaton that only uses priorities $\{0, 1\}$, since $\mathcal{C}'_{\mathbf{S}}$ uses no counters. Since we can build the intersection $\mathcal{C}'_{\mathbf{S}} \cap \mathcal{H}_Q$ in polynomial time, Lemmas 5.28 and 4.41 imply that $\mathcal{B}_Q$ has exponentially many states, and that it can be constructed in doubly exponential time within the required time bounds stated in Theorem 5.4. Moreover, Lemma 5.28 also implies that $\mathcal{B}_Q$ is bounded iff $\mathsf{cost}(Q) < \infty$.

It remains to be shown that the boundedness of $[\![\mathcal{B}_Q]\!]$ can be checked in double exponential time. By Theorem 5.24, there is a polynomial f such that the latter task can be carried out in time $\|\mathcal{B}_Q\|^{f(m)}$, where m is the number of states of $\mathcal{B}_Q$, and the claim follows. For schemas of bounded width, one easily checks that a similar analysis that yields a singly exponential time upper bound. □

5.5 Frontier-Guarded OMQs

As announced in the introductory part of this chapter, we will now show how to employ Theorem 5.4 in order to arrive at the upper bounds stated in Theorem 5.3. To this end, we need to show the following:

(i) $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{UCQ})$ is in 2ExpTime, and

(ii) $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{AQ})$ is in ExpTime, assuming that we only use relation symbols of bounded arity.

As in Subsection 4.3.2, it turns out that we can focus on the Boolean variants of these problems:

Lemma 5.29. (i) $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{UCQ})$ *can be polynomially reduced to* $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{BAQ})$.

(ii) $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{AQ})$ *can be polynomially reduced to* $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$.

Proof. We only prove the first item, the second one is shown accordingly. The reduction is similar to the one presented in the proof of Lemma 4.45 and is based on an according one for description logics given in [34]. Let $Q = (\mathbf{S}, \mathcal{O}, q(x_1, \ldots, x_n))$ be an OMQ from $(\mathsf{FG}, \mathsf{UCQ})$, where

$$q(x_1, \ldots, x_n) = \bigvee_{i=1}^{k} q_i(x_1, \ldots, x_n),$$

and all the $q_i(x_1, \ldots, x_n)$ are CQs. Assume w.l.o.g. that the variables $x_1, \ldots, x_n$ are all pairwise distinct. Pick fresh unary relation symbols $A_1, \ldots, A_n$ that do not occur in Q, and, for $i = 1, \ldots, k$, let q'_i be the BCQ that results from q_i by (i) adding the atoms $A_1(x_1), \ldots, A_n(x_n)$ as conjuncts to the body atoms of q_i, and (ii) closing it off under existential quantifiers. Let $\mathbf{S}' := \mathbf{S} \cup \{A_1, \ldots, A_n\}$, and let $\mathcal{O}'$ consist of the rules from $\mathcal{O}$

plus the rules

$$q_i' \to G, \quad i = 1, \ldots, n,$$

where G is a 0-ary predicate not occurring in Q. Let $Q' \coloneqq (\mathbf{S}', \mathcal{O}', G)$. We claim that Q is first-order rewritable iff Q' is.

Indeed, if $\varphi_Q(x_1, \ldots, x_n)$ is a first-order rewriting of Q, then obviously

$$\varphi_{Q'} \coloneqq \exists x_1, \ldots, x_n \, (\varphi_Q(x_1, \ldots, x_n) \land A_1(x_1) \land \cdots \land A_n(x_n))$$

is a first-order rewriting of Q'.

Conversely, suppose $q_{Q'}$ is a UCQ-rewriting of Q'. By the construction of Q', it is easy to show that we can assume w.l.o.g. that each disjunct of $q_{Q'}$ has body atoms $A_1(v_1), \ldots, A_n(v_n)$, where $v_1, \ldots, v_n$ is a sequence of pairwise distinct variables. Moreover, we can assume that these are the only atoms over the schema $\mathbf{S}' \setminus \mathbf{S}$ that $q_{Q'}$ contains in its constituent BCQs. Let us rename the variables $v_1, \ldots, v_n$ to $x_1, \ldots, x_n$ in $q_{Q'}$.

Let q_Q be the UCQ resulting from $q_{Q'}$ by dropping all atoms from $q_{Q'}$ of the form $A_i(x_i)$ and their existential quantifiers $\exists x_i$. We claim that q_Q is a UCQ-rewriting of Q.

Suppose that $\mathfrak{D}$ is an $\mathbf{S}$-databases such that $\mathfrak{D} \models Q(a_1, \ldots, a_n)$. Let $\mathfrak{D}'$ be the $\mathbf{S}'$-database $\mathfrak{D} \cup \{A_1(a_1), \ldots, A_n(a_n)\}$. From $\mathfrak{D} \models Q(a_1, \ldots, a_n)$ we infer that $\mathfrak{D}' \models Q'$, whence $\mathfrak{D}' \models q_{Q'}$ follows. Hence, there is a disjunct p of $q_{Q'}$ that contains atoms of the form $A_i(v_i)$ so that $\mathfrak{D}' \models p$, whence it follows that $\mathfrak{D} \models q_Q(a_1, \ldots, a_n)$.

Suppose on the other hand that $\mathfrak{D}$ is an $\mathbf{S}$-database such that $\mathfrak{D} \models q_Q(a_1, \ldots, a_n)$. Let $\mathfrak{D}' \coloneqq \mathfrak{D} \cup \{A_1(a_1), \ldots, A_n(a_n)\}$. It follows that $\mathfrak{D}' \models q_{Q'}$, whence from $\mathfrak{D}' \models Q'$ we conclude that $\mathfrak{D} \models Q(a_1, \ldots, a_n)$. □

Now Theorem 5.4 and Lemma 5.29 immediately entail that the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{AQ})$ is in ExpTime for schemas of bounded width. Therefore, it remains to be shown that $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{BAQ})$ is in 2ExpTime.

Treeification As mentioned in the introductory part of this chapter, to show that $\mathsf{FO}^{\Leftarrow}(\mathsf{FG}, \mathsf{BAQ})$ is in 2ExpTime, we exploit the treeification techniques developed in Subsection 4.3.2.

Fix an OMQ $Q = (\mathbf{S}, \mathcal{O}, G)$ from $(\mathsf{FG}, \mathsf{BAQ})$, and let C be a fresh relation symbol of arity $\mathsf{wd}(\mathcal{O})$ – recall that $\mathsf{wd}(\mathcal{O})$ denotes the maximum number of variables occurring in a rule body of $\mathcal{O}$. We remind the reader that $\eta_C(Q)$ denotes the C-treeification of Q defined in Subsection 4.3.2. As in the case of containment, first-order rewritability is preserved when passing from an OMQ to its C-treeification. This is the content of the following lemma:

Lemma 5.30. *Q is first-order rewritable iff $\eta_C(Q)$ is.*

Before proving Lemma 5.30, we need one additional technical results first:

Lemma 5.31. *Let $\mathfrak{D}$ be an $\mathbf{S}$-database. If $\mathfrak{D} \models Q$, then there exists an $\mathbf{S}$-database $\mathfrak{B}$ of tree-width at most $\max\{0, \mathsf{wd}(\mathcal{O}) - 1\}$ such that*

(i) $\mathfrak{B} \models Q$, *and*

(ii) *there is a weak homomorphism from* $\mathfrak{B}$ *to* $\mathfrak{D}$.

Proof. Suppose that $\mathfrak{D} \models Q$, and consider the C-treeification $\eta_C(Q)$ of Q. Lemma 4.57 tells us that $\mathfrak{D} \models Q$ implies $\mathfrak{D}_C \models \eta_C(Q)$, where $\mathfrak{D}_C$ denotes the $(\mathbf{S} \cup \{C\})$-database obtained by exhaustively adding all possible $\{C\}$-facts to $\mathfrak{D}$ using constants from $\mathsf{adom}(\mathfrak{D})$. By Lemma 4.36, we know that there is an acyclic $(\mathbf{S} \cup \{C\})$-database $\mathfrak{B}'$ such that $\mathfrak{B}' \models \eta_C(Q)$, and a weak homomorphism from $\mathfrak{B}'$ to $\mathfrak{D}_C$. Let $\mathfrak{B}$ be the $\mathbf{S}$-database obtained by building the $\mathbf{S}$-retract of $\mathfrak{B}'$, i.e., $\mathfrak{B} \coloneqq \mathfrak{B} \restriction \mathbf{S}$. Obviously, there is a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$. Moreover, we can easily see that $\mathfrak{B}$ has tree-width at most $\max\{0, \mathsf{wd}(\mathcal{O}) - 1\}$, since $\mathfrak{B}'$ is acyclic and the arity of C is $\mathsf{wd}(\mathcal{O})$. This proves the claim. □

Proof of Lemma 5.30. Throughout the proof, let us write w for $\max\{0, \mathsf{wd}(\mathcal{O}) - 1\}$. We can assume that $w > 0$, as the case $w = 0$ is entirely trivial, since in this case, Q and $\eta_C(Q)$ are always first-order rewritable. Suppose first that Q is first-order rewritable, and let $q = \bigvee_{i=1}^{n} p_i$ be a UCQ-rewriting of Q. By the *tree-width* of a CQ p, we mean the tree-width of the structure corresponding to p.

Claim 5.32. *The UBCQ q is equivalent to a UBCQ whose disjuncts are all of tree-width at most w.*

Proof. Let q' be a UBCQ that contains a disjunct p' iff (i) p' has tree-width at most w, (ii) $p' \models p_i$ for some $i = 1, \ldots, n$, and (iii) p' is minimal with respect to these properties. Moreover, we demand two distinct disjuncts of q' are not homomorphically equivalent. Thus, q' is indeed a finite UBCQ. We show that q is equivalent to q'.

Suppose first that $\mathfrak{D} \models q$. Then also $\mathfrak{D} \models Q$, whence by Lemma 5.31 there is an $\mathbf{S}$-database $\mathfrak{B}$ of tree-width at most w such that $\mathfrak{B} \models Q$ and a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$. Hence, $\mathfrak{B} \models q$. Since $\mathfrak{B}$ has tree-width at most w, there is a $\mathfrak{B}' \subseteq \mathfrak{B}$ and a disjunct p of q' such that $\mathfrak{B}'$ and p are *weakly homomorphically equivalent*, i.e., there is a weak homomorphism from $\mathfrak{A}_p$ to $\mathfrak{B}'$ and vice versa. Hence $\mathfrak{B}' \models p$ and thus $\mathfrak{B} \models p$. Since there is a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$, it follows that $\mathfrak{D} \models p$ and thus $\mathfrak{D} \models q'$.

Suppose now that $\mathfrak{D} \models q'$, i.e., $\mathfrak{D} \models p$ for some disjunct p of q'. Since $p \models p_i$, we immediately obtain $\mathfrak{D} \models q$. □

By Claim 5.32 we may therefore assume w.l.o.g. that each p_i has tree-width at most w. For each $i = 1, \ldots, n$, let $\delta_i = (\mathcal{T}_i, (X_{i,v})_{v \in T_i})$ be a tree decomposition of width at most w of the structure $\mathfrak{A}_{p_i}$ corresponding to p_i. Let

$$q' \coloneqq \bigvee \{\Lambda_{p_i}^{\mathbf{S} \cup \{C\}} \mid i = 1, \ldots, n\}$$

be the disjunction of the $(\mathbf{S} \cup \{C\})$-treeifications of the queries $p_1, \ldots, p_n$. Obviously, q' is a finite UBCQ, and we claim that q' is a UCQ-rewriting of $\eta_C(Q)$.

Indeed, suppose $\mathfrak{D}$ is an $(\mathbf{S} \cup \{C\})$-database such that $\mathfrak{D} \models \eta_C(Q)$. According to Lemma 4.36, there is an acyclic $(\mathbf{S} \cup \{C\})$-database $\mathfrak{B}$ such that $\mathfrak{B} \models \eta_C(Q)$ and a weak

homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$. By Lemma 4.55 we have $\mathfrak{B} \restriction \mathbf{S} \models Q$ as well, whence $\mathfrak{B} \restriction \mathbf{S} \models p_i$ for some $i = 1, \dots, n$ and thus $\mathfrak{B} \models p_i$. Lemma 4.53 tells us that

$$\mathfrak{B} \models p_i \iff \mathfrak{B} \models \Lambda_{p_i}^{\mathbf{S} \cup \{C\}},$$

since $\mathfrak{B}$ is acyclic. Hence, also $\mathfrak{B} \models q'$, whence $\mathfrak{D} \models q'$ follows due to the fact that there is a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$.

Conversely, suppose that $\mathfrak{D} \models q'$, i.e., $\mathfrak{D} \models p$ for some p that is a disjunct of the $(\mathbf{S} \cup \{C\})$-treeification of some p_i. Consider the structure $\mathfrak{A}_p$ corresponding to p. Obviously, $\mathfrak{A}_p \models p$ and since $p \models p_i$ also $\mathfrak{A}_p \models p_i$, whence $\mathfrak{A}_p \models q$ and thus $\mathfrak{A}_p \restriction \mathbf{S} \models Q$ follows. By Lemma 4.55 we obtain $\mathfrak{A}_p \models \eta_C(Q)$ as well. Now $\mathfrak{D} \models p$ obviously entails that there is a weak homomorphism from $\mathfrak{A}_p$ to $\mathfrak{D}$. Hence, $\mathfrak{D} \models \eta_C(Q)$ since OMQs are closed under weak homomorphisms (Proposition 3.18).

Suppose now that $\eta_C(Q)$ is first-order rewritable and let $q = \bigvee_{i=1}^{n} p_i$ be a UCQ-rewriting of $\eta_C(Q)$. We show that Q is first-order rewritable as well. We can assume that q is actually a disjunction of acyclic BCQs, since, if not, we can replace q simply by the disjunction of all $(\mathbf{S} \cup \{C\})$-treeifications of the BCQs p_i. It is not hard to show that the resulting UBCQ is equivalent to the original q.

For $i = 1, \dots, n$, let p_i' be the BCQ that results from p_i by dropping all body atoms of the form $C(x_0, \dots, x_w)$. Moreover, let $q' \coloneqq \bigvee_{i=1}^{n} p_i'$. We claim that the UBCQ q' is a UCQ-rewriting of Q.

Suppose first that $\mathfrak{D} \models Q$ for an $\mathbf{S}$-database $\mathfrak{D}$. By Lemma 5.31 there is an $\mathbf{S}$-database $\mathfrak{B}$ of tree-width at most w such that $\mathfrak{B} \models Q$ and there is a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$. Fix a tree decomposition $\delta = (\mathcal{T}, (X_v)_{v \in T})$ of $\mathfrak{B}$. We can turn $\mathfrak{B}$ into an acyclic $(\mathbf{S} \cup \{C\})$-database by adding to $\mathfrak{B}$ all facts of the form $C(a_0, \dots, a_w)$ such that $\{a_0, \dots, a_w\} \subseteq X_v$ for some $v \in T$. Call the resulting database $\mathfrak{D}'$. Obviously, $\mathfrak{D}' \restriction \mathbf{S} \models Q$, and since $\mathfrak{D}'$ is clearly acyclic, we obtain $\mathfrak{D}' \models \eta_C(Q)$ by Lemma 4.55. Therefore, $\mathfrak{D}' \models p_i$ for some $i = 1, \dots, n$, whence $\mathfrak{D}' \models p_i'$ since p_i' does not have any occurrence of the predicate C. Hence, also $\mathfrak{B} \models p_i'$, whence $\mathfrak{D} \models p_i'$ follows since p_i' is closed under weak homomorphisms. This proves that $\mathfrak{D} \models q'$, as required.

Conversely, suppose now that $\mathfrak{D} \models q'$, i.e., $\mathfrak{D} \models p_i'$ for some $i = 1, \dots, n$ and an $\mathbf{S}$-database $\mathfrak{D}$. Let $\mathfrak{A}_i$ denote the $\mathbf{S}$-structure corresponding to p_i', and notice that there is thus a weak homomorphism from $\mathfrak{A}_i$ to $\mathfrak{D}$. Obviously, $\mathfrak{A}_i$ must have tree-width at most w by the construction of p_i'. Let $\mathfrak{A}_{p_i}$ be the structure corresponding to p_i. By definition, we have $\mathfrak{A}_{p_i} \models p_i$, whence $\mathfrak{A}_{p_i} \models \eta_C(Q)$. By Lemma 4.55, we obtain $\mathfrak{A}_{p_i} \restriction \mathbf{S} \models Q$. Obviously, the structures $\mathfrak{A}_{p_i} \restriction \mathbf{S}$ and $\mathfrak{A}_i$ are weakly isomorphic by construction. Hence, also $\mathfrak{A}_i \models Q$, whence $\mathfrak{D} \models Q$ follows, since Q is closed under weak homomorphisms. This concludes the proof of Lemma 5.30. □

We are now ready to conclude the proof of Theorem 5.3:

Proof of Theorem 5.3. Let $Q = (\mathbf{S}, \mathcal{O}, G)$ be an OMQ from $(\mathsf{FG}, \mathsf{BAQ})$, and let C be a fresh predicate of arity $\mathsf{wd}(\mathcal{O})$. To decide whether Q is first-order rewritable, we first compute $\eta_C(Q)$ which is clearly feasible in exponential time. We then decide whether $\eta_C(Q)$ is first-order rewritable using the algorithm devised in the proof of

Theorem 5.4. Although $\eta_C(Q)$ is of exponential size, we only increase the maximum arity of the predicates used by a polynomial factor. Hence, we can decide whether $\eta_C(Q)$ is first-order rewritable in 2ExpTime with respect to the size of Q. By Lemma 5.30, Q is first-order rewritable iff $\eta_C(Q)$ is. This concludes the proofs of the upper bounds stated in Theorem 5.3. As announced in the introductory part of this section, a 2ExpTime lower bound for the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{AQ})$ – in fact, for the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ – is given in Appendix B.3. □

5.6 Summary

In this chapter, we studied first-order rewritability for OMQs based on (frontier-)guarded sets of TGDs. We observed that the setting in the case of having relations of arity greater than two renders the problem of deciding first-order rewritability more challenging. The main result of this section arguably is that first-order rewritability for OMQs from $(\mathsf{FG}, \mathsf{UCQ})$ is 2ExpTime-complete. The (tight) upper bound was obtained by relying on the rather sophisticated model of cost automata. In addition, we provided non-optimal results for first-order rewritability by relying on classical automata methods. Though not optimal, we believe that the techniques developed to prove it are interesting in their own.

Directions for future research include an investigation on the worst-case size of possible UCQ-rewritings. For the DL-setting, this has been studied in [36], where it is shown that there is a class of OMQs based on $\mathcal{EL}$ whose smallest UCQ rewritings are triply exponential in size. Rough upper bounds for the size of UCQ-rewritings can be obtained by analyzing the algorithm based on alternating automata given in Section 5.3. However, we do not believe that the obtained bounds there are optimal. Concerning lower bounds, one can inherit the triply exponential lower bounds from [36] since guarded rules extend $\mathcal{EL}$. We conjecture that a tight bound for the size of rewritings of guarded-based OMQs (also frontier-guarded) is actually a tower of exponentials of height four. We do not go into details here and leave this question up to future research.

Apart from theoretical results, another question for future research concerns the practical implementation of procedures that decide first-order rewritability. First steps have been taken in [91] for the description logic $\mathcal{EL}$. However, for logics that support inverse roles and predicates of higher arity, practical algorithms are still unknown.

CHAPTER 6

Pushing the Warded Envelope Further

In the previous chapters we studied static analysis tasks for various ontology-mediated query languages. A common feature of the languages studied is their incapability of expressing all Datalog queries. As we observed, this is necessary, since both containment and first-order rewritability are undecidable for Datalog queries. However, it turns out that some languages designed for practical reasoning neglect this fact and prefer to expose the full power of recursion to their users, i.e., they capture all Datalog queries. To wit, recall that VADALOG [26, 27] is a language based on warded rules,[1] and is designed for complex reasoning over knowledge graphs that provides full Datalog recursion and ontological reasoning capabilities, while still having tractable data complexity.

Let us examine Datalog queries more closely. Given a Datalog query $Q = (\Pi, G)$, it is known that Q is equivalent to a possibly infinite union $q = \bigvee_{k \geq 1} q_k$ of conjunctive queries [55]. Roughly speaking, we can view q_k as a result of "unfolding" the rules of Q_i backwards. Each of these "unfoldings" gives rise to a tree whose nodes are labeled with atomic formulas such that (i) leaf nodes are labeled by atoms having an extensional predicate, and (ii) if a node is not a leaf node, is labeled by an atom α, and has children respectively labeled by $\beta_1, \ldots, \beta_k$, then the rule $\beta_1, \ldots, \beta_k \rightarrow \alpha$ arises from a Datalog rule of Π by renaming variables. Let us call such a tree a *proof tree*. The CQs q_k are then obtained by taking the conjunction of the leaf nodes of some proof tree, and the UCQ q is the exhaustive union of all such CQs that arise from proof trees.

It turns out that proof trees are convenient structures that can be employed for several decision problems for Datalog queries. To wit, in [55], the authors show how proof trees can be employed in order to provide an automata-based decision procedure for the problem asking whether a Datalog query is contained in a UCQ. Apart from that, proof trees represent computations that the Datalog query performs over an input database

[1] Recall that we denote by W the class of all finite warded sets of rules.

$\mathfrak{D}$. Indeed, if $\mathfrak{D} \models Q(\bar{a})$ for some tuple $\bar{a}$ over $\mathsf{adom}(\mathfrak{D})$, then there is a $k \geq 1$ such that $\mathfrak{D} \models q_k(\bar{a})$. Thus, the proof tree from which q_k is obtained describes the derivation of the goal $G(\bar{a})$ using the rules from Π.

The starting point of this chapter is the question whether we can provide a reasonable notion of proof tree for rule-based OMQs, and, if yes, whether we can employ this notion to solve important questions concerning rule-based OMQs. We provide affirmative answers to both of these questions. More specifically, we provide a notion of proof tree for rule-based OMQs that serves as an elegant technical tool for answering rule-based OMQs. Moreover, they also enable us to provide a strong sufficient criterion for rule-based OMQs that to be expressible as Datalog queries. This allows us to show that OMQs from $(\mathsf{W}, \mathsf{CQ})$ are Datalog rewritable. Apart from that, we identify a fragment of $(\mathsf{W}, \mathsf{CQ})$ and use proof trees to show that this fragment has, under standard complexity-theoretic assumptions, a query evaluation problem of lower complexity than that of warded rules. By studying this fragment, we contribute to the theory of Vadalog since its language is based on warded sets of rules.

Contributions Let us give more details on the contributions of this chapter:

Proof trees for OMQs We first generalize the notion of proof trees to capture rules having existential quantifiers in their heads. Just as proof trees for Datalog queries are trees labeled by atomic formulas, proof trees for OMQs will be trees labeled by CQs that are constructed according to prescribed rules. Essentially, proof trees for OMQs turn out to be closely related to resolution-based proof procedures such as XRewrite from Section 4.2. However, the primary goal of proof trees is not to be a data structure for a particular algorithm. Instead, the guiding principle of proof trees is to provide a generic tree structure, where each node label contains all and only the information necessary to prove that node label using resolution. Another way to view proof trees is to consider them as computation trees of a generic alternating query answering algorithm – the CQs labeling the nodes of a proof tree can be viewed as memory snapshots of this alternating algorithm when its computation progresses to the according node.

Having this machinery and its relationship to alternating query answering algorithms in place has interesting applications. Firstly, it allows us to devise simple and elegant query answering algorithms for popular classes of TGDs. In fact, we shall provide a simple alternating algorithm for answering OMQs based on warded sets of rules. This algorithm will turn out to run in polynomial space, whence by the equality APSpace = ExpTime, it follows that $\mathsf{Eval}(\mathsf{W}, \mathsf{CQ})$ is in ExpTime – this is a result first established in [84]. Secondly, proof trees allow us to identify a strong sufficient criterion for OMQs based on existential to be rewritable into Datalog queries. We use this result to show that OMQs from $(\mathsf{W}, \mathsf{CQ})$ are Datalog rewritable, and we show how to effectively construct Datalog rewritings for them. It is an open question whether the size of these rewritings is optimal, and it is open whether the sufficient criterion provided is also necessary, i.e., whether this criterion *characterizes* Datalog rewritability of rule-based OMQs.

A space-efficient fragment of warded rules As mentioned above, Vadalog is a system for performing complex reasoning tasks such as those required for advanced knowledge

graphs [26, 27]. Its main language is based on warded sets of rules, and it is designed to handle large amounts of data, since knowledge graphs tend to be rather huge in size. VADALOG is Oxford's contribution to the VADA research project[2] and has already found industrial applications. In the context of the development of VADALOG, the following question has emerged:

> *Can we limit the recursion allowed by wardedness in order to obtain a formalism that provides a convenient syntax for expressing useful statements, importantly, most of the scenarios provided by our industrial partners, and at the same time achieves efficiency in terms of space, in particular,* NLOGSPACE *data complexity?*

Let us stress that NLOGSPACE data complexity is the best that one can hope for, since navigational capabilities are vital for graph-based structures, and already graph reachability is well-known to be NLOGSPACE-hard. It is known that NLOGSPACE is contained in the class NC_2 of parallelizable problems. This means that reasoning in the more refined formalism that we are aiming is, in principle, parallelizable, unlike warded sets of rules, for which query answering is PTIME-complete in data complexity and thus (presumably) intrinsically sequential.

Extensive benchmark results based on a variety of scenarios are available for the VADALOG system, including (i) *ChaseBench* [30], a benchmark that targets data exchange and query answering problems (ii) *iBench*, a data exchange benchmark developed at the University of Toronto [7] (iii) *iWarded*, a benchmark specifically targeted at warded sets of TGDs, (iv) a benchmark based on DBPEDIA, and (v) a number of other synthetic and industrial scenarios [27]. Let us stress that all the above benchmarks contain only warded sets of TGDs. In fact, a good part of them are not *warded by chance*, i.e., they contain joins among harmful variables, which is one of the distinctive features of wardedness (cf. [27] and Subsection 3.4.2). After analyzing the above benchmarks, we observed that recursion is often used in a restricted way. Approximately 70% of the TGDs use recursion in as follows: the body of a TGD contains at most one atom whose predicate is mutually recursive with a predicate in the head. More specifically, approximately 55% of the TGDs directly use the above type of recursion, while 15% can be transformed into warded sets of TGDs that use recursion as explained above. This transformation relies on a standard elimination procedure of unnecessary non-linear recursion. For example, the rules

$$E(x, y) \rightarrow T(x, y), \quad T(x, y), T(y, z) \rightarrow T(x, z),$$

which compute the transitive closure of the extensional binary relation E using non-linear recursion, can be rewritten as the set

$$E(x, y) \rightarrow T(x, y), \quad E(x, y), T(y, z) \rightarrow T(x, z),$$

that uses *linear* recursion [111, 113], which allows only one occurrence of an intensional predicate to appear in each rule body. Interestingly, the type of recursion discussed above

[2] http://vada.org.uk

has been already studied in the context of Datalog, and is known as *piecewise linear* [2]. It is a refinement of linear recursion, which we already mentioned in the above example.

Based on this key observation, the following research questions have immediately emerged:

(i) Do warded sets of rules with piecewise linear recursion achieve space-efficiency for ontological query answering?[3]

(ii) Is the combination of wardedness and piecewise linearity justified? In other words, can we achieve the same with piecewise linear rules without the wardedness condition?

(iii) What is the expressiveness of the ontology-mediated language based on warded sets of rules with piecewise linear recursion relative to prominent query languages such as Datalog?

We provide answers to all these questions in the following. More precisely, our results can be summarized as follows:

(i) We show that query answering under warded sets of rules that use piecewise linear recursion is NLogSpace-complete in data complexity, and PSpace-complete in combined complexity. To establish this result, we heavily rely on proof trees for sets of TGDs discussed above. In particular, we show that query answering under warded sets of rules that use piecewise linear recursion boils down to the problem of checking whether a proof tree of a certain shape exists. This in turn can be done via a space-bounded non-deterministic algorithm.

(ii) It turns out that query answering under arbitrary sets of TGDs that use piecewise linear recursion is undecidable. We show this via a simple reduction from the standard unbounded tiling problem. Hence, the combination of wardedness and piecewise linearity is indeed justified from an algorithmic point of view.

(iii) We again resort to the machinery of proof trees to show that OMQs based on warded piecewise linear sets of TGDs are in fact always rewritable into piecewise linear Datalog queries.

Outline After this introductory part, we introduce the central notion of proof tree in Section 6.1. In Section 6.2, we study piecewise linear warded sets of TGDs, and we show that query answering under them is PSpace-complete in combined, and NLogSpace-complete in data complexity. Moreover, we re-establish the complexity of query answering under warded sets of TGDs, i.e., ExpTime-completeness in combined, and PTime-completeness in data complexity. We also show that query answering under piecewise linear sets of TGDs, without assuming they are warded, is undecidable. In Section 6.3, using proof trees, we study rewritability of rule-based OMQs into Datalog queries. Finally, we close this chapter in Section 6.4 and provide directions for future research. Some of the proofs are deferred to Appendix B.4.

[3]The idea of combining wardedness with piecewise linearity has been already mentioned in the invited paper [26], where the obtained formalism is called *strongly warded*.

6.1 Proof Trees

It is known that given a CQ q and a set $\mathcal{O}$ of TGDs, we can unfold q using the TGDs of $\mathcal{O}$ into an infinite union of CQs $q_{\mathcal{O}}$ such that, for every database $\mathfrak{D}$, $\mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D}) = q_{\mathcal{O}}(\mathfrak{D})$ (cf. [82, 99] and the algorithm XRewrite presented in Appendix A). Let us clarify that in our context, an "unfolding" – which is essentially a resolution step – is more complex than in the context of Datalog due to the existentially quantified variables in the heads of TGDs. The intention underlying our notion of proof tree is to encode the sequence of CQs, generated during the unfolding of q with $\mathcal{O}$, in a tree that leads to a certain CQ q' of $q_{\mathcal{O}}$ in such a way that each intermediate CQ, as well as q', is carefully decomposed into smaller subqueries that form the nodes of the tree, while the root corresponds to q and the leaves to q'. As we shall see, if we focus on well-behaved classes of TGDs such as (piecewise linear) warded sets of TGDs, we can establish upper bounds on the size of these subqueries, which in turn allows us to devise space-bounded algorithms for query answering.

In what follows, we define the notion of proof tree (Definition 6.5), and establish its correspondence with query answering (Theorem 6.6). To this end, we need to introduce the main building blocks of a proof tree: *chunk-based resolution* (Definition 6.1), a *query decomposition* step (Definition 6.3), and the notion of *specialization* for CQs (Definition 6.4).

Notational conventions Before presenting the three main building blocks of proof trees, let us fix some notation that we are going to use throughout this chapter. We will denote conjunctive queries by rule-based expressions of the form

$$q(x_1, \ldots, x_n) \leftarrow \alpha_1, \ldots, \alpha_m,$$

where $\alpha_1, \ldots, \alpha_m$ is a list of relational atoms. The expression $q(x_1, \ldots, x_n)$ is the *head predicate* of the CQ, and when naming the CQ presented above we shall, if unambiguous, simply refer to it via its head predicate. The sequence of variables $x_1, \ldots, x_n$ is allowed to contain repetitions of variables, and we call this sequence the *answer variables* of $q(x_1, \ldots, x_n)$. Likewise, we write $\mathsf{body}(q)$ for the set $\{\alpha_1, \ldots, \alpha_m\}$ of its *body atoms*. Let $y_1, \ldots, y_k$ be the variables that occur in the body atoms of q, and let $v_1, \ldots, v_n$ be a sequence of pairwise distinct variables that do not occur in q. The meaning of q is simply the first-order formula

$$\exists y_1, \ldots, y_k \, (v_1 = x_1 \wedge \cdots \wedge v_n = x_n \wedge \alpha_1 \wedge \cdots \wedge \alpha_m),$$

which is a standard conjunctive query as defined in Subsection 2.3.2. All the notions defined for CQs defined in previous chapters immediately carry over to this slightly modified definition. We can, however, assume that the body atoms of q do *not* contain any equality atoms, since we can identify variables and we can repeat variables in the sequence $x_1, \ldots, x_n$ (cf. also Remark 2.14 in Subsection 2.3.2).

Concerning sets of TGDs, throughout this chapter, we assume that none of the TGDs employed contains equality atoms in its body. Thus, we also assume that none of the

TGDs has a body equal to $\top$, since $\top$ is defined as $\exists x\, x = x$. Notice that if a TGD has at least one relational atom in its body, we can remove equality atoms by simply identifying variables. These omission of bodies equal to $\top$ is made for technical reasons, but all the results hold, with mild modifications, also for the case where we allow $\top$ for rule bodies.

Chunk-based resolution Let A and B be sets of relational atoms that mention only constants and variables. The sets A and B *unify* if there is a substitution γ, which is the identity on const, called *unifier for A and B*, such that $\gamma(A) = \gamma(B)$. A *most general unifier* (MGU) for A and B is a unifier $\gamma_{A,B}$ for A and B such that, for each unifier γ for A and B, $\gamma = \gamma' \circ \gamma_{A,B}$ for some substitution γ'. It is easy to see that, if two sets of atoms unify, then there exists always a MGU which is unique modulo renaming variables. Hence, we shall always denote by $\gamma_{A,B}$ a canonical most general unifier of A and B and call it *the* most general unifier of A and B.

Given a CQ $q(\bar{x})$ and a set of atoms $S \subseteq \mathsf{body}(q)$, we say that a variable $y \in \mathsf{var}(S)$ is *S-shared* if $y \in [\bar{x}]$, or $y \in \mathsf{var}(\mathsf{body}(q) \setminus S)$. A *chunk unifier* of q with a TGD τ (where q and τ do not share variables) is a triple (S_1, S_2, γ), where $\varnothing \subset S_1 \subseteq \mathsf{body}(q)$, $\varnothing \subset S_2 \subseteq \mathsf{head}(\tau)$, and γ is a unifier for S_1 and S_2 such that, for each $x \in \mathsf{var}(S_2) \cap \mathsf{var}_\exists(\tau)$,

(i) $\gamma(x) \notin \mathsf{const}$, i.e., $\gamma(x)$ is not a constant, and

(ii) for all variables $y \neq x$, if $\gamma(x) = \gamma(y)$, then y occurs in S_1 and is not S_1-shared.

The chunk unifier (S_1, S_2, γ) is *most general* (MGCU) if γ is an MGU for S_1 and S_2. Notice that the variables of $\mathsf{var}_\exists(\tau)$ occurring in S_2 unify (via γ) only with variables of S_1 that are not S_1-shared. This ensures that S_1 is a "chunk" of q that can be resolved as a whole via τ using γ.[4] Without the additional conditions on the substitution γ, we may get unsound resolution steps. Consider, e.g., the CQ and TGD

$$q(x) \leftarrow R(x, y), S(y) \quad \text{and} \quad P(x') \rightarrow \exists y'\, R(x', y').$$

Resolving the atom $R(x, y)$ in the query q with the given TGD using the substitution $\gamma = \{x \mapsto x', y \mapsto y'\}$ is an unsound step since the shared variable y is "lost." This is because y' is unified with the shared variable y. On the other hand, $\{R(x, y), S(y)\}$ can be resolved with the TGD $\tau\colon P(x') \rightarrow \exists y'\,(R(x', y') \wedge S(y'))$ using γ. In fact, the chunk unifier is $(\mathsf{body}(q), \mathsf{head}(\tau), \gamma)$.

Definition 6.1. Let $q(\bar{x})$ be a CQ and τ a TGD. A *τ-resolvent* of q is a CQ q' with

$$\mathsf{body}(q') = \gamma((\mathsf{body}(q) \setminus S_1) \cup \mathsf{body}(\tau)),$$

for some MGCU (S_1, S_2, γ) of q with τ.

Query decomposition As discussed above, the purpose of a proof tree is to encode a finite branch of the unfolding of a CQ q with a set $\mathcal{O}$ of TGDs, which is obtained by applying chunk-based resolution. Such a branch is a sequence $q_0, \ldots, q_n$ of CQs, where $q = q_0$, while, for each $i \in [n]$, q_i is a τ-resolvent of q_{i-1} for some $\tau \in \mathcal{O}$.

[4] A similar notion known as *piece unifier* is defined in [99].

Here is a simple example, which will serve as a running example for the remainder of the section, that illustrates the notion of unfolding.

Example 6.2. Consider the set $\mathcal{O}$ of TGDs consisting of

$$\begin{aligned} R(x) &\to \exists y\, T(y, x), \\ T(x, y), S(y, z) &\to T(x, z), \\ T(x, y), P(y) &\to G. \end{aligned}$$

and the CQ that simply asks whether the atomic query G is entailed, i.e., the CQ $q \leftarrow G$. Since the unfolding of q with $\mathcal{O}$ should give the correct answer for *every* input database, and thus for databases of the form

$$\{R(c^n), S(c^n, c^{n-1}), \ldots, S(c^2, c^1), P(c^1)\}, \quad \text{for } n > 0,$$

one of its branches should be $q = q_0, q_1, \ldots, q_n$, where

$$q_1 \leftarrow T(x, y^1), P(y^1),$$

obtained by resolving q_0 using the third TGD,

$$q_i \leftarrow T(x, y^i), S(y^i, y^{i-1}), \ldots, S(y^2, y^1), P(y^1), \quad \text{for } 1 < i < n,$$

obtained by resolving q_{i-1} using the second TGD, and

$$q_n \leftarrow R(y^n), S(y^n, y^{n-1}), \ldots, S(y^2, y^1), P(y^1),$$

obtained by resolving q_{n-1} using the first TGD. ⊣

At this point, one may think that the proof tree that encodes the branch $q_0, \ldots, q_n$ of the unfolding of q with $\mathcal{O}$ is the finite labeled path $v_0, \ldots, v_n$, where each v_i is labeled by q_i. However, another crucial goal of such a proof tree, which is not achieved via the naive path encoding, is to split each resolvent q_i, for $i > 0$, into smaller subqueries $q_i^1, \ldots, q_i^{n_i}$ – which are essentially the children of q_i –, in such a way that they can be processed independently by resolution. The crux of this encoding is that it provides us with a mechanism for keeping the CQs that must be processed by resolution small. It should be clear from Example 6.2 that, by following the naive path encoding without splitting the resolvents into smaller subqueries, we may get CQs of unbounded size.

The key question here is how a CQ q can be decomposed into subqueries that can be processed independently. The subtlety is that, after splitting q, occurrences of the same variable may be separated into different subqueries. Thus, we need a way to ensure that a variable in q, which appears in different subqueries after the splitting, is indeed treated as the same variable, i.e., it has the same meaning. We deal with this issue by restricting those variables of q whose occurrences can be separated during the splitting step. More specifically, we can only separate occurrences of an answer variable. This relies on the convention that answer variables correspond to fixed constant values of const, and thus

their name is "frozen" and never renamed by subsequent resolution steps. Hence, we can separate occurrences of an answer variable into different subqueries, i.e., different branches of the proof tree, without losing the connection between them.

Summing up, the idea underlying query decomposition is to split the CQ at hand into smaller subqueries that keep all the occurrences of a non-answer variable together, but with the freedom of separating occurrences of answer variables.

Definition 6.3. For a CQ $q(\bar{x})$, a *decomposition* of q is a set of CQs $\{q_1(\bar{x}_1), \ldots, q_n(\bar{x}_n)\}$, where $n \geq 1$ and $\bigcup_{i \in [n]} \mathsf{body}(q_i) = \mathsf{body}(q)$, such that, for each $i \in [n]$, the following hold:

(i) $\bar{x}_i$ is the restriction of $\bar{x}$ to the variables in q_i.

(ii) For every $\alpha, \beta \in \mathsf{body}(q)$, if $\alpha \in \mathsf{body}(q_i)$ and $\mathsf{var}(\alpha) \cap \mathsf{var}(\beta) \not\subseteq [\bar{x}]$, then $\beta \in \mathsf{body}(q_i)$.

Query specialization From the above discussion, one expects that a proof tree of a CQ q w.r.t. a set $\mathcal{O}$ of TGDs can be constructed by starting from q, which is the root, and applying two steps, resolution and decomposition. Unfortunately, this is not enough for our purposes as we may run into the problem that some of the subqueries will mistakenly remain large since we have no way to realize that a non-answer variable corresponds to a fixed constant value, which in turn would allow us to "freeze" its name and separate different occurrences of it during decomposition steps. This is illustrated by Example 6.2. Observe that the size of the CQs $\{q_i\}_{i>0}$ grows arbitrarily, while our query decomposition has no effect on them since they are Boolean queries, and thus we cannot split them into smaller subqueries.

This issue can be resolved by having an intermediate step between resolution and decomposition, the so-called *specialization* step. A *specialization* of a CQ is obtained by converting some non-answer variables of it into output variables, while keeping their name, or taking the name of an existing output variable.

Definition 6.4. Let $q(\bar{x})$ be a CQ with $\mathsf{body}(q) = \{\alpha_1, \ldots, \alpha_n\}$. A *specialization* of q is a CQ

$$q'(\bar{v}) \leftarrow \rho_{\bar{z}}(\alpha_1), \ldots, \rho_{\bar{z}}(\alpha_n),$$

where $[\bar{x}] \subseteq [\bar{v}]$, $[\bar{z}] \cap [\bar{v}] = \varnothing$, and $\rho_{\bar{z}}$ is a substitution from $[\bar{z}]$ to $[\bar{v}]$.

Consider, for example, the CQ q_1 from Example 6.2

$$q_1 \leftarrow T(x, y^1), P(y^1)$$

obtained by resolving $q = q_0$ using the third TGD. The query decomposition cannot split it onto smaller subqueries since the variable y^1 is not an answer variable, and thus, all its occurrences should be kept together. We can consider the following specialization of q_1

$$q_1(y^1) \leftarrow T(x, y^1), P(y^1),$$

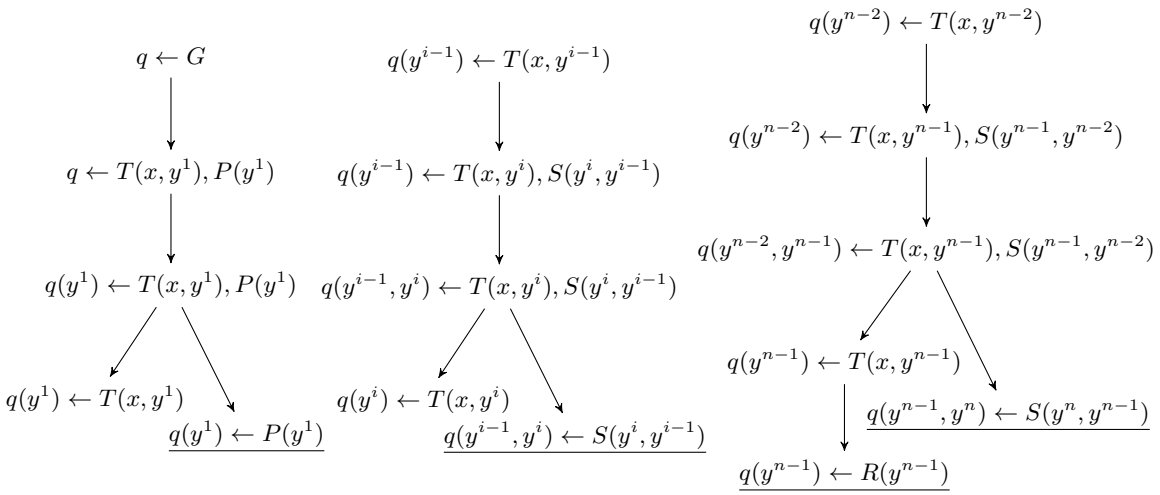

Figure 6.1: Partial trees of the proof tree that encodes the branch $q = q_0, \ldots, q_n$ of the unfolding of q with $\mathcal{O}$ from Example 6.2.

which simply converts y^1 into an answer variable, and now we can decompose it into the atomic queries

$$q_1(y^1) \leftarrow T(x, y^1) \quad \text{and} \quad q_1(y^1) \leftarrow P(y^1).$$

Proof trees We are now ready to introduce our new notion of proof tree. But let us first explain the high-level idea by exploiting our running example. Consider the set $\mathcal{O}$ of TGDs and the CQ q from Example 6.2. The branch $q_0, \ldots, q_n$ of the unfolding of q with $\mathcal{O}$ given in Example 6.2 is encoded via a proof tree that consists of trees $\mathcal{T}_1, \ldots, \mathcal{T}_{n-1}$ such that (i) each $\mathcal{T}_i$ is a rooted tree with two leaf nodes, and (ii) for $i = 1, \ldots, n-2$, $\mathcal{T}_{i+1}$ is rooted in the left-most child of $\mathcal{T}_i$. The actual trees are depicted in Figure 6.1: the left one is $\mathcal{T}_1$, the middle one are the $\mathcal{T}_i$ ($i = 2, \ldots, n-2$), while the right one is $\mathcal{T}_{n-1}$. For $i = 1 \ldots, n-1$, the child of the root of $\mathcal{T}_i$ is obtained via resolution, then we specialize it by converting the variable y^i into an answer variable, and then we decompose the specialized CQ into two subqueries. In the tree $\mathcal{T}_{n-1}$, we also apply an additional resolution step in order to obtain the leaf node $q(y^{n-1}) \leftarrow R(y^{n-1})$. The underlined CQs are actually the subqueries that represent the CQ q_n of the unfolding. Indeed, the conjunction of the atoms occurring in the underlined CQs is precisely the CQ q_n.

We proceed to give the formal definition. Given a partition $\pi = \{S_1, \ldots, S_m\}$ of a set of variables, we write eq_π for the substitution that maps the variables of S_i to the same variable x_i, where x_i is a distinguished element of S_i. We should not forget the convention that answer variables cannot be renamed, and thus, a resolution step should use a MGCU that preserves the output variables. In particular, given a CQ q and a TGD τ, a τ-resolvent of q is an *IDO-(τ-)resolvent*, if the underlying MGCU uses a substitution that is the identity on the answer variables of q. Finally, given a TGD τ and some arbitrary object o (e.g., o can be the node of a tree, or an integer number), we

write τ_o for the TGD obtained by renaming each variable x in τ to x_o. This is a simple mechanism for uniformly renaming the variables of a TGD in order to avoid undesirable clutter among variables during a resolution step.

Definition 6.5. Let $q(\bar{x})$ be a CQ with $\mathsf{body}(q) = \{\alpha_1, \ldots, \alpha_n\}$, and $\mathcal{O}$ a set of TGDs. A *proof tree* of q w.r.t. $\mathcal{O}$ is a triple $\mathcal{P} = (\mathcal{T}, \lambda, \pi)$, where $\mathcal{T}$ is a finite rooted tree, λ a labeling function that assigns a CQ to each node of $\mathcal{T}$, and π a partition of $[\bar{x}]$, such that, for each node v of $\mathcal{T}$:

(i) If v is the root node of $\mathcal{T}$, then $\lambda(v)$ is the CQ

$$q_v(\mathsf{eq}_\pi(\bar{x})) \leftarrow \mathsf{eq}_\pi(\alpha_1), \ldots, \mathsf{eq}_\pi(\alpha_m).$$

(ii) If v has only one child u, $\lambda(u)$ is an IDO-τ_v-resolvent of $\lambda(v)$ for some $\tau \in \mathcal{O}$, or a specialization of $\lambda(v)$.

(iii) If v has $k > 1$ children $u_1, \ldots, u_k$, then $\{\lambda(u_1), \ldots, \lambda(u_k)\}$ is a decomposition of $\lambda(v)$.

Assuming that $v_1, \ldots, v_m$ are the leaf nodes of $\mathcal{T}$, the *CQ induced by* $\mathcal{P}$ is defined as

$$q_{\mathcal{P}}(\mathsf{eq}_\pi(\bar{x})) \leftarrow \beta_1, \ldots, \beta_\ell,$$

where $\{\beta_1, \ldots, \beta_\ell\} = \bigcup_{i=1}^{m} \mathsf{body}(\lambda(v_i))$. Moreover, we call π the *equality type* of $\mathcal{P}$.

The purpose of the partition π is to indicate that some answer variables correspond to the same constant value – this is why variables in the same set of π are unified via the substitution eq_π. This unification step is crucial in order to safely use – in subsequent resolution steps – substitutions that are the identity on the answer variables. If we omitted this initial unification step, we would lose important resolution steps and would thus be incomplete for query answering purposes. The main result of this section, which exposes the connection between proof trees and query answering, follows (its proof can be found in Appendix B.4):

Theorem 6.6. *Consider a database $\mathfrak{D}$, a set $\mathcal{O}$ of TGDs, a CQ $q(\bar{x})$, and a tuple $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$. The following are equivalent:*

(i) $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$.

(ii) *There exists a proof tree $\mathcal{P}$ of q w.r.t. $\mathcal{O}$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$.*

6.2 Piecewise Linearity

Linear Datalog is a key fragment of Datalog queries that achieves a good balance between expressivity and complexity [111, 113]. In particular, answering linear Datalog queries is PSpace-complete in combined, and NLogSpace-complete in data complexity. Formally, a Datalog query is *linear* if each of its rule has at most one atom in its body whose predicate is intensional. After analyzing several real-life examples of warded sets of

TGDs, we observed that the use of recursion goes beyond the syntax of linear Datalog queries. On the other hand, most of the examples coming from our industrial partners use recursion in a restrictive way: each TGD has at most one body atom whose predicate is *mutually recursive* with a predicate occurring in the head of the TGD. Interestingly, this more liberal version of linear recursion has been already investigated in the context of Datalog, and is known as *piecewise linear* recursion [2].

Let us formally define the class of piecewise linear sets of TGDs. To this end, we need to define when two predicates are mutually recursive, which is defined using its predicate graph. Recall that the *predicate graph* of a set $\mathcal{O}$ of TGDs, denoted $\mathsf{pg}(\mathcal{O})$, is a directed graph whose set of nodes is $\mathsf{sig}(\mathcal{O})$, and that has an edge from a predicate P to a predicate R iff there exists a TGD $\tau \in \mathcal{O}$ such that P occurs in $\mathsf{body}(\tau)$ and R occurs in $\mathsf{head}(\tau)$. Two predicates $P, R \in \mathsf{sig}(\mathcal{O})$ are *mutually recursive* w.r.t. $\mathcal{O}$ if there exists a directed cycle in $\mathsf{pg}(\mathcal{O})$ of length at least one that contains both P and R (i.e., R is reachable from P by a non-trivial directed path and vice versa). We are now ready to define piecewise linearity for TGDs:

Definition 6.7. A set $\mathcal{O}$ of TGDs is *piecewise linear* if, for each TGD $\tau \in \mathcal{O}$, there exists at most one atom in $\mathsf{body}(\tau)$ whose predicate is mutually recursive with a predicate in $\mathsf{head}(\tau)$. We denote by PWL the class of piecewise linear sets of TGDs.

We will focus on piecewise linear sets of TGDs that are also warded. The main result of this section reads as follows:

Theorem 6.8. *The problem* $\mathsf{Eval}(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$ *is* PSpace*-complete in combined complexity, and* NLogSpace*-complete in data complexity.*

The lower bounds are inherited from the complexity of answering linear Datalog queries. We thus mainly focus on the upper bounds. As we shall see, our notion of proof tree leads to a space-bounded algorithm that allow us to establish the upper bounds stated in Theorem 6.8.

Besides proving Theorem 6.8, we will also establish the following result:

Theorem 6.9. $\mathsf{Eval}(\mathsf{W}, \mathsf{CQ})$ *is* ExpTime*-complete in combined complexity, and* PTime*-complete in data complexity.*

Theorem 6.9 has already been established in [84]. However, our algorithm is significantly simpler than the one given in [84], and it reveals the main property of warded sets of TGDs that we are going to use later for devising Datalog rewritings.

Our strategy toward proofs for Theorems 6.8 and 6.9 is to first strengthen Theorem 6.6 of Subsection 6.2.1 for warded and piecewise linear warded sets of TGDs. We then use these strengthened versions of Theorem 6.6 in Subsection 6.2.2 to prove Theorems 6.8 and 6.9.

6.2.1 Query Answering via Proof Trees

Theorem 6.6 states that checking whether a tuple $\bar{a}$ is a certain answer boils down to deciding whether there exists a proof tree $\mathcal{P}$ such that $\bar{a}$ is an answer to the CQ induced by $\mathcal{P}$ over the given database. Of course, the latter is an undecidable problem in general. However, if we focus on (piecewise linear) warded sets of TGDs, it turns out that it suffices to check for the existence of a well-behaved proof tree with certain properties, which in turn allows us to devise space-bounded decision procedures.

We proceed to make this more precise. For technical clarity, we assume, without loss of generality, TGDs with only one atom in the head. It is easy to check that we can always convert, a (piecewise linear) warded set of TGDs in linear time into one with single-atom heads, while certain answers are preserved (cf. the transformation given in Section 4.3).

Piecewise linear warded sets of TGDs For piecewise linear warded sets of TGDs, we can strengthen Theorem 6.6 by focussing on a certain class of proof trees that enjoy two properties: (i) they have a path-like structure, and (ii) the size of the CQs that label their nodes is bounded by a polynomial. The first property is formalized via a *linearity* property on proof trees. Let $\mathcal{P} = (\mathcal{T}, \lambda, \pi)$ be a proof tree of a CQ q w.r.t. a set $\mathcal{O}$ of TGDs. We call $\mathcal{P}$ *linear* if, for each node $v \in T$, it holds that v has at most one child that is not a leaf. For example, the proof tree given above, which consists of the partial trees depicted in Figure 6.1, is linear. The second property relies on the maximum size of a CQ occurring as a label of $\mathcal{P}$. The *node-width* of $\mathcal{P}$ is

$$\mathsf{nwd}(\mathcal{P}) \coloneqq \max_{v \in T}\{|\lambda(v)|\},$$

i.e., the size of the largest CQ that labels a node of $\mathcal{T}$.

Before defining the polynomial we use to bound the node-width of proof trees, let us introduce some additional technical notions. Let $\mathcal{O} \in \mathsf{PWL}$. For a predicate $P \in \mathsf{sig}(\mathcal{O})$, we write $\mathsf{rec}(P)$ for the set of predicates of $\mathsf{sig}(\mathcal{O})$ that are mutually recursive to P according to $\mathsf{pg}(\mathcal{O}) = (V, E)$. Let $\ell_{\mathcal{O}} \colon \mathsf{sig}(\mathcal{O}) \to \mathbb{N}$ be the unique function that satisfies

$$\ell_{\mathcal{O}}(P) = \max\{\ell_{\mathcal{O}}(R) \mid (R, P) \in E, R \notin \mathsf{rec}(P)\} + 1,$$

with $\ell_{\mathcal{O}}(P)$ being the *level* (w.r.t. $\mathcal{O}$) of P. If α is an atom whose predicate is P, we write $\ell_{\mathcal{O}}(\alpha)$ for $\ell_{\mathcal{O}}(P)$. We say that $\mathcal{O}$ is in *level-wise normal form*, if the following condition is satisfied for every $\sigma \in \mathcal{O}$: if the head predicate of σ has level k, then each of the predicates occurring in $\mathsf{body}(\sigma)$ has level k or $k-1$.

Lemma 6.10. *Every piecewise linear set of TGDs $\mathcal{O}$ can be transformed into a piecewise linear $\mathcal{O}^+$ in level-wise normal form such that, for any CQ $q(\bar{x})$ over $\mathsf{sig}(\mathcal{O})$ and any database $\mathfrak{D}$ over $\mathsf{sig}(\mathcal{O})$, we have that $\mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D}) = \mathsf{cert}_{q,\mathcal{O}^+}(\mathfrak{D})$.*

Moreover, $\ell(\mathcal{O}^+) \leq \ell(\mathcal{O})$, and $\mathcal{O}^+$ can be obtained from $\mathcal{O}$ by introducing polynomially many fresh predicates and rules.

PROOF. Suppose $\mathcal{O}$ is not in level-wise normal form. Then there is a $\sigma \in \mathcal{O}$ of the form

$$R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m) \to \psi,$$

such that $n_i \coloneqq \ell_{\mathcal{O}}(P) - \ell_{\mathcal{O}}(R_i) > 1$, where P is the predicate of the single atom belonging to ψ – let us call such a σ *bad* in the following. For $i = 1, \ldots, m$, we add to $\mathcal{O}^+$ the rules

$$\begin{aligned} R_i(\bar{x}_i) &\to R_i^{\sigma,1}(\bar{x}_i), \\ R_i^{\sigma,k}(\bar{x}_i) &\to R_i^{\sigma,k+1}(\bar{x}_i), \quad \text{for } k \in [n_i - 1]. \end{aligned}$$

Here, the predicates $R_i^{\sigma,k}$ are all fresh. Notice that $\ell_{\mathcal{O}^+}(R_i^{\sigma,k}) = \ell_{\mathcal{O}}(R_i) + k$, and thus $\ell_{\mathcal{O}^+}(R_i^{\sigma,n_i-1}) = \ell_{\mathcal{O}}(P) - 1$.

Now we add to $\mathcal{O}^+$ the rule

$$R_1^{\sigma,n_1}(\bar{x}_1), \ldots, R_m^{\sigma,n_m}(\bar{x}_m) \to \psi.$$

Notice that $\ell_{\mathcal{O}^+}(P) = \ell_{\mathcal{O}}(P)$. We do this step exhaustively for all bad rules $\sigma \in \mathcal{O}$. Moreover, we add to $\mathcal{O}^+$ those rules from $\mathcal{O}$ that are not bad without any change.

It is not hard to check that this construction introduces only polynomially many fresh predicates and rules. Moreover, it is clear that $\mathcal{O}^+$ is still piecewise linear, and that $\mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D}) = \mathsf{cert}_{q,\mathcal{O}^+}(\mathfrak{D})$ for every CQ $q(\bar{x})$ over $\mathsf{sig}(\mathcal{O})$ and every database $\mathfrak{D}$. □

We now define

$$f_{\mathsf{W}\cap\mathsf{PWL}}(q, \mathcal{O}) \coloneqq (|q| + 1) \cdot \max\{\ell_{\mathcal{O}}(P) : P \in \mathsf{sig}(\mathcal{O})\} \cdot \max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\}.$$

We can now strengthen Theorem 6.6 as follows:

THEOREM 6.11. *Consider a database $\mathfrak{D}$, a set $\mathcal{O} \in \mathsf{WARD} \cap \mathsf{PWL}$ of TGDs in level-wise normal form, a CQ $q(\bar{x})$, and $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$. The following are equivalent:*

(i) $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$.

(ii) *There is a linear proof tree $\mathcal{P}$ of q w.r.t. $\mathcal{O}$ with $\mathsf{nwd}(\mathcal{P}) \leq f_{\mathsf{W}\cap\mathsf{PWL}}(q, \mathcal{O})$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$.*

We shall prove Theorem 6.11 below. Before that, we discuss strengthen Theorem 6.6 for warded sets of TGDs.

Warded sets of TGDs In the case of arbitrary warded sets of TGDs, we cannot focus only on linear proof trees. Nevertheless, we can still bound the node-width of the proof trees that we need to consider by the following polynomial:

$$f_{\mathsf{W}}(q, \mathcal{O}) \coloneqq 2 \cdot \max\{|q|, \max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\}\}.$$

Theorem 6.6 can be strengthened as follows:

THEOREM 6.12. *Consider a database $\mathfrak{D}$, a set $\mathcal{O} \in \mathsf{W}$ of TGDs, a CQ $q(\bar{x})$, and $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$. The following are equivalent:*

(i) $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$.

(ii) *There exists a proof tree* $\mathcal{P}$ *of* q *w.r.t.* $\mathcal{O}$ *with* $\mathsf{nwd}(\mathcal{P}) \leq f_{\mathsf{W}}(q, \mathcal{O})$ *such that* $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$.

For the proofs of Theorems 6.11 and 6.12 we need some additional technical notions first. Our strategy for these proofs is as follows:

▶ We introduce the auxiliary notion of *chase tree*, which can be seen as a concrete instantiation of a proof tree. It serves as an intermediate structure between proof trees and chase derivations, which allows us to use the chase as our underlying technical tool. The notions of linearity and node-width can be naturally defined for chase trees as well.

▶ We then show that the existence of a (linear) chase tree for the image of q to $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ with node-width at most m implies the existence of a (linear) proof tree $\mathcal{P}$ of q w.r.t. $\mathcal{O}$ with node-width at most m such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$ (Lemma 6.15).

▶ We finally show that, if the given tuple of constants $\bar{a}$ is a certain answer to the given CQ q w.r.t. the given database $\mathfrak{D}$ and (piecewise linear) warded set $\mathcal{O}$ of TGDs, then there exists a (linear) chase tree for the image of q to $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ such that its node-width respects the bounds given in the above theorems (Lemmas 6.16 and 6.17).

Thus, our goal is to introduce chase trees in order to make use of them as explained above. The definition of chase trees relies on several intermediate technical notions: we first need to introduce *chase graphs*, then introduce *unravelings* of chase graphs, and finally introduce *unfolding* and *decomposition steps* for sets of atoms in the unraveling of the chase graph. All these notions will be detailed in the following.

The chase graph and its unraveling Fix a database $\mathfrak{D}$ and a set $\mathcal{O}$ of TGDs. Let $\pi = \mathfrak{J}_0, \mathfrak{J}_1, \ldots$ be a chase sequence for $\mathfrak{D}$ and $\mathcal{O}$, and suppose that $\mathfrak{J}_{i+1}$, for $i \geq 0$, results from $\mathfrak{J}_i$ by an application of (h_i, τ_i). The *chase graph for* π is a directed edge-labeled graph $\mathcal{G}_\pi = (V, E, \mu)$ whose set of nodes V is the set of all facts occurring in $\bigcup_{i \geq 0} \mathfrak{J}_i$ and that has an edge $(\alpha, \beta) \in E$ labeled (via μ) with (h_i, τ_i) iff $\alpha \in h_i(\mathsf{body}(\tau_i))$ and β is a fact of $\mathfrak{J}_{i+1}$ but not one of $\mathfrak{J}_i$. In other words, α has an edge to β labeled (h_i, τ_i) if β is derived using $\alpha \in h_i(\mathsf{body}(\tau_i))$ and if β is new in the sense that it has not been derived before in the chase sequence π. Notice that $\mathcal{G}_\pi$ depends on π, however, we write $\mathcal{G}^{\mathfrak{D},\mathcal{O}}$ for the chase graph for $\pi_{\mathfrak{D},\mathcal{O}}$, where $\pi_{\mathfrak{D},\mathcal{O}}$ is the globally fixed chase sequence for $\mathfrak{D}$ and $\mathcal{O}$.

Consider the chase graph $\mathcal{G}^{\mathfrak{D},\mathcal{O}} = (V, E, \mu)$ for $\mathfrak{D}$ and $\mathcal{O}$ and a node $v \in V$. The *unraveling of* $\mathcal{G}^{\mathfrak{D},\mathcal{O}}$ *around* v is the directed tree $\mathcal{G}_v^{\mathfrak{D},\mathcal{O}} = (V_v, E_v)$, where

- V_v is the set of all finite sequences

$$\bar{v} \coloneqq v_1 v_2 \cdots v_n$$

of nodes from V such that $v_1 = v$ and $v_{i+1} E v_i$ for all $i = 1, \ldots, n-1$. We write $\mathsf{last}(\bar{v})$ for v_n.

- For $\bar{v} = v_1 \cdots v_n$ and $\bar{v}' = v_1 \cdots v_n v_{n+1}$ we have that $\bar{v} E_v \bar{v}'$ iff $v_{n+1} E v_n$.

Given a set $\Theta \subseteq V$ of nodes (i.e., facts from $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$, the *unraveling of* $\mathcal{G}^{\mathfrak{D},\mathcal{O}}$ *around* Θ is the directed node- and edge-labeled forest $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}} = (V_\Theta, E_\Theta, \mu_\Theta)$, where $V_\Theta \coloneqq \bigcup_{v \in \Theta} V_v$

and $E_\Theta \coloneqq \bigcup_{v \in \Theta} E_v$. For the definition of the labeling function μ_Θ, we need some auxiliary notions first. Intuitively, $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ is a forest-like reorganization of the facts of $\mathsf{chase}(\mathfrak{D},\mathcal{O})$ that are needed to derive Θ. Due to its forest-like nature, it may contain multiple copies of facts of $\mathsf{chase}(\mathfrak{D},\mathcal{O})$. Most importantly, these multiple copies are supplied with redundant fresh labeled nulls formalized via the notion of *t-connectivity* that we are going to define in the following.

A *pseudo path* in $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ is a sequence of nodes $v_1, \dots, v_n$ from V_Θ such that, for all $1 \leq i < n$, one of $v_i E_\Theta v_{i+1}$, $v_{i+1} E_\Theta v_i$, or $|v_i| = |v_{i+1}| = 1$ holds. Thus, a pseudo path in the forest $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ is a path in $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ where we consider the root nodes of the forest to be connected. Notice that there is thus a unique shortest pseudo path between any two nodes of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$.

Let v and w be nodes from $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$. If v and w lie in the same tree component $\mathcal{T}$, then we denote by $\mathsf{gca}(v,w)$ the singleton set consisting of the greatest common ancestor of v and w w.r.t. the natural tree order $\prec_\mathcal{T}$. If, on the other hand, v and w are respectively located in different tree components $\mathcal{T}_1$ and $\mathcal{T}_2$, then we set $\mathsf{gca}(v,w) = \varnothing$.

Given a term t, we say that v and w are *t-connected in* $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$, if one of the following holds:

(i) t is a constant, and t occurs in $\mu(\mathsf{last}(v))$ and in $\mu(\mathsf{last}(w))$.

(ii) t occurs in $\mu(\mathsf{last}(u))$ for every $u \notin \mathsf{gca}(v,w)$ that lies on the unique shortest pseudo path between v and w in $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$.

Clearly, the notion of being t-connected defines an equivalence relation among the nodes of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$. We write $[v]_t$ for the according equivalence class of $v \in V_\Theta$.[5] Moreover, if a is a constant, then since $[v]_a = [w]_a$ for any $v, w \in V_\Theta$, we identify the class $[v]_a$ simply with a. For $[v]_t$ with t being a labeled null, we call $[v]_t$ a *(labeled) null* as well.

For $v \in V_\Theta$ and $\mu(\mathsf{last}(v)) = R(t_1, \dots, t_k)$, we define

$$\mu_\Theta(v) \coloneqq R([v]_{t_1}, \dots, [v]_{t_k}).$$

Moreover, if $v E_\Theta w$ and $\mu(\mathsf{last}(v), \mathsf{last}(w)) = (\tau, h)$, we set

$$\mu_\Theta(v,w) \coloneqq (\tau, h^*), \quad \text{where } h^* \colon x \longmapsto [v]_{h(x)}.$$

We write $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$ for the structure corresponding to the set of facts $\{\mu_\Theta(v) \mid v \in V_\Theta\}$. Notice that since we identify $[v]_a$ with a when a is a constant, this entails that if a fact $R(a_1, \dots, a_n)$, where $a_1, \dots, a_n \in \mathsf{const}$, lies on some path in $\mathcal{G}^{\mathfrak{D},\mathcal{O}}$ leading from a fact of $\mathfrak{D}$ to some atom in Θ, then $R(a_1, \dots, a_n)$ is also a fact of $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$.

Given a node v of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$, we denote by $\mathsf{succ}_{\tau,h}(v)$ the set of labels of all children of v whose edge from v is labeled (τ, h). Accordingly, we write $\mathsf{succ}(v)$ for the set of all labels of children of v. When using this notation, we assume that the particular unraveling we are referring to is clear from context. We write $\alpha_1, \dots, \alpha_k \Rightarrow_{\tau,h} \beta$ if there is a node v of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ such that $\mu_\Theta(v) = \beta$ and $\mathsf{succ}_{\tau,h}(v) = \{\alpha_1, \dots, \alpha_k\}$. Accordingly, we write $\alpha_1, \dots, \alpha_k \Rightarrow \beta$ if there is a $\tau \in \mathcal{O}$ and some h such that $\alpha_1, \dots, \alpha_k \Rightarrow_{\tau,h} \beta$.

[5] Formally, we set $[v]_t \coloneqq \{(u,t) \mid u \text{ is } t\text{-connected to } v\}$ to ensure that $[v]_t = [w]_{t'}$ only if $t = t'$.

Lemma 6.13. *Consider a node v of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ and suppose that $\mathsf{succ}_{\tau,h}(v) = \{\beta_1, \ldots, \beta_k\}$ for some $\tau \in \mathcal{O}$ and some h. Then if some labeled null occurs in β_i, it either occurs also in $\mu_\Theta(v)$, or it does not occur in the label of any node of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ that is not a descendant of v.*

Proof. Immediate by the definitions of the equivalence classes $[v]_t$ and the labeling function μ_Θ. □

Let us remark that there is an obvious homomorphism h_Θ from Θ to $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$ defined by $h_\Theta \colon t \mapsto [v_0]_t$, where v_0 is any of the root nodes whose label has an occurrence of t. Notice that h_Θ is well-defined, since $[v_0]_t = [w_0]_t$ for all $t \in \mathsf{adom}(\Theta)$ and all root nodes v_0, w_0 of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ that have an occurrence of t in their labels.

Blocking, depth, and rank Consider again the unraveling $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ around Θ, and let α be a fact from $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, and suppose that $\beta_1, \ldots, \beta_k \Rightarrow \alpha$. For a set of facts $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, we say that the *application of $\beta_1, \ldots, \beta_k \Rightarrow \alpha$ is blocked in* Γ, if there is a labeled null occurring in α that occurs in $\Gamma \setminus \{\alpha\}$, but that does not occur in any of the facts $\beta_1, \ldots, \beta_k$.

Given a node v of $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$, the *depth* of v, denoted $\mathsf{dp}(v)$, is defined inductively as follows:

$$\mathsf{dp}(v) \coloneqq \max\{\mathsf{dp}(u) \mid u \text{ is a child node of } v \text{ in } \mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}\} + 1,$$

For a fact α from $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, we define the *depth* of α as

$$\mathsf{dp}(\alpha) \coloneqq \min\{\mathsf{dp}(v) \mid \mu_\Theta(v) = \alpha\}.$$

Notice that $\mathsf{dp}(\alpha) = 1$ iff α a leaf node in $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$. For a set of facts Γ, we set

$$\mathsf{dp}(\Gamma) \coloneqq \max\{\mathsf{dp}(\alpha) \mid \alpha \in \Gamma\}.$$

The *rank* of v, denoted $\mathrm{rk}(v)$, is defined as

$$\mathrm{rk}(v) \coloneqq \begin{cases} 1 & \text{if } v \text{ is a leaf node in } \mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}, \\ \sum_{u \in \mathsf{succ}(v)} \mathrm{rk}(u), & \text{otherwise.} \end{cases}$$

For a fact α from $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, we define the *rank* of α as

$$\mathrm{rk}(\alpha) \coloneqq \min\{\mathrm{rk}(v) \mid \mu_\Theta(v) = \alpha\}.$$

For a set of facts Γ, we set

$$\mathrm{rk}(\Gamma) \coloneqq \sum_{\alpha \in \Gamma} \mathrm{rk}(\alpha).$$

Intuitively, the rank of a fact gives a measure of how many database facts are used to derive that fact.

Both the depth and rank will be used as induction parameters in the proofs of this section. We remark that these parameters are, of course, always defined relative to a particular unraveling $\mathcal{U}$ of $\mathcal{G}^{\mathfrak{D},\mathcal{O}}$. The concrete unraveling of $\mathcal{G}^{\mathfrak{D},\mathcal{O}}$ they refer to will always be clear from context in the following, and we adhere to no particular additional notation to indicate the reference to $\mathcal{U}$.

Chase trees Given sets of facts $\Gamma, \Gamma' \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, we say that Γ' is an *unfolding of* Γ, if there are $\alpha \in \Gamma$ and $\beta_1, \ldots, \beta_k \in \Gamma'$ such that the following conditions hold:

(i) It holds that $\beta_1, \ldots, \beta_k \Rightarrow \alpha$, and its application is not blocked in Γ.

(ii) $\Gamma' = (\Gamma \setminus \{\alpha\}) \cup \{\beta_1, \ldots, \beta_k\}$.

Given a non-empty set of facts $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, a *decomposition* of Γ is a set $\{\Gamma_1, \ldots, \Gamma_n\}$, where $n \geq 1$, of non-empty subsets of Γ such that (i) $\Gamma = \bigcup_{i=1}^{n} \Gamma_i$, and (ii) $i \neq j$ implies that Γ_i and Γ_j do not share a labeled null.

Definition 6.14. Consider a database $\mathfrak{D}$, a set of TGDs $\mathcal{O}$, and a set of facts $\Theta \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$. A *chase tree* for a set of facts $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$ (w.r.t. $\mathcal{G}^{\mathfrak{D},\mathcal{O}}_{\Theta}$) is a pair $\mathcal{C} = (\mathcal{T}, \lambda)$, where $\mathcal{T}$ is a finite rooted tree, and λ a labeling function that assigns sets of facts from $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$ to each node of $\mathcal{T}$, such that, for each node $v \in T$, the following conditions hold:

(i) If v is the root node of $\mathcal{T}$, then $\lambda(v) = \Gamma$.

(ii) Suppose v has the children $u_1, \ldots, u_k$ ($k \geq 1$). Then one of the following holds:
   (a) $k = 1$ and $\lambda(u_k)$ is an unfolding of $\lambda(v)$.
   (b) $\{\lambda(u_1), \ldots, \lambda(u_k)\}$ is a decomposition of $\lambda(v)$.

(iii) If v is a leaf node, then $\lambda(v) \subseteq \mathfrak{D}$.

The *node-width* of $\mathcal{C}$ is $\mathsf{nwd}(\mathcal{C}) \coloneqq \max_{v \in T}\{|\lambda(v)|\}$. Moreover, we say that $\mathcal{C}$ is *linear* if each node of $\mathcal{T}$ has at most one child that is not a leaf node.

The following lemma – whose proof can be found in Appendix B.4 – states that transitioning from the existence of a chase tree allows us to construct an appropriate proof tree that respects according size bounds:

Lemma 6.15. *Consider a set of facts $\Theta \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$, let $q(\bar{x})$ be a CQ, and $\bar{a}$ a tuple of constants such that $h(\mathsf{body}(q)) \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$ and $h(\bar{x}) = \bar{a}$ for some homomorphism h. If there is a (linear) chase tree $\mathcal{C}$ for $h(\mathsf{body}(q))$ with $\mathsf{nwd}(\mathcal{C}) \leq m$, then there is a (linear) proof tree $\mathcal{P}$ for q w.r.t. $\mathcal{O}$ such that $\mathsf{nwd}(\mathcal{P}) \leq m$ and $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$.*

We now proceed to establish the crucial lemmas that are used to prove Theorems 6.11 and 6.12. We begin with the warded case. Recall that, for a set of facts Γ and a set of TGDs $\mathcal{O}$, we have

$$f_{\mathsf{W}}(\Gamma, \mathcal{O}) \coloneqq 2 \cdot \max\{|\Gamma|, \max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\}\}.$$

Lemma 6.16. *Let $\Theta \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$ be a set of facts and consider a set of facts $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$. If $\mathcal{O} \in \mathsf{W}$, then there exists a chase tree $\mathcal{C}$ for Γ such that $\mathsf{nwd}(\mathcal{C}) \leq f_{\mathsf{W}}(\Gamma, \mathcal{O})$.*

Proof. We are construct a chase tree $\mathcal{C}$ for Γ (w.r.t. $\mathcal{G}^{\mathfrak{D},\mathcal{O}}_{\Theta}$), whose node-width is bounded by $m_\Gamma \coloneqq f_{\mathsf{W}}(\Gamma, \mathcal{O})$, by induction on $\mathsf{dp}(\Gamma)$.

Assume first that $\mathsf{dp}(\Gamma) = 1$. Then we must have $\Gamma \subseteq \mathfrak{D}$, i.e., Γ solely consists of facts of $\mathfrak{D}$, and a trivial chase tree for Γ is thus the tree consisting of a single node whose

root is labeled with Γ. The node-width of that tree is trivially at most m_Γ.

Suppose now that $\mathsf{dp}(\Gamma) = n + 1$. We perform a subsidiary induction on the number of facts in Γ that have depth $n + 1$.

Suppose first that there is exactly one fact $\alpha \in \Gamma$ that has depth $n+1$. Let $\beta_1, \ldots, \beta_k$ be such that such that the application of $\beta_1, \ldots, \beta_k \Rightarrow \alpha$ is not blocked in Γ. (It is easy to see that such an application cannot be blocked, since α is of maximal depth.) Since $\mathcal{O}$ is warded, there is at most one fact β_i such that all the nulls occurring in α are also present in β_i. Moreover, by the construction of the unraveling $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$, β_i does not share any other nulls with any of the β_j, for $j \neq i$. In case such a fact β_i exists, we set $\Gamma' \coloneqq (\Gamma \setminus \{\alpha\}) \cup \{\beta_i\}$ and $\Gamma'' \coloneqq \{\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_k\}$. Otherwise, we set $\Gamma' \coloneqq \Gamma \setminus \{\alpha\}$ and $\Gamma'' \coloneqq \{\beta_1, \ldots, \beta_k\}$. In both cases, we see that $\{\Gamma', \Gamma''\}$ is a decomposition of Γ, since the nulls that do not appear in α, yet that appear in some fact among $\beta_1, \ldots, \beta_k$, are all fresh by Lemma 6.13. Moreover, we know that there are no nulls that are present in α, yet not in any of the $\beta_1, \ldots, \beta_k$, since the application of $\beta_1, \ldots, \beta_k \Rightarrow \alpha$ is not blocked in Γ.

Notice that we have $\mathsf{dp}(\Gamma') \leq n$ and $\mathsf{dp}(\Gamma'') \leq n$. Hence, by induction hypothesis, there are chase trees $\mathcal{C}'$ and $\mathcal{C}''$ that are respectively for Γ' and Γ''. We build a chase tree $\mathcal{C}$ for Γ by labeling its root v_0 with Γ, and declaring that v_0 has one child v_1 whose label is $\Gamma' \cup \Gamma''$. Furthermore, v_1 has two children, v' and v'', that are respectively labeled with Γ' and Γ''. Notice that $m_{\Gamma'} \leq m_\Gamma$ and that $m_{\Gamma''} \leq m_\Gamma$. Moreover, $|\Gamma' \cup \Gamma''| \leq |\Gamma| + \max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\} \leq m_\Gamma$. Thus, $\mathcal{C}$ is a chase tree for Γ with the desired bound on the node-width.

The induction step of the subsidiary induction is performed *mutatis mutandis* to the base case, and we thus omit it for brevity. □

Proof of Theorem 6.12. Consider a CQ $q(\bar{x})$ and a tuple $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$ such that $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$. We need to show that if $\mathcal{O} \in \mathsf{W}$, then there exists a proof tree $\mathcal{P}$ of q w.r.t. $\mathcal{O}$ with $\mathsf{nwd}(\mathcal{P}) \leq f_{\mathsf{W}}(q, \mathcal{O})$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$.

By hypothesis, there is a homomorphism h such that $h(\mathsf{body}(q)) \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$ and $h(\bar{x}) = \bar{a}$. Let Θ_q be the set of facts $h(\mathsf{body}(q))$. Recall that there is a homomorphism h_{Θ_q} that maps Θ_q to $\mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta_q)$. Thus, the homomorphism $h' \coloneqq h_{\Theta_q} \circ h$ is such that $h'(\mathsf{body}(q)) \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta_q)$ and $h'(\bar{x}) = \bar{a}$. By Lemma 6.16, there exists a chase tree $\mathcal{C}$ for $h'(\mathsf{body}(q))$ with $\mathsf{nwd}(\mathcal{C}) \leq f_{\mathsf{W}}(h'(\mathsf{body}(q)), \mathcal{O})$. By Lemma 6.15, there exists a proof tree $\mathcal{P}$ of q w.r.t $\mathcal{O}$ with $\mathsf{nwd}(\mathcal{P}) \leq f_{\mathsf{W}}(h'(\mathsf{body}(q)), \mathcal{O}) \leq f_{\mathsf{W}}(q, \mathcal{O})$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$, and the claim follows. □

For the remainder of this subsection, we focus on the (technically more intriguing) proof of Theorem 6.11. For a set of facts Γ and a set of TGDs $\mathcal{O} \in \mathsf{PWL}$, recall that

$$f_{\mathsf{W}\cap\mathsf{PWL}}(\Gamma, \mathcal{O}) \coloneqq (|\Gamma| + 1) \cdot \max\{\ell_{\mathcal{O}}(P) : P \in \mathsf{sig}(\mathcal{O})\} \cdot \max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\}.$$

Our goal is to show that we can find linear chase trees for all sets $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, whose node-width is bounded by $f_{\mathsf{W}\cap\mathsf{PWL}}(\Gamma, \mathcal{O})$, provided $\mathcal{O}$ is warded and piecewise linear. This is the content of the following lemma, which is the main ingredient toward a proof of Theorem 6.11:

Lemma 6.17. *Let $\Theta \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$ be a set of facts and consider a set of facts $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$. If $\mathcal{O} \in \mathsf{W} \cap \mathsf{PWL}$ is in level-wise normal form, then there exists a linear chase tree $\mathcal{C}$ for Γ such that $\mathsf{nwd}(\mathcal{C}) \leq f_{\mathsf{W}\cap\mathsf{PWL}}(\Gamma, \mathcal{O})$.*

Fix a set of TGDs $\mathcal{O} \in \mathsf{W}\cap\mathsf{PWL}$. We are going to write $\ell(\alpha)$ for $\ell_{\mathcal{O}}(\alpha)$ in the following, and $\ell(\mathcal{O})$ for $\max\{\ell_{\mathcal{O}}(P) \mid P \in \mathsf{sig}(\mathcal{O})\}$. Moreover, let us abbreviate $\max\{|\mathsf{body}(q)| : \tau \in \mathcal{O}\}$ by b in the following. Before proceeding to the actual proof Lemma 6.17, we need some additional technical notions first.

Conflict-free facts The *stratification of* $\mathcal{O}$ is a partition $(\mathcal{S}_1, \ldots, \mathcal{S}_{\ell(\mathcal{O})})$ of $\mathsf{sig}(\mathcal{O})$ such that, for $P \in \mathsf{sig}(\mathcal{O})$ we have that $P \in \mathcal{S}_k$ iff $\ell_{\mathcal{O}}(P) = k$. We let $(\Gamma[\mathcal{S}_1], \ldots, \Gamma[\mathcal{S}_{\ell(\mathcal{O})}])$ denote the unique partition of $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$ such that for every $\alpha \in \Gamma$ we have $\alpha \in \Gamma[\mathcal{S}_k]$ if and only if $k = \ell(\alpha)$.

Given a set of facts Γ, the *join graph* of Γ is the undirected edge-labeled graph $\mathbb{J}(\Gamma)$ whose set of nodes is Γ and that has an edge between α and β labeled with terms $t_1, \ldots, t_k$ iff (i) $\alpha \neq \beta$, and (ii) $t_1, \ldots, t_k$ exhausts all terms that jointly occur in α and β.

We say that a fact $\alpha \in \Gamma[\mathcal{S}_i]$ is *conflict-free* (in Γ), if there is no fact $\beta \in \bigcup_{j>i} \Gamma[\mathcal{S}_j]$ such that there is a path from α to β in $\mathbb{J}(\Gamma)$ that has an occurrence of a null in each of its edge labels – we shall call such a path *conflicting*. Hence, a fact that only has constants as arguments is trivially conflict-free. Notice also that every Γ trivially has conflict-free facts, namely those that are among $\Gamma[\mathcal{S}_r]$, where r is the largest number such that $\Gamma[\mathcal{S}_r]$ is non-empty.

Size measures For a set of facts $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$, we let

$$l_\Gamma \coloneqq \min\{k \mid \exists \alpha \in \Gamma[\mathcal{S}_k] \colon \alpha \text{ is conflict-free}\}.$$

Given Γ, we let

$$(a_1^\Gamma, \ldots, a_{\ell(\mathcal{O})}^\Gamma) \quad \text{and} \quad (n_1^\Gamma, \ldots, n_{\ell(\mathcal{O})}^\Gamma)$$

be sequences of natural numbers such that $a_i^\Gamma = |\Gamma[\mathcal{S}_i]|$, and n_i^Γ is the number of facts from $\Gamma[\mathcal{S}_i]$ that are *not* conflict-free.

We set

$$m_\Gamma \coloneqq \sum_{i=1}^{\ell(\mathcal{O})} b \cdot \max\{a_i^\Gamma, 1 + n_i^\Gamma\}.$$

We are now ready to prove Lemma 6.17:

Proof of Lemma 6.17. Throughout the proof, fix a set of facts $\Gamma \subseteq \mathcal{U}(\mathcal{G}^{\mathfrak{D},\mathcal{O}}, \Theta)$. We are going to prove that there exists a chase tree $\mathcal{C}$ for Γ (w.r.t. $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$) whose node-width is bounded by m_Γ. This suffices to prove the claim, since $m_\Gamma \leq f_{\mathsf{W}\cap\mathsf{PWL}}(\Gamma, \mathcal{O})$. To this end, we shall proceed by induction on $\mathrm{rk}(\Gamma)$.

Base case. Suppose first that $\mathrm{rk}(\Gamma) = 1$. The *level depth* of Γ is

$$\mathsf{ldp}(\Gamma) \coloneqq \max\{\mathsf{dp}(\alpha) \mid \alpha \in \Gamma[\mathcal{S}_{l_\Gamma}] \text{ and } \alpha \text{ is conflict-free}\}.$$

We perform an auxiliary induction on $\mathsf{ldp}(\Gamma)$ in order to prove our claim.

Suppose that $\mathsf{ldp}(\Gamma) = 1$. Since $\mathrm{rk}(\Gamma) = 1$, this means that actually $\Gamma = \{\alpha\}$ for some fact $\alpha \in \mathfrak{D}$. A linear chase tree for Γ of node-width at most $m_\Gamma \geq |\Gamma|$ is simply the tree with a single node whose label is Γ.

Suppose now that $\mathrm{rk}(\Gamma) = 1$ and $\mathsf{ldp}(\Gamma) = n + 1$. This means that $\Gamma = \{\alpha\}$ for some α such that $\beta \Rightarrow \alpha$ for some β with $\mathsf{dp}(\beta) = n$. Notice that the application of $\beta \Rightarrow \alpha$ is trivially not blocked. We let $\Gamma' \coloneqq \{\beta\}$, and we observe that $\mathsf{ldp}(\Gamma) = n$. By induction hypothesis, there is a linear chase tree $\mathcal{C}'$ for Γ' whose node-width is bounded by $m_{\Gamma'}$. Let $\mathcal{C}$ be the linear chase tree whose root v_0 is labeled with Γ such that v_0 has a single child labeled with Γ'. It is easy to check that $m_\Gamma = m_{\Gamma'} = b$. Thus, $\mathcal{C}$ is a linear chase tree for Γ whose node-width is bounded by m_Γ, as desired.

Induction step. Suppose now that $\mathrm{rk}(\Gamma) = m + 1$. A *reduction sequence for* Γ is a finite sequence $(\Gamma_0, \Gamma_1, \ldots, \Gamma_k, \Xi)$ of sets of facts that satisfy the following conditions:

(i) $\Gamma_0 = \Gamma$,

(ii) $m_{\Gamma_0} \geq m_{\Gamma_1} \geq \cdots \geq m_{\Gamma_k}$,

(iii) each Γ_{i+1} is an unfolding of Γ_i, where $i = 0, \ldots, k-2$,

(iv) $\mathrm{rk}(\Gamma_k) \leq m$, and

(v) $\Xi = \Gamma_{k-1} \cap \mathfrak{D}$ and $\Gamma_k = \Gamma_{k-1} \setminus \Xi$. Thus, $\{\Gamma_k, \Xi\}$ is a decomposition of Γ_{k-1}.

Lemma 6.18. *There exists a reduction sequence for* Γ.

Proof. We shall again perform a subsidiary induction on $\mathsf{ldp}(\Gamma)$ to show that, for every $n \geq 1$, if $\mathsf{ldp}(\Gamma) = n$, then there exists a reduction sequence for Γ.

So suppose first that $\mathsf{ldp}(\Gamma) = 1$. This means that Γ contains at least one fact from $\mathfrak{D}$. Let $\Gamma \cap \mathfrak{D} = \{\alpha_1, \ldots, \alpha_k\}$. We form the reduction sequence $\pi \coloneqq (\Gamma, \Gamma', \{\alpha_1, \ldots, \alpha_k\})$, where $\Gamma' \coloneqq \Gamma \setminus \{\alpha_1, \ldots, \alpha_k\}$. Notice that $\mathrm{rk}(\Gamma') \leq m$. In order to prove that π is indeed a reduction sequence for Γ, it remains to be shown that $m_{\Gamma'} \leq m_\Gamma$. Observe that

$$\begin{aligned} m_\Gamma - m_{\Gamma'} &= \sum_{i=1}^{\ell(\mathcal{O})} b \cdot \max\{a_i^\Gamma, 1 + n_i^\Gamma\} - \sum_{i=1}^{\ell(\mathcal{O})} b \cdot \max\{a_i^{\Gamma'}, 1 + n_i^{\Gamma'}\} \\ &= b \cdot \underbrace{\max\{a_{l_\Gamma}^\Gamma, 1 + n_{l_\Gamma}^\Gamma\}}_{= a_{l_\Gamma}^\Gamma \text{ since } k \geq 1} - b \cdot \max\{ \underbrace{a_{l_\Gamma}^{\Gamma'}}_{= a_{l_\Gamma}^\Gamma - k}, \overbrace{1 + n_{l_\Gamma}^{\Gamma'}}^{= 1 + n_{l_\Gamma}^\Gamma} \}. \end{aligned}$$

Now if $a_{l_\Gamma}^\Gamma - k \geq 1 + n_{l_\Gamma}^\Gamma$, then we obtain

$$m_\Gamma - m_{\Gamma'} = b a_{l_\Gamma}^\Gamma - b(a_{l_\Gamma}^\Gamma - k) = bk > 0,$$

and if $a_{l_\Gamma}^\Gamma - k < 1 + n_{l_\Gamma}^\Gamma$, we obtain

$$m_\Gamma - m_{\Gamma'} = b a_{l_\Gamma}^\Gamma - b(1 + n_{l_\Gamma}^\Gamma) \geq 0, \quad \text{since } 1 + n_{l_\Gamma}^\Gamma \leq a_{l_\Gamma}^\Gamma.$$

Thus, in both cases we have that $m_\Gamma \leq m_{\Gamma'}$.

Suppose now that $\mathsf{ldp}(\Gamma) = n+1$. Let $\alpha_1, \ldots, \alpha_k$ enumerate all the conflict-free facts from $\Gamma[\mathcal{S}_{l_\Gamma}]$ of depth $n+1$. For $i \in [k]$, let $\beta_{i,1}, \ldots, \beta_{i,k_i}$ be facts such that the application of $\beta_{i,1}, \ldots, \beta_{i,k_i} \Rightarrow \alpha_i$ is not blocked in Γ – these facts exist, since the α_i are conflict-free in Γ and of maximal depth. Let $\tau_i \in \mathcal{O}$ and h_i be such that $\beta_{i,1}, \ldots, \beta_{i,k_i} \Rightarrow_{\tau_i,h_i} \alpha_i$. Let $\Gamma_0 \coloneqq \Gamma$ and $\Gamma_i \coloneqq (\Gamma_{i-1} \setminus \{\alpha_i\}) \cup \{\beta_{i,1}, \ldots, \beta_{i,k_i}\}$, for all $i \in [k]$.

Claim 6.19. *For $i = 1, \ldots, k$, either $l_{\Gamma_i} = l_{\Gamma_{i-1}}$ or $l_{\Gamma_i} = l_{\Gamma_{i-1}} - 1$.*

Proof. As a preliminary remark, notice that, by our normal form assumptions, we know that $\ell(\beta_{i,j}) \in \{\ell(\alpha_i), \ell(\alpha_i) - 1\}$ for all $j = 1, \ldots, k_i$.

We distinguish cases. Suppose first that α_i has no labeled nulls as arguments. In this case, all nulls appearing in $\beta_{i,1}, \ldots, \beta_{i,k_i}$ must be new in the sense that they do not appear in Γ_{i-1} (cf. Lemma 6.13). Thus, at least one of $\beta_{i,1}, \ldots, \beta_{i,k_i}$ must be conflict-free in Γ_i and hence $l_{\Gamma_i} = l_{\Gamma_{i-1}}$ or $l_{\Gamma_i} = l_{\Gamma_{i-1}} - 1$ since $\mathcal{O}$ is in level-wise normal form. This proves the claim for this case.

Suppose now that α_i has labeled nulls as arguments. Since $\mathcal{O}$ is warded, it must be the case that τ_i has at most one atom in its body (the ward) that shares nulls with α_i at all. Suppose first that τ_i has no ward in its body. All labeled nulls that appear in α_i are thus not present in $\beta_{i,1}, \ldots, \beta_{i,k_i}$ (Lemma 6.13) and, moreover, all the nulls that appear in $\beta_{i,1}, \ldots, \beta_{i,k_i}$ do not appear in Γ_{i-1}, whence it follows that at least one of $\beta_{i,1}, \ldots, \beta_{i,k_i}$ must be conflict-free in Γ_i. Hence, $l_{\Gamma_i} = l_{\Gamma_{i-1}}$ or $l_{\Gamma_i} = l_{\Gamma_{i-1}} - 1$.

Suppose now that $\beta_{i,j}$ is the ward among $\beta_{i,1}, \ldots, \beta_{i,k_i}$. If $k_i \geq 2$, then the claim is immediate, since the facts from $\{\beta_{i,1}, \ldots, \beta_{i,k_i}\} \setminus \{\beta_{i,j}\}$ are all conflict-free in Γ_i due to the fact that the nulls they contain do not appear in Γ_{i-1} by Lemma 6.13. Suppose now that $k_i = 1$. Then $\beta_{i,j}$ is conflict-free in Γ_i or not. In the former case, we immediately obtain $l_{\Gamma_i} \leq l_{\Gamma_{i-1}}$ and thus $l_{\Gamma_i} = l_{\Gamma_{i-1}}$ or $l_{\Gamma_i} = l_{\Gamma_{i-1}} - 1$ by our level-wise normal form assumption. In the latter case, $\beta_{i,j}$ is connected to some fact $\gamma \in \bigcup_{r > \ell(\beta_{i,j})} \Gamma_i[\mathcal{S}_r]$ in the join graph of Γ_{i-1} via a path that is conflicting. Notice again that all the nulls that occur in $\beta_{i,j}$ are either only contained in $\beta_{i,j}$ or they also appear in α_i (cf. Lemma 6.13). Hence, there is also a conflicting path in $\mathbb{J}(\Gamma_{i-1})$ that connects α_i and γ. Since α_i was assumed to be conflict-free in Γ_{i-1}, it follows that $\gamma \in \Gamma_{i-1}[\mathcal{S}_1] \cup \cdots \cup \Gamma_{i-1}[\mathcal{S}_{\ell(\alpha)}]$. Moreover, this entails that γ is conflict-free in Γ_{i-1}. Now γ must also be conflict-free in Γ_i, since all the nulls present in Γ_i that do not occur in Γ_{i-1} must be solely contained in $\beta_{i,j}$. Thus, $l_{\Gamma_i} \leq l_{\Gamma_{i-1}}$ and so $l_{\Gamma_i} = l_{\Gamma_{i-1}}$ or $l_{\Gamma_i} = l_{\Gamma_{i-1}} - 1$, since $\mathcal{O}$ is in level-wise normal form. □

Claim 6.20. *Assume that $l_{\Gamma_i} = l_{\Gamma_{i-1}} - 1$ and that there is a $\beta_{i,j}$ that is not conflict-free in Γ_i. Then there is a $\gamma \in \Gamma_i[\mathcal{S}_{l_{\Gamma_{i-1}}}]$ that is conflict free in Γ_i.*

Proof. Notice that $\ell(\beta_{i,j}) = l_{\Gamma_i}$, since all the nulls appearing in $\beta_{i,j}$ either occur also in α_i or are fresh. Hence, if we had $\ell(\beta_{i,j}) = l_{\Gamma_{i-1}} = l_{\Gamma_i} + 1$, then α would not be conflict-free as well. Now since $\beta_{i,j}$ is not conflict-free, it follows that $\beta_{i,j}$ is connected via a conflicting path to some $\gamma \in \bigcup_{r > \ell(\beta_{i,j})} \Gamma_i[\mathcal{S}_r]$. It is easy to see that, in fact, we must have $\ell(\gamma) = \ell(\beta_{i,j}) + 1 = l_{\Gamma_{i-1}}$. Now $\beta_{i,j}$ therefore shares a null with γ, and thus γ shares a null with α_i. Hence, γ must be conflict-free since α_i is. □

Claim 6.21. *Suppose that* $l_{\Gamma_i} = l_{\Gamma_{i-1}} = 1$. *Then* $k_i = 1$.

Proof. Since $\mathsf{ldp}(\alpha_i) = n + 1 \geq 2$, $\ell(\alpha_i) = 1$ only if α_i is derived by a sequence of facts that have the same predicate as α_i. Hence, we must necessarily have $k_i = 1$, since $\mathcal{O}$ is piecewise linear. □

Claim 6.22. *Assume that* $l = l_{\Gamma_i} = l_{\Gamma_{i-1}} > 1$. *Then* $|\Gamma_i[\mathcal{S}_l] \setminus \Gamma_{i-1}[\mathcal{S}_{l-1}]| \leq 1$, *that is,* $a_{l-1}^{\Gamma_i} - a_{l-1}^{\Gamma_{i-1}} \leq 1$.

Proof. Suppose $a_{l-1}^{\Gamma_i} - a_{l-1}^{\Gamma_{i-1}} > 1$, i.e., at least two of the $\beta_{i,1}, \ldots, \beta_{i,k_i}$ have level $l-1$. Since $l_{\Gamma_i} = l_{\Gamma_{i-1}}$, this means that they are actually not conflict-free, whence it follows that they share a null with α_i. But this is impossible, since at most one of the $\beta_{i,1}, \ldots, \beta_{i,k_i}$ can share nulls with α_i by wardedness. □

Claim 6.23. *For* $i = 1, \ldots, k$ *it holds that* $m_{\Gamma_i} \leq m_{\Gamma_{i-1}}$.

Proof. For the sake of readability, let us set $l \coloneqq l_{\Gamma_{i-1}}$, $l' = l_{\Gamma_i}$, $\Gamma \coloneqq \Gamma_{i-1}$, and $\Gamma' \coloneqq \Gamma_i$ in the following. We proceed by distinguishing cases.

Case 1. Suppose first that $l = l'$. We distinguish subcases.

Subcase 1.1. Suppose first that $l = l' = 1$. Then,

$$\begin{aligned} m_\Gamma - m_{\Gamma'} &= b \cdot \max\{a_1^\Gamma, 1 + n_1^\Gamma\} - b \cdot \max\{a_1^{\Gamma'}, 1 + n_1^{\Gamma'}\} \\ &= b a_1^\Gamma - b a_1^{\Gamma'} \\ &= b a_1^\Gamma - b(a_1^\Gamma + k_i - 1) = 0, \quad \text{since } k_i = 1 \text{ by Claim 6.21.} \end{aligned}$$

Hence, $m_{\Gamma'} \leq m_\Gamma$.

Subcase 1.2. Suppose that $l > 1$. Then one can verify that

$$\begin{aligned} m_\Gamma - m_{\Gamma'} = b \cdot \max\{a_l^\Gamma, 1 + n_l^\Gamma\} + b \cdot \max\{a_{l-1}^\Gamma, 1 + n_{l-1}^\Gamma\} \\ - b \cdot \max\{a_l^{\Gamma'}, 1 + n_l^{\Gamma'}\} - b \cdot \max\{a_{l-1}^{\Gamma'}, 1 + n_{l-1}^{\Gamma'}\}. \end{aligned}$$

Notice that, by Claim 6.22, we know that $a_{l-1}^{\Gamma'} \leq a_{l-1}^\Gamma + 1$. Now if $a_{l-1}^{\Gamma'} = a_{l-1}^\Gamma$, we immediately obtain $m_\Gamma - m_{\Gamma'} \geq 0$ due to $a_l^\Gamma = a_l^{\Gamma'}$.

Therefore, assume now that $a_{l-1}^{\Gamma'} = a_{l-1}^\Gamma + 1$, and also observe that we must have $n_{l-1}^{\Gamma'} = a_{l-1}^{\Gamma'}$ and $a_{l-1}^\Gamma = n_{l-1}^\Gamma$, since Γ and Γ' do not have any conflict-free facts of level $l-1$ by assumption. Moreover, notice that $n_l^{\Gamma'} = n_l^\Gamma$. From this bulk of information, we obtain

$$\begin{aligned} m_\Gamma - m_{\Gamma'} &= b \cdot \max\{a_l^{\Gamma'} + k_i, 1 + n_l^{\Gamma'}\} + b \cdot \max\{a_{l-1}^{\Gamma'} - 1, a_{l-1}^{\Gamma'}\} \\ &\quad - b \cdot \underbrace{\max\{a_l^{\Gamma'}, 1 + n_l^{\Gamma'}\}}_{= a_l^{\Gamma'} \text{ by Claim 6.20}} - b \cdot \max\{a_{l-1}^{\Gamma'}, 1 + a_{l-1}^{\Gamma'}\} \\ &= b(a_l^{\Gamma'} + k_i) + b \cdot a_{l-1}^{\Gamma'} - b(a_{l-1}^{\Gamma'} - 1) - b \cdot a_l^{\Gamma'} \\ &= b(k_i - 1) \geq 0. \end{aligned}$$

This proves the claim for the case where $l = l' > 1$.

Case 2. Now suppose that $l' = l - 1$. Observe that

$$\begin{aligned} m_\Gamma - m_{\Gamma'} &= \sum_{i=1}^{\ell(\mathcal{O})} b \cdot \max\{a_i^\Gamma, 1 + n_i^\Gamma\} - \sum_{i=1}^{\ell(\mathcal{O})} b \cdot \max\{a_i^{\Gamma'}, 1 + n_i^{\Gamma'}\} \\ &= b \cdot \max\{a_{l'}^\Gamma, 1 + n_{l'}^\Gamma\} - b \cdot \max\{a_{l'}^{\Gamma'}, 1 + n_{l'}^{\Gamma'}\} + \\ &\quad b \cdot \underbrace{\max\{a_l^\Gamma, 1 + n_l^\Gamma\}}_{= a_l^\Gamma \text{ by assumption}} - b \cdot \max\{a_l^{\Gamma'}, 1 + n_l^{\Gamma'}\} \\ &\geq b \cdot \max\{a_{l'}^\Gamma, 1 + n_{l'}^\Gamma\} - b \cdot \max\{a_{l'}^{\Gamma'}, 1 + n_{l'}^{\Gamma'}\}, \end{aligned}$$

where the last inequality holds since $a_l^{\Gamma'} \leq a_l^\Gamma$ by piecewise linearity and thus

$$\max\{a_l^{\Gamma'}, 1 + n_l^{\Gamma'}\} = \max\{a_l^{\Gamma'}, 1 + n_l^\Gamma\} \leq a_l^\Gamma .$$

By the construction of Γ', we know that $a_{l'}^\Gamma = n_{l'}^\Gamma$ and $a_{l'}^{\Gamma'} \leq a_{l'}^\Gamma + k_i \leq a_{l'}^\Gamma + b$. We are going to distinguish two subcases.

Subcase 2.1. Suppose first that $\max\{a_{l'}^{\Gamma'}, 1 + n_{l'}^{\Gamma'}\} = a_{l'}^{\Gamma'}$. Then,

$$m_\Gamma - m_{\Gamma'} = b(1 + n_{l'}^\Gamma) - b \cdot a_{l'}^{\Gamma'} \geq 0,$$

since $a_{l'}^{\Gamma'} \leq a_{l'}^\Gamma + b$ and $n_{l'}^\Gamma = a_{l'}^\Gamma$.

Subcase 2.2. Suppose now that $\max\{a_{l'}^{\Gamma'}, 1 + n_{l'}^{\Gamma'}\} = 1 + n_{l'}^{\Gamma'}$. Now $1 + n_{l'}^{\Gamma'} \geq a_{l'}^{\Gamma'}$ entails that the number of conflict-free facts in $\Gamma'[\mathcal{S}_{l'}]$ is at most 1. Since $l' = l - 1$, it must actually be the case that the number of conflict-free facts in $\Gamma'[\mathcal{S}_{l'}]$ is exactly one, i.e., $a_{l'}^{\Gamma'} - n_{l'}^{\Gamma'} = 1$. Therefore, we must have $n_{l'}^{\Gamma'} = n_{l'}^\Gamma$. Now we obtain

$$m_\Gamma - m_{\Gamma'} = b(1 + n_{l'}^\Gamma) - b(1 + n_{l'}^{\Gamma'}) = 0,$$

which proves that $m_\Gamma \geq m_{\Gamma'}$. □

By construction we know that $\mathsf{ldp}(\Gamma_k) \leq n$, whence by induction hypothesis it follows that there is a reduction sequence $\pi' = (\Gamma_{0,k}, \Gamma_{1,k}, \ldots, \Gamma_{k',k}, \Xi)$ for Γ_k such that $\Gamma_{0,k} = \Gamma_k$, $\mathrm{rk}(\Gamma_{k',k}) \leq m$, and $m_{\Gamma_{0,k}} \geq m_{\Gamma_{1,k}} \geq \cdots \geq m_{\Gamma_{k',k}}$. Now we construct the reduction sequence

$$\pi \coloneqq (\Gamma_0, \Gamma_1, \ldots, \Gamma_k, \Gamma_{1,k}, \ldots, \Gamma_{k',k}, \Xi),$$

and recall that $\Gamma_0 = \Gamma$. Now Claim 6.23 yields $m_\Gamma = m_{\Gamma_0} \geq m_{\Gamma_1} \geq \cdots \geq m_{\Gamma_k}$, whence it follows that

$$m_\Gamma = m_{\Gamma_0} \geq m_{\Gamma_1} \geq \cdots \geq m_{\Gamma_k} \geq m_{\Gamma_{1,k}} \geq \cdots \geq m_{\Gamma_{k',k}}.$$

Thus, π is a reduction sequence for Γ. This concludes the induction step of our subsidiary induction on $\mathsf{ldp}(\Gamma)$ and thus the proof of Lemma 6.18. □

We can now easily conclude the induction step for the induction on $\mathrm{rk}(\Gamma)$ as follows. We know by Lemma 6.18 that there is a reduction sequence $(\Gamma_0, \Gamma_1, \ldots, \Gamma_k, \Xi)$ for Γ such that

(i) $\Gamma_0 = \Gamma$,

(ii) $m_{\Gamma_0} \geq m_{\Gamma_1} \geq \cdots \geq m_{\Gamma_k}$,

(iii) each Γ_{i+1} is an unfolding of Γ_i, where $i = 0, \ldots, k-2$,

(iv) $\mathrm{rk}(\Gamma_k) \leq m$, and

(v) $\Xi = \Gamma_{k-1} \cap \mathfrak{D}$ and $\Gamma_k = \Gamma_{k-1} \setminus \Xi$.

Since $\mathrm{rk}(\Gamma_k) \leq m$, by induction hypothesis, there exists a linear chase tree $\mathcal{C}'$ for Γ_k whose node-width is bounded by m_{Γ_k}. Let $\mathcal{C}$ be the linear chase tree for Γ constructed as follows. The root v_0 of $\mathcal{C}$ is labeled Γ, and there are nodes $v_1, \ldots, v_k, v'_k$ such that (i) for all $i = 0, 1, \ldots, k-1$, v_i is labeled with Γ_i, (ii) for all $i = 0, 1, \ldots, k-2$, v_{i+1} is the only child of v_i, while (iii) v_{k-1} has two children, namely v_k and v'_k, where the former is labeled with Γ_k and the latter with Ξ. We declare that $\mathcal{C}'$ is a subtree of $\mathcal{C}$ that is rooted in v_k. Then $\mathcal{C}$ is a linear chase tree for Γ whose node-width is bounded by m_Γ, as required. This concludes the induction step and thus the proof of Lemma 6.17. □

Proof of Theorem 6.11. This proof is carried out *mutatis mutandis* as the proof of Theorem 6.12 by invoking Lemma 6.17. □

6.2.2 Algorithms for Query Answering

We now have all the tools for showing that answering CQs under piecewise linear warded sets of TGDs is in PSpace in combined complexity, and in NLogSpace in data complexity (Theorem 6.8), and also for re-establishing the complexity of warded sets of TGDs (Theorem 6.9) in a more transparent way than the approach of [84].

The case of $\mathsf{Eval}(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$ Given an OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ that falls into $(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$, an $\mathbf{S}$-database $\mathfrak{D}$, and a tuple $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$, by Theorem 6.11, our problem boils down to checking whether there exists a linear proof tree $\mathcal{P}$ of q w.r.t. $\mathcal{O}$ such that $\mathsf{nwd}(\mathcal{P}) \leq f_{\mathsf{W}\cap\mathsf{PWL}}(q, \mathcal{O})$ and $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$. This can be easily checked via a space-bounded algorithm that aims to build such a proof tree in a level-by-level fashion. Essentially, the algorithm builds the i-th level from the $(i-1)$-th level of the proof tree by non-deterministically applying the constituent operations for proof trees, i.e., resolution, decomposition, and specialization.

The algorithm is depicted in Algorithm 6.1. Here is a semi-formal description of it. The first step is to store in p the BCQ obtained after instantiating the output variables of q with $\bar{a}$. The rest of the algorithm is an iterative procedure that non-deterministically constructs p' (the i-th level) from p (the $(i-1)$-th level) until it reaches a level that can be matched to the database $\mathfrak{D}$. Notice that p and p' always hold one CQ, since at each level of a linear proof tree only, one node has a child, while all the other nodes are leaves, which essentially means that their atoms appear in the database $\mathfrak{D}$. At each

Algorithm 6.1: Algorithm for $\mathsf{Eval}(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$

Input: An OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$, an $\mathbf{S}$-database $\mathfrak{D}$, and $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$.

Output: ACCEPT if $\mathfrak{D} \models Q(\bar{a})$; otherwise, REJECT.

```
Let p be the CQ q(ā) ← α_1, ..., α_n, where {α_1, ..., α_n} = body(q(ā));
repeat
    guess op ∈ {r, d, s};
    if op = r then
        guess a TGD σ ∈ O;
        if mgcu(p, σ) = ∅ then
            REJECT;
        else
            guess U ∈ mgcu(p, σ);
            if |p[σ, U]| > f_{W∩PWL}(q, O) then
                REJECT;
            else
                p' := p[σ, U];
            end
        end
    end
    if op = d then
        let p' be the CQ obtained from p by deleting all body atoms of p that
         occur as facts in D;
    end
    if op = s then
        guess V ⊆ var(p) and a substitution γ: V → adom(D);
        p' := γ(p);
    end
    p := p';
until body(p) ⊆ D;
ACCEPT;
```

iteration, the algorithm constructs p' from p by applying resolution (r), decomposition (d), or specialization (s):

- *Resolution* It guesses a TGD $\sigma \in \mathcal{O}$. If the set $\mathsf{mgcu}(p, \sigma)$, i.e., the set of all MGCUs of p with σ, is empty, then it rejects; otherwise, it guesses $U \in \mathsf{mgcu}(p, \sigma)$. If the size of the σ-resolvent of p obtained via U, denoted $p[\sigma, U]$, does not exceed the bound given by Theorem 6.11, then it assigns $p[\sigma, U]$ to p'; otherwise, it rejects. Recall that, during a resolution step, we need to rename variables in order to avoid undesirable clutter. However, we cannot blindly use new variables at each step since this will explode the space used by algorithm. Instead, we should reuse variables that have

been lost due to their unification with an existentially quantified variable. A simple analysis shows that we only need polynomially many variables, while this polynomial depends only on q and $\mathcal{O}$.

- *Decomposition* It deletes from p the atoms that occur in $\mathfrak{D}$, and it assigns the obtained CQ to p'. Notice that this CQ may be empty in case $\mathsf{body}(p) \subseteq \mathfrak{D}$. Essentially, the algorithm decomposes p in such a way that the subquery of p having the body atoms $\mathsf{body}(p) \cap \mathfrak{D}$ forms a child of p that is a leaf, while the subquery having the body atoms $\mathsf{body}(p) \setminus \mathfrak{D}$ is the non-leaf child.
- *Specialization* It assigns to p' a specialized version of p, where some variables are instantiated by constants of $\mathsf{adom}(\mathfrak{D})$. Notice that the convention that answer variables correspond to constants is implemented by directly instantiating them with actual constants from $\mathsf{adom}(\mathfrak{D})$.

After constructing p', the algorithm assigns it to p, and this ends one iteration. If $\mathsf{body}(p) \subseteq \mathfrak{D}$, then a linear proof tree $\mathcal{P}$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$ has been found, and the algorithm accepts; otherwise, it proceeds with the next iteration.

It is easy to see that the algorithm uses polynomial space in general. Moreover, in case the set of TGDs and the CQ are fixed, the algorithm uses logarithmic space, which is the space needed for representing constantly many elements of $\mathsf{adom}(\mathfrak{D})$ – each element of $\mathsf{adom}(\mathfrak{D})$ can be represented using logarithmically many bits. Hence, the desired upper bounds claimed in Theorem 6.8 follow.

The case of Eval(W, CQ) The non-deterministic algorithm discussed above cannot be directly used for warded sets of TGDs, since it is not enough to search for a linear proof tree as in the case of piecewise linear warded sets of TGDs. However, by Theorem 6.12, we can search for a proof tree that has bounded node-width. This allows us to devise a space-bounded algorithm, which is similar in spirit as the one presented above, with the crucial difference that it constructs in a level-by-level fashion the branches of the proof tree in parallel universal computations using alternation. Since this alternating algorithm uses polynomial space in general, and logarithmic space when the set of TGDs and the CQ are fixed, we immediately get an ExpTime upper bound in combined, and a PTime upper bound in data complexity (recall that APSpace = ExpTime and ALogSpace = PTime). This confirms Theorem 6.9 established in [84]. However, our new algorithm is significantly simpler than the one employed in [84], and Theorem 6.12 reveals the main property of warded sets of TGDs that leads to the desirable complexity upper bounds.

Remark 6.24. We remark that it would be possible to solve $\mathsf{Eval}(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$ using word automata as well. In fact, linear proof trees can be easily encoded as words over a finite alphabet (see [55]), and one can devise an appropriate automaton that, given an input database, checks for the existence of a linear proof tree whose induced CQ is satisfied by the input database. Likewise, one can employ non-deterministic tree automata to solve $\mathsf{Eval}(\mathsf{W}, \mathsf{CQ})$. However, we find that the exposition using Algorithm 6.1 is simpler and more transparent.

6.2.3 A Justified Combination

We have studied the complexity of query answering under sets of TGDs that are both piecewise linear and warded. In fact, in Theorem 6.8, we established that query answering under such sets of TGDs is PSpace-complete in combined, and NLogSpace-complete in data complexity. It is interesting to observe that the class of piecewise linear warded sets of TGDs generalizes the class of *intensionally linear* sets of TGDs, denoted IL, where each TGD has at most one body atom whose predicate is intensional. Therefore, Theorem 6.8 will immediately imply that $\mathsf{Eval}(\mathsf{IL}, \mathsf{CQ})$ is PSpace-complete in combined, and NLogSpace-complete in data complexity. Notice that OMQs that have ontologies from IL generalize linear Datalog queries. Thus, we can extend linear Datalog by allowing existentially quantified variables in rule heads, which essentially leads to IL, without affecting the complexity of query answering.

At this point, one maybe tempted to think that the same holds for piecewise linear Datalog, i.e., that we can extend it with existentially quantified variables in rule heads, which leads to PWL, without affecting the complexity of query answering, that is, PSpace-complete in combined, and NLogSpace-complete in data complexity. However, if this is the case, then wardedness becomes redundant since the formalism that we are looking for is the class of piecewise linear sets of TGDs, without the wardedness condition. It turned out that this is not the case. To our surprise, the following holds:

Theorem 6.25. $\mathsf{Eval}(\mathsf{PWL}, \mathsf{CQ})$ *is undecidable in data complexity.*

Proof. We exploit an undecidable tiling problem [39]. A *tiling system* is a tuple $\mathbb{T} = (T, L, R, H, V, a, b)$, where T is a finite set of *tiles*, $L, R \subseteq T$ are special sets of *left* and *right border tiles*, respectively, with $L \cap R = \varnothing$, $H, V \subseteq T \times T$ are respectively the *horizontal* and *vertical* constraints, and $a, b \in T$ are distinguished tiles of $\mathbb{T}$ called the *start* and the *end* tile, respectively. A *tiling* for $\mathbb{T}$ is a function $f\colon [n] \times [m] \to T$, for some $n, m > 0$, such that $f(1,1) = a$, $f(1,m) = b$, $f(1,i) \in L$, $f(n,i) \in R$ for every $i \in [m]$, and f respects the horizontal and vertical constraints. In other words, the first and the last rows of a tiling for $\mathbb{T}$ respectively start with a and b, while the leftmost and rightmost columns respectively contain only tiles from L and R.

We reduce from the *unbounded tiling problem*, that is, given a tiling system $\mathbb{T} = (T, L, R, H, V, a, b)$, decide whether there is a tiling for $\mathbb{T}$. The goal is to construct in polynomial time a database $\mathfrak{D}_\mathbb{T}$, a set of TGDs $\mathcal{O} \in \mathsf{PWL}$, and a BCQ q such that, for the OMQ $Q = (\mathcal{O}, q)$ it holds that $\mathfrak{D}_\mathbb{T} \models Q$ iff $\mathbb{T}$ has a tiling. Note that $\mathcal{O}$ and q should not depend on $\mathbb{T}$.

The database $\mathfrak{D}_\mathbb{T}$. It simply stores the tiling system $\mathbb{T}$:

$$\mathfrak{D}_\mathbb{T} \coloneqq \{\mathsf{Tile}(t) \mid t \in T\} \cup \{\mathsf{Left}(t) \mid t \in L\} \cup \{\mathsf{Right}(t) \mid t \in R\} \cup \{H(t,t') \mid (t,t') \in H\} \cup \{V(t,t') \mid (t,t') \in V\} \cup \{\mathsf{Start}(a), \mathsf{End}(b)\}.$$

The set of TGDs $\mathcal{O}$. It is responsible for generating all the candidate tilings for $\mathbb{T}$ – i.e., tilings not necessarily satisfying the condition $f(1,m) = b$ – of arbitrary width and depth. Whether there exists a candidate tiling for $\mathbb{T}$ that satisfies the condition $f(1,m) = b$ will

be checked by the CQ q. The set $\mathcal{O}$ essentially implements the following idea: construct rows of size ℓ from rows of size $\ell - 1$, for $\ell > 1$, that respect the horizontal constraints, and then construct all the candidate tilings by combining compatible rows, i.e., rows that respect the vertical constraints. A row r is encoded as an atom $\mathsf{Row}(p, c, s, e)$, where p is the identifier of the row from which r has been obtained, i.e., the previous one, c is the identifier of r, i.e., the current one, s is the starting tile of r, and e is the end tile of r. We write $\mathsf{Row}(c, c, s, s)$ for rows consisting of a single tile which do not have a previous row (hence the identifier of the previous row coincides with the identifier of the current row), and the starting tile is the same as the end tile. The following two TGDs construct all the rows that respect the horizontal constraints:

$$\mathsf{Tile}(x) \rightarrow \exists z\, \mathsf{Row}(z, z, x, x),$$
$$\mathsf{Row}(\_, x, y, z), H(z, w) \rightarrow \exists u\, \mathsf{Row}(x, u, y, w).$$

Here we write '_' for a "don't-care" variable that occurs only once in the TGD. The next TGDs construct all the pairs of compatible rows, i.e., pairs of rows (r_1, r_2) such that we can place r_2 below r_1 without violating the vertical constraints. This is done again inductively as follows:

$$\mathsf{Row}(x, x, y, y), \mathsf{Row}(x', x', y', y'), V(y, y') \rightarrow \mathsf{Comp}(x, x'),$$
$$\mathsf{Row}(x, y, \_, z), \mathsf{Row}(x', y', \_, z'), \mathsf{Comp}(x, x'), V(z, z') \rightarrow \mathsf{Comp}(y, y').$$

We finally compute all the candidate tilings, together with their bottom-left tile, using the following two TGDs:

$$\mathsf{Row}(\_, x, y, z), \mathsf{Start}(y), \mathsf{Right}(z) \rightarrow \mathsf{CTiling}(x, y),$$
$$\mathsf{CTiling}(x, \_), \mathsf{Row}(\_, y, z, w), \mathsf{Comp}(x, y), \mathsf{Left}(z), \mathsf{Right}(w) \rightarrow \mathsf{CTiling}(y, z).$$

This concludes the definition of $\mathcal{O}$.

The BCQ q. Recall that q is responsible for checking whether there exists a candidate tiling such that its bottom-left tile is b. This can be easily done via the query

$$q \leftarrow \mathsf{CTiling}(x, y), \mathsf{End}(y).$$

By construction, the set $\mathcal{O}$ of TGDs belongs to PWL. Moreover, there is a tiling for $\mathbb{T}$ iff $\mathfrak{D}_{\mathbb{T}} \models Q$. □

6.3 Expressive Power

In this section, we are going to provide a sufficient criterion for OMQs from $(\mathsf{TGD}, \mathsf{CQ})$ to be expressible as Datalog queries. Moreover, we are going to show that every OMQ from $(\mathsf{W}, \mathsf{CQ})$ is expressible as a Datalog query, and that every OMQ from $(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$ is even expressible as a piecewise linear Datalog query.

To this end, we exploit the existence of (linear) proof trees of bounded node-width for OMQs based on (piecewise linear) warded sets of TGDs (Theorem 6.12 and Theorem 6.11).

In fact, it turns out that uniform bounds on the node-width of proof trees provides a criterion for Datalog rewritability for any OMQ from $(\mathsf{TGD}, \mathsf{CQ})$. This is the content of the following statement, which is the main result of this section:

THEOREM 6.26. *An OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $(\mathsf{TGD}, \mathsf{CQ})$ is equivalent to a (linear) Datalog query, if there exists a constant k_Q such that, for every $\mathbf{S}$-database $\mathfrak{D}$ and all tuples $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$, it holds that $\mathfrak{D} \models Q(\bar{a})$ implies that there is a (linear) proof tree $\mathcal{P}$ of $q(\bar{x})$ w.r.t. $\mathcal{O}$ such that*

(i) $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$, *and*

(ii) $\mathsf{nwd}(\mathcal{P}) \leq k_Q$.

PROOF. We carry out the proof for the case of arbitrary Datalog queries and explain the case of linear Datalog queries later. Let $\mathcal{S}$ be the set of all CQs over $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ whose number of body atoms is bounded by k_Q. Consider $\mathcal{S}$ to be factorized modulo query equivalence, i.e., if (the queries defined by) two CQs p, p' over $\mathbf{S} \cup \mathsf{sig}(\mathcal{O})$ with at most k_Q atoms are equivalent, then p and p' are represented in $\mathcal{S}$ by the equivalence class $[p] = [p']$. Notice that all the CQs of an equivalence class have the same number of answer variables by definition, and notice also that $\mathcal{S}$ is clearly finite.

For each equivalence class $\alpha = [p(v_1, \ldots, v_n)] \in \mathcal{S}$, let C_α be a fresh n-ary predicate. A *successor rule* of $\alpha \in \mathcal{S}$ is a Datalog rule of the form

$$C_{[p_1]}(\bar{v}_1), \ldots, C_{[p_k]}(\bar{v}_k) \rightarrow C_\alpha(\bar{v}), \tag{6.1}$$

such that the following holds: there are CQs $q_i(\bar{v}_i) \in [p_i]$ $(i = 1, \ldots, k)$ and a $q'(\bar{v}) \in \alpha$ such that there is a proof tree $\mathcal{P}$ of $q'(\bar{v})$ w.r.t. $\mathcal{O}$ that has $k > 0$ children whose respective labels are $q_1(\bar{v}_1), \ldots, q_k(\bar{v}_k)$. We may assume w.l.o.g. that the number of atoms in the rule bodies of successor rules is bounded, since the number of body atoms of each CQ in α is bounded, and the branching degree of a proof tree can only be arbitrarily inflated by a decomposition step that introduces repetitive CQs.

Notice, however, that α may have infinitely many successor rules, since one can choose different variables for these rules. We denote by $\mathcal{R}(\alpha)$ the set of all successor rules of α factorized modulo logical equivalence, i.e., $\mathcal{R}(\alpha)$ consists of equivalence classes containing logically equivalent successor rules of α. It is clear that $\mathcal{R}(\alpha)$ is finite, since there are only finitely many predicates of the form C_β, and since we assume that the number of atoms in rule bodies of successor rules is bounded.

We inductively construct sets of Datalog rules $\Pi_0, \Pi_1, \ldots$ as follows:

(i) We let Π_0 be the set of rules containing precisely one representative of each rule in $\mathcal{R}([q(\bar{x})])$.

(ii) Π_{n+1} consists of all rules of Π_n plus the following rules: for all rules

$$C_{[p_1]}(\bar{v}_1), \ldots, C_{[p_k]}(\bar{v}_k) \rightarrow C_\alpha(\bar{v}),$$

of Π_n, we add, for all $i = 1, \ldots, k$, precisely one representative τ of each rule in $\mathcal{R}([p_i])$, provided τ is not equivalent to any rule contained in Π_n.

A little thought shows that, up to logical equivalence, there are only finitely many rules of the form (6.1) at all. Hence, it must be the case that $\Pi_l = \Pi_m$ for all $l \geq m$. Thus, the set $\Pi' \coloneqq \bigcup_{i \geq 0} \Pi_i$ is finite. Now we add to Π' all rules of the form

$$R(v_1, \ldots, v_k) \to C_{[p_R]}(v_1, \ldots, v_k), \quad R/k \in \mathbf{S}, \tag{6.2}$$

where p_R is the atomic query $p_R(v_1, \ldots, v_k) \leftarrow R(v_1, \ldots, v_k)$. These rules ensure that the resulting Datalog query will have $\mathbf{S}$ as an extensional schema. Call the resulting set of rules Π, and let $Q' \coloneqq (\Pi, C_{[q(\bar{x})]}(\bar{x}))$ be the resulting Datalog query. It is not hard to show that Q' is equivalent to Q, i.e., $Q'(\mathfrak{D}) = Q(\mathfrak{D})$ for all $\mathbf{S}$-databases $\mathfrak{D}$. Let us only give some hints for the proof of this fact.

If $\mathfrak{D} \models Q'(\bar{a})$, then, for each predicate $C_{[p(\bar{v})]}(\bar{v})$, one can show by induction on n that $T^n_{Q'}(\mathfrak{D}) \models C_{[p(\bar{v})]}(\bar{b})$ entails that $(\mathfrak{D}, \mathcal{O}) \models p(\bar{b})$. Thus, $(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$, that is, $\mathfrak{D} \models Q(\bar{a})$.

On the other hand, if $\mathfrak{D} \models Q(\bar{a})$, then there is a proof tree $\mathcal{P}$ of $q(\bar{x})$ w.r.t. $\mathcal{O}$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$. An induction on the height of $\mathcal{P}$ shows that if a node is labeled by a CQ $p(\bar{v})$, then $(\mathfrak{D}, \mathcal{O}) \models p(\bar{b})$ implies that $(\mathfrak{D}, \Pi) \models C_{[p(\bar{v})]}(\bar{b})$. Since $(\mathfrak{D}, \mathcal{O}) \models q(\bar{a})$ by assumption, one obtains $(\mathfrak{D}, \Pi) \models C_{[q(\bar{x})]}(\bar{a})$ and thus $\mathfrak{D} \models Q'(\bar{a})$ as desired.

Notice that if we use linear proof trees in the construction of the rules of the form (6.1), then the set Π' constructed above is linear. Instead of adding to Π' rules of the form (6.2), we construct Π as follows. For each atomic query $p_R(v_1, \ldots, v_k) \leftarrow R(v_1, \ldots, v_k)$ (with $R \in \mathbf{S}$), we replace occurrences of $C_{[p_R]}(\bar{v})$ in rule bodies of Π' simply by $R(\bar{v})$. The Datalog query $Q' \coloneqq (\Pi, C_{[q(\bar{x})]}(\bar{x}))$ is then easily seen to be linear. This concludes the proof of Theorem 6.26. □

Using Theorem 6.26, we can almost immediately conclude the following:

Corollary 6.27. *Every OMQ from* $(\mathsf{W}, \mathsf{CQ})$ *is equivalent to a Datalog query. Moreover, every OMQ from* $(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$ *is equivalent to a linear Datalog query.*

Proof. Consider a $Q = (\mathbf{S}, \mathcal{O}, q)$ from $(\mathsf{W}, \mathsf{CQ})$. Let $k_Q \coloneqq f_{\mathsf{W}}(\mathcal{O}, q)$. The result is now immediate by Theorems 6.12 and 6.26. The result for the case of $Q \in (\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$ is shown accordingly by letting $k_Q \coloneqq f_{\mathsf{W} \cap \mathsf{PWL}}(\mathcal{O}, q)$ and invoking Theorem 6.11. □

Notice in particular that Corollary 6.27 states that OMQs based on piecewise linear warded sets of TGDs are not only equivalent to piecewise linear Datalog queries, but even to linear ones. This is not surprising, as it is known that every piecewise linear Datalog query can be rewritten into an equivalent linear one [2].

6.4 Summary

We introduced proof trees for rule-based OMQs and applied them to obtain several interesting results. We showed that they provide a useful tool for query answering algorithms and for establishing Datalog rewritability. Moreover, we contributed to the theory of Vadalog by identifying a space-efficient fragment of warded sets of rules. This fragment is based on piecewise linear recursion, and we established that the evaluation

problem for OMQs based on piecewise linear warded sets of TGDs is PSpace-complete in combined, and NLogSpace-complete in data complexity. The lower bounds were inherited from (piecewise) linear Datalog, while the upper bounds were proved by using our novel notion of proof tree. In this context, we also re-established the complexity of answering OMQs from $(\mathsf{W}, \mathsf{CQ})$. Finally, we provided a strong sufficient criterion based on proof trees for arbitrary rule-based OMQs to be Datalog rewritable, and we used this result to show that OMQs from $(\mathsf{W}, \mathsf{CQ})$ are rewritable into Datalog queries, while those from $(\mathsf{W} \cap \mathsf{PWL}, \mathsf{CQ})$ are even rewritable into (piecewise) linear Datalog queries. We mention that the optimal size of these rewritings is open, but we conjecture that these rewritings can be improved to be polynomial by developing more refined rewriting algorithms that make better use of wardedness. It is also an open question whether this criterion is also necessary, i.e., whether this criterion *characterizes* Datalog rewritability for all OMQs from $(\mathsf{TGD}, \mathsf{CQ})$.

It would be interesting to devise query answering algorithms based on proof trees for other known decidable classes of TGDs. In fact, we view proof trees as a generic tool for studying the complexity of classes of TGDs, where one can obtain space-bounded query answering algorithms once the bounds on the node-width are settled. For languages that are Datalog rewritable, their Datalog rewritability can be elegantly established by providing appropriate uniform bounds on the node-width, as demanded by Theorem 6.26. For example, we believe that it should not be too hard to establish that OMQs based on guarded sets of TGDs always permit proof trees whose node-width is exponentially bounded in the maximum arity of the OMQ in question. This would yield an AExpSpace = 2ExpTime algorithm for answering guarded OMQs, which is optimal. Furthermore, the obtained bound should be independent of the actual input data, whence Datalog rewritability follows.

Concerning more applied research directions, we mention that the Vadalog system is currently optimized for piecewise linear warded sets of TGDs in three ways [27]: (i) the first one is related to the way that existential quantifiers interact with recursion, (ii) the second one is related to the optimizer, which detects and uses piecewise linearity for the purpose of join ordering, and (iii) the third way is related to the architecture of the system. Here are some promising directions for future research:

- As NLogSpace is contained in the class NC_2 of highly parallelizable problems, this means that reasoning under piecewise linear warded sets of TGDs is principally parallelizable, while reasoning under warded sets of TGDs is presumably not. We plan to exploit this for the parallel execution of reasoning tasks.
- Reasoning with piecewise linear warded sets of TGDs is LogSpace-equivalent to reachability in directed graphs. Reachability in very large graphs has been well-studied and many algorithms and heuristics have been designed that work well in practice (see, e.g., [56, 95, 98]). We are confident that several of these algorithms can be adapted for our purposes.
- Reachability in directed graphs is known to be in the *dynamic* parallel complexity class DynFO [68, 116]. We plan to analyze whether reasoning under piecewise linear warded sets of TGDs, or relevant subclasses thereof, can be shown to be in DynFO.

CHAPTER 7

Conclusion

In this thesis, we studied static analysis tasks for ontology-mediated queries (OMQs) which are based on TGDs. Static analysis tasks are data-independent and aim at deciding semantic properties of OMQs, which may be useful for query optimization. In our study, we focused on problems for the arguably most popular classes of TGDs, namely linear, (frontier-)guarded, sticky, and non-recursive sets of TGDs.

In Chapter 4 we studied the containment problem for OMQs based on well-established decidable classes of TGDs. Since containment is undecidable for languages that capture Datalog queries, we focused on linear, sticky, non-recursive, and (frontier-)guarded sets of TGDs. For these classes of TGDs, we devised specifically tailored algorithms for solving the containment problem. For UCQ-rewritable languages (i.e., linear, sticky, and non-recursive sets of TGDs), we relied on the existence of UCQ-rewritings to decide containment, while for guarded-based classes, we exploited a tree-like witness property of these queries and resorted to automata techniques for deciding containment. Moreover, we also considered case of containment problems where the left-hand side and right-hand side query fall into different languages. The algorithms devised turn out to be worst-case optimal in all cases, except for the case of non-recursive TGDs, where we established $\text{PTime}^{\text{NExpTime}}$-hardness and membership in $\text{coNExpTime}^{\text{NP}}$. Establishing a tight complexity bound for this case is left for future research.

We continued in Chapter 5 to study the problem of first-order rewritability for OMQs based on (frontier-)guarded sets of TGDs. It was shown that the setting of having relations of arity greater than two renders the problem more challenging. We provided a semantic characterization for first-order rewritability that exploited the tree-like witness property of (frontier-)guarded OMQs. Then, we first provided a non-optimal solution to first-order rewritability based on classical automata techniques, and afterward we resorted to cost automata in order to obtain worst-case optimal algorithms. It turned out that deciding first-order rewritability is 2ExpTime-complete for frontier-guarded TGDs, even if we assume schemas of bounded width. An open question concerning first-order rewritable (frontier-)guarded OMQs is the worst-case size of their rewritings:

we conjecture that their worst-case size is a tower of exponentials of height four and leave this question for future research.

At this point we would like to stress again that containment and first-order rewritability become undecidable once one focuses on plain Datalog queries. One can attribute the decidability of containment and first-order rewritability for the case of (frontier-)guarded OMQs to the existence of tree-like witnesses. Indeed, tree-like witnesses allowed us to resort to tree automata, which served as a generic tool to answer the universal claims posed by the problems of containment and first-order rewritability. Likewise, for first-order rewritable OMQs, the decidability of containment can be attributed to the existence of UCQ-rewritings, since for containment of UCQs is clearly decidable. A natural research question is therefore to identify the limits of the containment and first-order rewritability problems for all classes of OMQs. More specifically, let us consider the case of containment. We know that all (frontier-)guarded OMQs can be naturally expressed as (frontier-)guarded Datalog queries. Hence, containment of such OMQs reduces to the containment problem for special classes of Datalog queries. Moreover, it is known that deciding whether a Datalog query is contained in a (frontier-)guarded Datalog query is decidable [40]. A natural question is therefore to provide a non-trivial characterization of those classes $\mathcal{C}$ of Datalog queries for which $\mathsf{Cont}(\textsc{Datalog}, \mathcal{C})$ is decidable, where Datalog denotes the class of all Datalog queries. Likewise, it would be interesting to characterize those classes $\mathcal{C}$ for which $\mathsf{Cont}(\mathcal{C}, \textsc{Datalog})$ is decidable. We leave these open questions to future research.

Although we were cautious in Chapters 4 and 5 to exclude OMQ languages capturing all Datalog queries from our study, we revisited the study of such languages in Chapter 6. More specifically, we analyzed proof trees built using Datalog rules, and we extended them to the case of OMQs based on existential rules. While proof trees for Datalog queries are trees labeled by atomic facts, proof trees for OMQs are trees labeled by CQs. We observed that the size of the largest CQ present in such proof trees (i.e., their node-width) provides guidance for the space consumption of an alternating query answering algorithm. This observation allowed us to use proof trees to specify elegant query answering algorithms – we provided such algorithms for OMQs based on warded sets of rules, and for those based on piecewise linear warded sets of rules. Warded sets of TGDs constitute the main language of Vadalog – a reasoning system developed at the University of Oxford –, while piecewise linear warded sets of TGDs are a space-efficient fragment thereof. We showed that the data complexity of answering OMQs based on piecewise linear warded sets of TGDs is NLogSpace-complete in data complexity, and PSpace-complete in combined complexity. As NLogSpace is contained in the class NC_2 of parallelizable problems, there is hope to exploit these insights when it comes to the development of parallel reasoning algorithms.

Apart from the study of Vadalog, we believe that the notion of proof tree developed in this thesis is useful for query answering algorithms for other decidable classes of TGDs as well. In fact, it would be interesting to devise query answering algorithms based on proof trees for existing classes of TGDs, thereby establishing a unifying framework for query answering algorithms.

The nice structural properties of proof trees allowed us to answer another open question concerning OMQs based on warded sets of rules, namely that they can be equivalently expressed as Datalog queries. Likewise, we showed that OMQs based on piecewise linear sets of rules are always expressible as (piecewise) linear Datalog queries. In fact, these results were based on a more general insight: we provided a strong sufficient criterion for any OMQ based on TGDs to be expressible as a Datalog query – this is the case whenever one can always find proof trees of bounded node-width, uniformly across all input databases. A natural question that arises is whether this criterion is also necessary, i.e., whether it *characterizes* expressibility of OMQs based on TGDs in the language of Datalog. Again, we leave this highly interesting question to future research.

APPENDIX A

The Procedure XRewrite

In view of the fact that the rewriting algorithm XRewrite is heavily used in our complexity analysis in Chapter 4, we would like to recall its definition. We only introduce the basic concepts here that are needed for the proofs of some of the results from Section 4.2 – for a more thorough presentation, we refer the interested reader to [82]. This algorithm is based on resolution, and thus, before proceeding further, we need to recall the notion of unification (cf. also Chapter 6). A set of atoms $A = \{\alpha_1, \ldots, \alpha_n\}$, where $n \geq 2$, *unifies* if there exists a substitution γ, called *unifier* for A, such that $\gamma(\alpha_1) = \cdots = \gamma(\alpha_n)$. A *most general unifier (MGU)* for A is a unifier for A, such that for each other unifier γ for A, there exists a substitution γ' such that $\gamma = \gamma' \circ \gamma_A$. Notice that if a set of atoms unify, then there exists an MGU. Furthermore, the MGU for a set of atoms is unique (modulo variable renaming), and we denote by γ_A an MGU for A that is fixed.

The algorithm proceeds by exhaustively applying two steps: *rewriting* and *factorization*, which in turn rely on the technical notions of *applicability* and *factorizability*, respectively. We assume, w.l.o.g., that TGDs and CQs do not share variables. Given a CQ q, a variable x is called *shared* in q if x is a free variable of q, or it occurs more than once in q. In what follows, we assume, w.l.o.g., that TGDs are in a convenient normal form: they have only one atom in the head, and only one occurrence of an existentially quantified variable [47]. Moreover, we can assume that neither q nor the body of any rule contains equality atoms (except for rules with body $\top$), as these can be eliminated by identifying variables. We write $\pi_\exists(\tau)$ for the position[1] at which the existentially quantified variable of τ occurs. In case τ does not mention any existentially quantified variables, then $\pi_\exists(\tau) \coloneqq \varepsilon$.

We are now ready to recall applicability and factorizability. Consider a CQ q and a TGD τ. Given a set of atoms $S \subseteq \mathsf{body}(q)$, we say that τ is *applicable* to S if the following conditions are satisfied:

(i) the set $S \cup \{\mathsf{head}(\tau)\}$ unifies, and

[1]Recall that a position $R[i]$ identifies the i-th attribute of a predicate R.

Algorithm A.1: The algorithm XRewrite

Input: An OMQ $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ from $(\mathsf{TGD}, \mathsf{CQ})$
Output: A UCQ $q'(\bar{x})$ such that $Q(\mathfrak{D}) = q'(\mathfrak{D})$, for every $\mathbf{S}$-database $\mathfrak{D}$

$i := 0$;
$Q_{\text{REW}} := \{\langle q, \mathsf{r}, \mathsf{u}\rangle\}$;
repeat
  $Q_{\text{TEMP}} := Q_{\text{REW}}$;
  foreach $\langle q, x, \mathsf{u}\rangle \in Q_{\text{TEMP}}$, where $x \in \{\mathsf{r}, \mathsf{f}\}$ **do**
    foreach $\sigma \in \mathcal{O}$ **do**
      /* rewriting step */
      foreach $S \subseteq \mathsf{body}(q)$ such that σ is applicable to S **do**
        $i := i + 1$;
        $q' := \gamma_{S,\sigma_i}(q[S/\mathsf{body}(\sigma_i)])$;
        if there is no $(q'', \mathsf{r}, *) \in Q_{\text{REW}}$ such that $q' \simeq q''$ **then**
          $Q_{\text{REW}} := Q_{\text{REW}} \cup \{\langle q', \mathsf{r}, \mathsf{u}\rangle\}$;
        end
      end
      /* factorization step */
      foreach $S \subseteq \mathsf{body}(q)$ that is factorizable w.r.t. σ **do**
        $q' := \gamma_S(q)$;
        if there is no $(q'', *, *) \in Q_{\text{REW}}$ such that $q' \simeq q''$ **then**
          $Q_{\text{REW}} := Q_{\text{REW}} \cup \{\langle q', \mathsf{f}, \mathsf{u}\rangle\}$;
        end
      end
    end
    /* the query q is now explored */
    $Q_{\text{REW}} := (Q_{\text{REW}} \setminus \{(q, x, \mathsf{u})\}) \cup \{(q, x, \mathsf{e})\}$;
  end
until $Q_{\text{TEMP}} = Q_{\text{REW}}$;
$Q_{\text{FIN}} := \{q \mid \langle q, \mathsf{r}, \mathsf{e}\rangle \in Q_{\text{REW}},$ and q contains only predicates of $\mathbf{S}\}$;
return Q_{FIN};

(ii) for each $\alpha \in S$, if the term at position π in α is either a constant or a shared variable in q, then $\pi \neq \pi_\exists(\tau)$.

Roughly, whenever τ is applicable to S, this means that the atoms of S may be generated during the chase procedure by applying τ. Therefore, we are allowed to apply a rewriting step (which is essentially a resolution step) that resolves S using τ, i.e., S is replaced by $\mathsf{body}(\tau)$, and a new CQ that is "closer" to the input database is obtained.

If we start applying rewriting steps blindly, without checking for applicability, then the soundness of the rewriting procedure is not guaranteed. However, it is possible that the applicability condition is not satisfied, but we should still apply a rewriting step. This may happen due to the presence of redundant atoms in a query. For example, given

the CQ

$$q \coloneqq \exists x, y, z\,(R(x, y) \wedge R(x, z)),$$

and the TGD

$$\tau\colon P(u, v) \rightarrow \exists w\, R(w, u),$$

the applicability condition fails since the shared variable x in q occurs at the position $\pi_{\exists}(\tau) = R[1]$. However, q is essentially the CQ $q = \exists x, y\, R(x, y)$, and now the applicability condition is satisfied. From the above informal discussion, we conclude that the applicability condition may prevent the algorithm from being complete, since some valid rewriting steps are blocked. Because of this reason, we need the so-called factorization step, which aims at converting some shared variables into non-shared variables, and thus, satisfy the applicability condition. In general, this can be achieved by exhaustively unifying all the atoms that unify in the body of a CQ. However, some of these unifications do not contribute in any way to satisfying the applicability condition, and, as a result, many superfluous CQs are generated. It is thus better to apply a restricted form of factorization that generates a possibly small number of CQs which are vital for the completeness of the rewriting algorithm. This corresponds to the identification of all the atoms in the query whose shared existential variables come from the same atom in the chase, and they can be unified with no loss of information.

Consider a CQ q and a TGD τ. Given a set of atoms $S \subseteq \mathsf{body}(q)$, where $|S| \geq 2$, we say that S is *factorizable* w.r.t. τ if the following conditions are satisfied:

(i) S unifies,

(ii) $\pi_{\exists}(\tau) \neq \varepsilon$, and

(iii) there exists a variable $x \notin \mathsf{var}(\mathsf{body}(q) \setminus S)$ that occurs in every atom of S only at position $\pi_{\exists}(\tau)$.

The algorithm XRewrite Having the above key notions in place, we are now ready to recall the algorithm XRewrite, which is depicted in Algorithm A.1. As said above, the UCQ-rewriting of an OMQ $Q = (\mathbf{S}, \mathcal{O}, q)$ is computed by exhaustively applying (i.e., until a fixpoint is reached) the rewriting and the factorization steps. Notice that the CQs that are the result of the factorization step, are nothing else than auxiliary queries which are critical for the completeness of the final rewriting, but are not needed in the final rewriting. Thus, during the iterative procedure, the queries are labeled with r (resp., f) in order to keep track which of them are generated by the rewriting (resp., factorization) step. The CQ that is part of the input OMQ, although is not a result of the rewriting step, is labeled by r since it must be part of the final rewriting. Moreover, once the two crucial steps have been exhaustively applied to a CQ q, it is not necessary to revisit q, since this will lead to redundant queries. Hence, the queries are also labeled with e (respectively, u) indicating that a query has been already explored (respectively, is unexplored). Let us now describe the two main steps of the algorithm. In the sequel,

consider a triple (q, x, y), where $(x, y) \in \{\mathsf{r}, \mathsf{f}\} \times \{\mathsf{e}, \mathsf{u}\}$ (this is how we indicate that q is labeled by x and y), and a TGD $\sigma \in \mathcal{O}$.

Rewriting Step For each $S \subseteq \mathsf{body}(q)$ such that σ is applicable to S, the i-th application of the rewriting step generates the query $q' = \gamma_{S,\sigma_i}(q[S/\mathsf{body}(\sigma_i)])$, where σ_i is the TGD obtained from σ by replacing each variable x with x_i, γ_{S,σ_i} is the MGU for the set $S \cup \{\mathsf{head}(\sigma_i)\}$ (which is the identity on the variables that appear in the body but not in the head of σ_i), and $q[S/\mathsf{body}(\sigma_i)]$ is obtained from q be replacing S with $\mathsf{body}(\sigma_i)$. By considering σ_i (instead of σ) we basically rename, using the integer i, the variables of σ. This renaming step is needed in order to avoid undesirable clutter among the variables introduced during different applications of the rewriting step. Finally, if there is no $(q'', \mathsf{r}, *) \in Q_{\text{REW}}$, i.e., an explored or unexplored query that is the result of the rewriting step, such that q' and q'' are the same (modulo variable renamings), denoted $q' \simeq q''$, then $(q', \mathsf{r}, \mathsf{u})$ is added to Q_{REW}.

Factorization Step For each $S \subseteq \mathsf{body}(q)$ that is factorizable w.r.t. σ, the factorization step generates the query $q' = \gamma_S(q)$, where γ_S is the MGU for S. If there is no $(q'', *, *) \in Q_{\text{REW}}$, i.e., a query that is the result of the rewriting or the factorization step, and is explored or unexplored, such that $q' \simeq q''$, then $(q', \mathsf{f}, \mathsf{u})$ is added to Q_{REW}.

APPENDIX B

Missing Proofs

B.1 Proofs for Chapter 2

B.1.1 Proof of Theorem 2.28

In the following we are going to provide a sketch of a proof for Theorem 2.28. The construction we are going to provide closely follows the well-known translation by Vardi who shows in [133] how to convert two-way alternating parity automata working on infinite trees, where every node has the same fixed branching degree, to one-way non-deterministic parity tree automata. The construction of Vardi also incurs an exponential blowup in the number of states of the constructed automaton. In contrast to Vardi's construction, we are going to work exclusively on finite trees, and this is why we translate to ordinary 1NTAs (cf. Definition 2.19) instead of transitioning to non-deterministic parity tree automata. However, the construction provided here is pretty close to that in [133], and therefore we cite results from [133] when their proofs are *mutatis mutandis* to according results in [133].

Throughout the proof, let us fix an m-2APTA

$$\mathcal{A} = (S, \Gamma, \mathsf{Dir} \coloneqq \{-1, 0, 1, \ldots, m\}, \delta, \Omega, s_0).$$

Before proving Theorem 2.28 we need several additional notions first.

Strategy trees Consider a Γ-labeled input tree t. A *candidate strategy for t w.r.t. $\mathcal{A}$* is a class of functions $\tau = (\tau_x)_{x \in \mathsf{dom}(t)}$, where each function τ_x is of type $S \to 2^{\mathsf{Dir} \times S}$. Notice that the graphs of the functions τ_x can be conceived to be subsets of $S \times \mathsf{Dir} \times S$ and thus each τ_x can be captured by a finite alphabet. Intuitively, an element $(d, s') \in \tau_x(s)$ tells us that, once we reach node x in state s, we should send a copy of $\mathcal{A}$ in state s' to $x \cdot d$. We call the pair (t, τ) a *strategy tree for $\mathcal{A}$ on t* if it satisfies the following conditions:

(i) $\tau_\varepsilon(s_0) \neq \varnothing$.

(ii) For all $x \in \mathsf{dom}(t)$ and all $s \in S$, the assignment $\theta_{s,x}$, defined as follows, satisfies $\delta(s, t(x))$:
- for all $d \in \mathsf{Dir}$, $\theta_{s,x}(\langle d \rangle p) = 1$ iff $(d, p) \in \tau_x(s)$;
- for all $d \in \mathsf{Dir}$, $\theta_{s,x}([d]p) = 1$ iff $d_t(x) = \varnothing$ or $(d, p) \in \tau_x(s)$.

A *strategy path in* (t, τ) is a sequence $\alpha = (x_1, s_1), (x_2, s_2), \ldots$ of pairs from $\mathsf{dom}(t) \times S$ such that, for all $k \geq 0$, there is some d such that $(s_{k+1}, d) \in \tau_{x_k}(s_k)$ and $x_{k+1} = x_k \cdot d$. We say that α is *accepting* if the minimal number n such that $\Omega(s_i) = n$, for infinitely many i, is even. We say that (t, τ) is *accepting* if all infinite strategy paths in (t, τ) that start in (ε, s_0) are accepting.

A proof of a (slightly different) variant of the following lemma can be found in [133]:

Lemma B.1. *$\mathcal{A}$ accepts t iff there is an accepting strategy tree for $\mathcal{A}$ on t.*

Priority annotations Let $M \coloneqq \{i \mid \exists s \in S \colon \Omega(s) = i\}$, i.e., M is the set of priorities that are assigned to states of $\mathcal{A}$. Let t be a Γ-labeled tree. A *priority annotation for t* is a class $\eta = (\eta_x)_{x \in \mathsf{dom}(t)}$ of functions which are all of type $S \to 2^{2^M \times S}$, that is, each η_x assigns to every state $s \in S$ a set of pairs of the form (H, s'), where $H \subseteq M$ and $s' \in S$. Intuitively, a pair $(H, s') \in \eta_x(s)$ states that, when a copy of $\mathcal{A}$ is launched at node x in state s, then it is possible that another copy of $\mathcal{A}$ returns to x in state s' and, on its way "back" to x, it encountered states with priorities among H. Thus, η_x provides information about the possible upward and stationary movements of $\mathcal{A}$ from node x.

Given a strategy tree (t, τ) for $\mathcal{A}$ on t, we say that η *annotates* (t, τ) if the following properties are satisfied for all $x \in \mathsf{dom}(t)$:

(i) If $(H_1, s') \in \eta_x(s)$ and $(H_2, s'') \in \eta_x(s')$, then $(H_1 \cup H_2, s'') \in \eta_x(s)$.

(ii) If $(0, s') \in \tau_x(s)$ then $(\{\Omega(s')\}, s') \in \eta_x(s)$.

(iii) If $x = y \cdot i$ for some $i = 1, \ldots, m$, $(-1, s') \in \tau_x(s)$, $(H, s'') \in \eta_y(s')$, and $(i, s''') \in \tau_y(s'')$, then $(H \cup \{\Omega(s'), \Omega(s''')\}, s''') \in \eta_x(s)$.

(iv) If $y = x \cdot i$ for some $i = 1, \ldots, m$, $(i, s') \in \tau_x(s)$, $(H, s'') \in \eta_y(s')$, and $(-1, s''') \in \tau_y(s'')$, then $(H \cup \{\Omega(s'), \Omega(s''')\}, s''') \in \eta_x(s)$.

We call the triple (t, τ, η) an *annotated strategy tree for $\mathcal{A}$ on t*.

A *downward path* α in (t, τ, η) is a sequence $(x_0 = \varepsilon, s_0, \beta_1), (x_2, s_2, \beta_2), \ldots$ of triples, where each x_i is from $\mathsf{dom}(t)$, each s_i is from S, and for each β_i either of the following holds:

- $\beta_i \in \tau_{x_i}(s_i)$ and $\beta_i = (d, s_{i+1})$ with $d \in \{1, \ldots, m\}$ and $x_{i+1} = x_i \cdot d$. In this case, we define $\Omega(\beta_i) \coloneqq \Omega(s_{i+1})$.
- $\beta_i \in \eta_{x_i}(s_i)$ and $\beta_i = (H, s_{i+1})$ with $H \subseteq M$ and $x_{i+1} = x_i$. In this case, we define $\Omega(\beta_i) \coloneqq \min H$.

If α is infinite, then we let $\Omega(\alpha)$ be the minimal n such that $\Omega(\beta_i)$ for infinitely many i. If α is finite, i.e., of the form $(x_0 = \varepsilon, s_0, \beta_1), \ldots, (x_\ell, s_\ell, \beta_\ell)$ with $\beta_\ell = (H, s_\ell) \in \eta_{x_\ell}(s_\ell)$,

then we let $\Omega(\alpha) \coloneqq \Omega(s_\ell)$. We say that α *violates the parity condition* if $\Omega(\alpha)$ is odd. We say that (t, τ, η) is *accepting*, if no downward path in (t, τ, η) violates the parity condition.

Using Lemma B.1, the following lemma can be shown (see again [133] for an according result):

LEMMA B.2. *$\mathcal{A}$ accepts t iff there is an accepting annotated strategy tree for $\mathcal{A}$ on t.*

Fully annotated strategy trees Fix a strategy tree (t, τ) for $\mathcal{A}$ on t. For $x \in \mathsf{dom}(t)$ and $d \in \{-1, 0, 1, \dots, m\}$, let

$$\zeta_{x \cdot d} \coloneqq \begin{cases} \tau_{x \cdot d}, & \text{if } d_t(x) \neq \varnothing, \\ \sharp, & \text{otherwise.} \end{cases}$$

Thus, $\zeta_{x \cdot d}$ is the function $\tau_{x \cdot d}$ in case $x \cdot d$ is a node of t (i.e., $d_t(x) \neq \varnothing$), and the special symbol $\sharp$ otherwise. Suppose $\eta = (\eta_x)_{x \in \mathsf{dom}(t)}$ is a priority annotation for t that annotates the strategy tree (t, τ). Let

$$\zeta \colon x \longmapsto (\zeta_{x \cdot -1}, \zeta_{x \cdot 0}, \zeta_{x \cdot 1}, \dots, \zeta_{x \cdot m}), \quad x \in \mathsf{dom}(t),$$

and let us write (t, ζ, η) for the labeled tree defined by

$$(t, \zeta, \eta) \colon x \longmapsto (t(x), \zeta(x), \eta_x), \quad x \in \mathsf{dom}(t).$$

Notice that (t, ζ, η) is over the finite alphabet

$$\Lambda \coloneqq \Gamma \times (2^{S \times \mathsf{Dir} \times S} \cup \{\sharp\})^{m+2} \times 2^{S \times 2^M \times S}.$$

We call (t, ζ, η) a *fully annotated strategy tree for $\mathcal{A}$ on t* and we call ζ the *strategy information* of (t, ζ, η). Thus, given a node x, the label of x in a fully annotated strategy tree carries information on the strategy labellings τ_y for all neighbors y of x. In addition, if a neighbor in a certain direction does not exist, this is indicated in the label of x via the special symbol $\sharp$, and (t, ζ, η) is thus also capable of storing information concerning the existence of its neighbors. We use the same terminology for fully annotated strategy trees as introduced for annotated strategy trees, that is, a *downward path in* (t, ζ, η) is just a downward path in (t, τ, η), and (t, ζ, η) is *accepting* iff (t, τ, η) is.

The following lemma shows that fully annotated strategy trees are recognizable by 1NTAs:

LEMMA B.3. *There is a 1NTA $\mathcal{C}$ running on Λ-labeled trees that accepts a Λ-labeled tree (t, ζ, η) iff it is a fully annotated strategy tree for $\mathcal{A}$ on t. The state set of $\mathcal{C}$ is exponential in the state set of $\mathcal{A}$, and $\mathcal{C}$ can be constructed in doubly exponential time from $\mathcal{A}$, where the second exponent only depends on $|M|$.*

PROOF SKETCH. We only sketch a high-level construction here and leave the details to the reader. $\mathcal{C}$ has to perform several tasks, each of which can be considered to be performed by a separate 1NTA. The 1NTA $\mathcal{C}$ will then be the intersection of all the

separate 1NTAs. By Proposition 2.20, we have to devise each of the separate automata in such a way that they respect the size and time bounds as given in the statement of the lemma.

Suppose t' is a Λ-labeled tree and let $x \in \mathsf{dom}(t')$. Suppose that

$$t'(x) = (t(x), \zeta_{-1}^{(x)}, \zeta_0^{(x)}, \zeta_1^{(x)}, \ldots, \zeta_m^{(x)}, \eta_x),$$

and let us write t for the Γ-labeled tree defined by $t\colon x \mapsto t(x)$ as dictated by t'. The following list enumerates the tasks that are performed by the separate automata.

(i) Firstly, we must check that $\tau \coloneqq (\zeta_0^{(x)})_{x \in \mathsf{dom}(t)}$ is such that (t, τ) is indeed a strategy tree for $\mathcal{A}$ on t. Doing this is not hard – the 1NTA simply traverses the tree, checks as a first step that $\tau_\varepsilon(s_0) \neq \varnothing$, and ensures that $\delta(s, t(x))$ is satisfied by the assignment $\theta_{s,x}$ as defined in the definition of strategy trees.

(ii) Secondly, we must ensure that the strategy information $\xi_{x\cdot -1}, \xi_{x\cdot 0}, \xi_{x\cdot 1}, \ldots, \xi_{x\cdot m}$ matches the information present in the respective nodes $x\cdot -1, x\cdot 0, x\cdot 1, \ldots, x\cdot m$ (for $x \cdot 0 = x$ this is of course trivial). That is, for all $d \in \{-1, 0, 1, \ldots, m\}$, we have

$$d_t(x) \neq \varnothing \implies \zeta_0^{(x\cdot d)} = \zeta_d^{(x)}. \tag{B.1}$$

Achieving this is computationally the most expensive task: Being at node x, the 1NTA has to remember in its state the information presented by $\zeta_0^{(x)}$ in the current node. It passes this information down to its children via states. The copies of the 1NTA that read the labels of the children can use the information present in the states to verify condition (B.1). Notice that the 1NTA has to have an exponential supply of states in order to be able to store the information of $\zeta_0^{(x)}$.

(iii) Thirdly, we have to ensure that $\eta \coloneqq (\eta_x)_{x \in \mathsf{dom}(t)}$ is a priority annotation that annotates the strategy tree $(t, (\eta_0^{(x)})_{x \in \mathsf{dom}(t)})$. Doing this is easy, since we can use the strategy information $\zeta_{-1}^{(x)}, \zeta_0^{(x)}, \zeta_1^{(x)}, \ldots, \zeta_m^{(x)}$ of the neighboring nodes to verify the conditions that η has to satisfy.

It is not hard to check that all the automata described can be constructed within the required size bound. □

We are now ready to sketch a proof of Theorem 2.28. The proof of this result closely follows an according result in [133] with the difference that we always rely on automata that work on finite objects only.

Proof sketch of Theorem 2.28. Suppose t' is a Λ-labeled tree and let $x \in \mathsf{dom}(t')$. Suppose that

$$t'(x) = (t(x), \zeta_{-1}^{(x)}, \zeta_0^{(x)}, \zeta_1^{(x)}, \ldots, \zeta_m^{(x)}, \eta_x),$$

and let us again write t for the Γ-labeled tree defined by $t\colon x \mapsto t(x)$ as dictated by t'.

We first construct an automaton $\mathcal{B}$ that accepts a finite t' iff (i) t' is a fully annotated strategy tree for $\mathcal{A}$ on t, and (ii) t' is accepting. $\mathcal{B}$ is the intersection of two automata, the first one is $\mathcal{C}$ from Lemma B.3, and the second one is $\mathcal{B}'$ whose construction we are going to describe in the following.

$\mathcal{B}$ works on fully annotated strategy trees for $\mathcal{A}$. Thus, suppose (t, ζ, η) is a fully annotated strategy tree for $\mathcal{A}$ on the finite Γ-labeled tree t. $\mathcal{B}'$ has to ensure that (t, ζ, η) is indeed accepting – by Lemma B.2, this entails that t is accepted by $\mathcal{A}$. Let $\alpha = (x_0, s_0, \beta_0), (x_1, s_1, \beta_1), \ldots$ be a downward path in (t, ζ, η). Since t is finite, it is easy to check that α must be finite as well – i.e., $\alpha = (x_0, s_0, \beta_0), \ldots, (x_\ell, s_\ell, \beta_\ell)$ for some $\ell \geq 0$ with $\beta_\ell = (H, s_\ell) \in \eta_{x_\ell}(s_\ell)$. Hence, α violates the parity condition iff $\Omega(\alpha) = \Omega(s_\ell)$ is odd. We call the sequence $(s_0, \beta_0), \ldots, (s_\ell, \beta_\ell)$ the *projection of* α. Notice that the projection of α is over the finite alphabet

$$S \times ((S \times \{-1, 0, 1, \ldots, m\} \times S) \cup (S \times 2^M \times S)),$$

and we can devise a non-deterministic word automaton $\mathcal{N}$ that accepts a projection of a finite downward path α iff α violates the parity condition. All that $\mathcal{N}$ does is (i) checking that α indeed satisfies the properties of a downward path, and (ii) checking whether there is some (s_i, β_i) with $\beta_i = (H, s_i)$, in which case it accepts if $\Omega(\beta_i) = \min H$ is odd. To achieve this, it only needs linearly many states in $|S|$ (when it traverses the sequence $\alpha = (s_0, \beta_0), \ldots, (s_\ell, \beta_\ell)$, it has to remember s_i when proceeding from (s_i, β_i) to (s_{i+1}, β_{i+1}) in order to check whether α is indeed a downward path). From $\mathcal{N}$, we can construct another non-deterministic word automaton $\mathcal{N}'$ that operates on sequences on words over the alphabet

$$(2^{S \times \mathsf{Dir} \times S} \cup \{\sharp\})^{m+2} \times 2^{S \times 2^M \times S}.$$

The automaton $\mathcal{N}'$ thus reads words that contain the strategy information and the priority annotation of (t, ζ, η). It accepts if and only if it finds a projection of a downward path that violates the parity condition. The state set of $\mathcal{N}'$ is the same as that of $\mathcal{N}$, but it has to operate on a richer alphabet and guesses possible projections of downward paths. Now let $\mathcal{N}''$ be the determinization of the complement of $\mathcal{N}'$. The construction of $\mathcal{N}''$ involves the standard powerset construction [120] and incurs an exponential blowup in the size of $\mathcal{N}'$. Moreover, the construction be carried out in exponential time [120]. We define $\mathcal{B}'$ to be the 1DTA that is obtained by simply running $\mathcal{N}''$ in parallel over all branches – notice that $\mathcal{B}'$ knows the successors of each node due to the strategy information present in the labels of (t, ζ, η).

Now recall that $\mathcal{B}$ is the intersection of the automaton $\mathcal{B}'$ thus constructed and the automaton $\mathcal{C}$ from Lemma B.3. Now the desired automaton $\mathcal{A}'$ is simply obtained from $\mathcal{B}$ by building its Γ-projection so that it accepts Γ-labeled input trees (Proposition 2.22). By construction, $\mathcal{A}'$ accepts the same finite Γ-labeled trees as $\mathcal{A}$. Since, by assumption, $\mathcal{A}$ accepts finite trees only, we obtain $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ as desired. Moreover, $\mathcal{A}'$ the state set of $\mathcal{A}'$ is of exponential size in the size of the state set of $\mathcal{A}$. □

B.2 Proofs for Chapter 4

B.2.1 Proof of Proposition 4.5

The construction underlying Proposition 4.5 relies on the idea of encoding Boolean operations (in our case the 'or' operator) using a set of atoms. This idea has been exploited in several other works; see, e.g., [29, 41, 83]. Let $Q = (\mathbf{S}, \mathcal{O}, q(\bar{x}))$ be an OMQ from $(\mathcal{C}, \mathsf{UCQ})$. Our goal is to construct in polynomial time a $Q' = (\mathbf{S}, \mathcal{O}', q'(\bar{x}))$ that falls into $(\mathcal{C}, \mathsf{CQ})$ such that $Q \equiv Q'$. We assume, w.l.o.g., that the predicates of $\mathbf{S}$ do not appear in the head of a TGD of $\mathcal{O}$. Indeed, we can copy the content of a relation $R/k \in \mathbf{S}$ into an auxiliary predicate R^*/k, using the TGD $R(x_1, \ldots, x_k) \to R^*(x_1, \ldots, x_k)$, while staying inside $\mathcal{C}$, and then rename each predicate R in $\mathcal{O}$ and $q(\bar{x})$ by R^*. The set $\mathcal{O}'$ consists of the following TGDs:

(i) For every $R/k \in \mathbf{S}$:

$$R(x_1, \ldots, x_k) \to R'(x_1, \ldots, x_k, 1), \mathsf{True}(1).$$

These TGDs are annotating the database atoms with the truth constant "true," indicating that these are true atoms.

(ii) Assuming that $q(\bar{x}) = \exists \bar{y}\, \varphi(\bar{x}, \bar{y})$, a TGD

$$\mathsf{True}(t) \to \exists \bar{x}, \bar{y}, f\, (\varphi'_\wedge(\bar{x}, \bar{y}, f) \wedge \psi(t, f)),$$

where $\varphi'_\wedge$ is the conjunction of atoms in φ after replacing each atom $R(v_1, \ldots, v_k)$ with $R'(v_1, \ldots, v_k, f)$, and ψ is the conjunction of atoms

$$\mathsf{False}(f), \mathsf{Or}(t, t, t), \mathsf{Or}(t, f, t), \mathsf{Or}(f, t, t), \mathsf{Or}(f, f, f).$$

This TGD generates a "copy" of the atoms in q, while annotating them with a null value that represents the truth constant "false," indicating that are not necessarily true atoms. Moreover, the truth table of 'or' is generated.

(iii) Finally, for each TGD $\varphi(\bar{x}, \bar{y}) \to \exists \bar{z}\, \psi(\bar{x}, \bar{z})$ in $\mathcal{O}$, where $\varphi(\bar{x}, \bar{y})$ and $\psi(\bar{x}, \bar{z})$ are conjunctions of atoms, we add a TGD

$$\varphi'(\bar{x}, \bar{y}, w) \to \exists \bar{z}\, \psi'(\bar{x}, \bar{z}, w),$$

where φ' and ψ' are obtained from φ and ψ, respectively, by replacing each atom $R(v_1, \ldots, v_k)$ with $R'(v_1, \ldots, v_k, w)$. In fact, this is the actual set of TGDs $\mathcal{O}$, with the difference that the value at the last position of each atom – which indicates whether it is true or false – is propagated to the inferred atoms.

Now, assuming that $q(\bar{x}) = q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$, the CQ $q'(\bar{x})$ is defined as follows. Let $\bar{v} = v_1 \ldots v_n$ and $\bar{w} = w_1 \ldots w_{n+1}$. Then:

$$q'(\bar{x}) \coloneqq \exists \bar{v}, \bar{w}\, (\mathsf{False}(w_1) \wedge \bigwedge_{1 \leq i \leq n} (q'_i[v_i] \wedge \mathsf{Or}(w_i, v_i, w_{i+1})) \wedge \mathsf{True}(w_{n+1})),$$

where $\bar{w}$ and $\bar{v}$ are fresh variables not in q, and $q'_i[v_i]$ is a conjunction of atoms obtained from the body atoms of q_i by replacing each atom $R(t_1, \ldots, t_k)$ by $R'(t_1, \ldots, t_k, v_i)$. This completes our construction.

It is not difficult to show that $Q \equiv Q'$. The key observation is that, in order to satisfy $\mathsf{True}(w_{n+1})$ in the CQ q', at least one of the v_i must be mapped to 1, which means that at least one q_i must be mapped via a homomorphism to $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$. Finally, it is easy to verify that, for each $\mathcal{C} \in \{\mathsf{G}, \mathsf{FG}, \mathsf{L}, \mathsf{NR}, \mathsf{S}\}$, $\mathcal{O} \in \mathsf{C}$ implies $\mathcal{O}' \in \mathcal{C}$, and Proposition 4.5 follows. □

B.2.2 Proof of Proposition 4.11

Stratification Let us first give give an alternative definition for non-recursiveness that is based on the well-known notion of *stratification*, which is more convenient for the combinatorial analysis that we are going to perform in the proof of Proposition 4.11.

Definition B.4. Consider a set $\mathcal{O}$ of TGDs. A *stratification* of $\mathcal{O}$ is a partition $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$, where $n > 0$, of $\mathcal{O}$ such that, for some function $\mu \colon \mathsf{sig}(\mathcal{O}) \to \{0, 1, \ldots, n\}$, the following hold:

(i) For each predicate $R \in \mathsf{sig}(\mathcal{O})$, all the TGDs with R in their head belong to $\mathcal{O}_{\mu(R)}$, i.e., they belong to the same set of the partition.

(ii) If there exists a TGD in $\mathcal{O}$ such that the predicate R appears in its body, while the predicate P appears in its head, then $\mu(R) < \mu(P)$.

We call μ a *stratification function* for $\mathcal{O}$. We say that $\mathcal{O}$ is *stratifiable* if it admits a stratification.

It is an easy exercise to show that the predicate graph of a set $\mathcal{O}$ of TGDs is non-recursive/acyclic iff $\mathcal{O}$ is stratifiable. Then:

Lemma B.5. *$\mathcal{O}$ is non-recursive iff it is stratifiable.*

We are now ready to prove Proposition 4.11:

Proof of Proposition 4.11. We assume, w.l.o.g., that the predicates of $\mathbf{S}$ do not appear in the head of a TGD of $\mathcal{O}$. For if predicates of $\mathbf{S}$ appear in the head of a TGD of $\mathcal{O}$, we can simply introduce auxiliary predicates and rules that copy the contents of the according relations in these predicates. Assume also that $\mathcal{O}$ is in normal form as required by XRewrite. Starting from $\mathcal{O}$, this transformation and the one that ensures that predicates from $\mathbf{S}$ do not appear in the heads of TGDs only introduce at most polynomially many new predicates. Let $p(\cdot)$ be this polynomial, and let us abuse notation and write $\mathcal{O}$ for the resulting set of TGDs as well. Since $\mathcal{O} \in \mathsf{NR}$, by Lemma B.5, $\mathcal{O}$ admits a stratification $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ with stratification function $\mu \colon \mathsf{sig}(\mathcal{O}) \to \{0, 1, \ldots, k\}$.

Let us briefly explain how the *rewriting tree* $\mathcal{T}_Q$ of the OMQ $Q = (\mathbf{S}, \mathcal{O}, q)$ is defined. $\mathcal{T}_Q$ is a rooted labeled tree with q being the label its root (let us identify nodes with their labels in the following). The i-th level of $\mathcal{T}_Q$ consists of the CQs obtained from the CQs of

the $(i-1)$-th level by applying rewriting steps (see the algorithm XRewrite in Appendix A for details on the rewriting step) using only TGDs from $\mathcal{O}_{k-i+1}$. It is easy to verify that the CQs of the i-th level contain only predicates P such that $\mu(P) < k - i + 1$. It is now clear that the k-th level of $\mathcal{T}_Q$ (i.e., the leaves of $\mathcal{T}_Q$) consists only of CQs obtained during the execution of XRewrite that contain only predicates from $\mathbf{S}$. Thus, in order to obtain the desired upper bound, it suffices to show that the number of atoms that occur in a CQ that is a leaf of $\mathcal{T}_Q$ is at most $|q| \cdot n^{p(|\mathsf{sig}(\mathcal{O})|)}$, where $n \coloneqq \max\{|\mathsf{body}(\tau)| : \tau \in \mathcal{O}\}$.

To this end, let us focus on one branch B of $\mathcal{T}_Q$ from the root q to a leaf q'. Such a branch can be naturally represented as an n-ary forest $\mathcal{F}_Q^B$, where the root nodes are the atoms of q, and whenever an atom α is resolved during the rewriting step using a TGD τ, the atoms of $\mathsf{body}(\tau)$, after applying the appropriate MGU, are the child nodes of α. Therefore, to obtain the desired upper bound, it suffices to show that the number of leaves of $\mathcal{F}_Q^B$ is at most $|q| \cdot n^{p(|\mathsf{sig}(\mathcal{O})|)}$.

By construction, $\mathcal{F}_Q^B$ consists, in general, of $|q|$ different n-ary rooted trees, where all are of depth at most k. Hence, the number of leaves of $\mathcal{F}_Q^B$ is at most $|q| \cdot n^k$. Since $k \leq p(|\mathsf{sig}(\mathcal{O})|)$, the claim follows. □

B.2.3 Proof of Theorem 4.13

A proof sketch for the coNExpTime$^{\text{NP}}$ upper bound is given in the main body of the thesis. We proceed to establish PTime$^{\text{NExpTime}}$-hardness. Our proof is by reduction from a tiling problem that has been recently introduced in [72], which in turn relies on the standard *Exponential Tiling Problem*. Let us first recall the latter problem.

An instance of the Exponential Tiling Problem is a tuple $\mathbb{T} = (n, m, H, V, \bar{s})$, where n, m are numbers (in unary), H, V are subsets of $\{1, \ldots, m\} \times \{1, \ldots, m\}$, and $\bar{s}$ is a sequence of numbers from $\{1, \ldots, m\}$. Such a tuple specifies that we desire a $2^n \times 2^n$ grid, where each cell is tiled with a tile from $\{1, \ldots, m\}$. H (respectively, V) is the horizontal (respectively, vertical) compatibility relation, while $\bar{s}$ represents a constraint on the initial part of the first row of the grid. A *solution* to such an instance of the Exponential Tiling Problem is a function $f\colon \{0, \ldots, 2^n - 1\} \times \{0, \ldots, 2^n - 1\} \to \{1, \ldots, m\}$ such that:

(i) $f(i, 0) = s_i$, for each $0 \leq i \leq |\bar{s}| - 1$.

(ii) $(f(i, j), f(i + 1, j)) \in H$, for each $0 \leq i \leq 2^n - 2$ and $0 \leq j \leq 2^n - 1$.

(iii) $(f(i, j), f(i, j + 1)) \in V$, for each $0 \leq i \leq 2^n - 1$ and $0 \leq j \leq 2^n - 2$.

We will refer to $\{0, \ldots, 2^n - 1\} \times \{0, \ldots, 2^n - 1\}$ as a *grid*, with the pairs in it being *cells*. A cell consists of two *coordinates*, the *column-coordinate* (for short col-coordinate) and the *row-coordinate*, and any function on a grid is a *tiling*. The Exponential Tiling Problem is defined as follows: given an instance $\mathbb{T}$ as above, decide whether $\mathbb{T}$ has a solution. It is known that this problem is NExpTime-hard (see, e.g., Section 3.2 of [96]).

We are now ready to recall the tiling problem introduced in [72], called *Extended Tiling Problem* (ETP), which is PTime$^{\text{NExpTime}}$-hard. An instance of this problem is a tuple $\mathbb{T} = (k, n, m, H_1, V_1, H_2, V_2)$, where k, n, m are numbers (in unary), and H_1, V_1, H_2, V_2 are subsets of $\{1, \ldots, m\} \times \{1, \ldots, m\}$. The question is as follows: is it the case that for

every sequence $\bar{s}$, where $|\bar{s}| = k$, of numbers of $\{1, \ldots, m\}$, $\mathbb{T}_1 \coloneqq (n, m, H_1, V_1, \bar{s})$ has no solution *or* $\mathbb{T}_2 \coloneqq (n, m, H_2, V_2, \bar{s})$ has a solution? If this question can be answered positively, then we say that $\mathbb{T}$ has a *solution*.

We give a reduction from the ETP to $\mathsf{Cont}(\mathsf{NR}, \mathsf{CQ})$. More precisely, given an instance $\mathbb{T} = (k, n, m, H_1, V_1, H_2, V_2)$ of the ETP, our goal is to construct in polynomial time two queries $Q_i = (\mathbf{S}, \mathcal{O}_i, q_i) \in (\mathsf{NR}, \mathsf{CQ})$, for $i \in \{1, 2\}$, such that $\mathbb{T}$ has a solution iff $Q_1 \subseteq Q_2$.

The Data Schema $\mathbf{S}$

The data schema $\mathbf{S}$ consists of:

- 0-ary predicates C_i^j, for each $i \in \{0, \ldots, k-1\}$ and $j \in \{1, \ldots, m\}$. The atom C_i^j indicates that $s_i = j$.

The Query Q_1

The goal of the query Q_1 is twofold: (i) to check that the so-called *existence property* of the input database, i.e., whether, for every $i \in \{0, \ldots, k-1\}$, there exists at least one atom of the form C_i^j, is satisfied, and (ii) to check whether $\mathcal{T}_1 \coloneqq (n, m, H_1, V_1, \bar{s})$, where $\bar{s}$ is the sequence of tilings encoded in the input database, has a solution.

To this end, the query Q_1 will mention the following predicates:

- 0-ary predicates C_i, indicating that there exists at least one atom of the form C_i^j in the input database.
- A 0-ary predicate $\mathsf{Existence}$, indicating that the input database enjoys the existence property.
- Unary predicates Tile_i, for each $i \in \{1, \ldots, m\}$. The atom $\mathsf{Tile}_i(x)$ states that x represents the tile i.
- A binary predicate H. The atom $H(x, y)$ encodes the fact that $(x, y) \in H_1$.
- A binary predicate V. The atom $V(x, y)$ encodes the fact that $(x, y) \in V_1$.
- 5-ary predicates T_i, for each $i \in \{1, \ldots, n\}$. The atom $T_i(x, x_1, x_2, x_3, x_4)$ states that x is a $(2^i \times 2^i)$-tiling obtained from the $(2^{i-1} \times 2^{i-1})$-tilings $x_1, \ldots, x_4$ – details on the inductive construction of $(2^i \times 2^i)$-tilings from $(2^{i-1} \times 2^{i-1})$-tilings are given below.
- Unary predicates $\mathsf{Initial}_i$, for each $i \in \{0, \ldots, k-1\}$. The atom $\mathsf{Initial}_i(x)$ states that $s_i = x$, i.e., the i-th element of the sequence $\bar{s}$ is x.
- Binary predicates Top_i^j, for each $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, k-1\}$. The atom $\mathsf{Top}_i^j(x, y)$ states that in the $(2^i \times 2^i)$-tiling x the tile at position $(j, 0)$ is y.
- A 0-ary predicate Tiling, indicating that there exists a $(2^n \times 2^n)$-tiling that is compatible with the initial tiling $\bar{s}$ encoded in the input database.

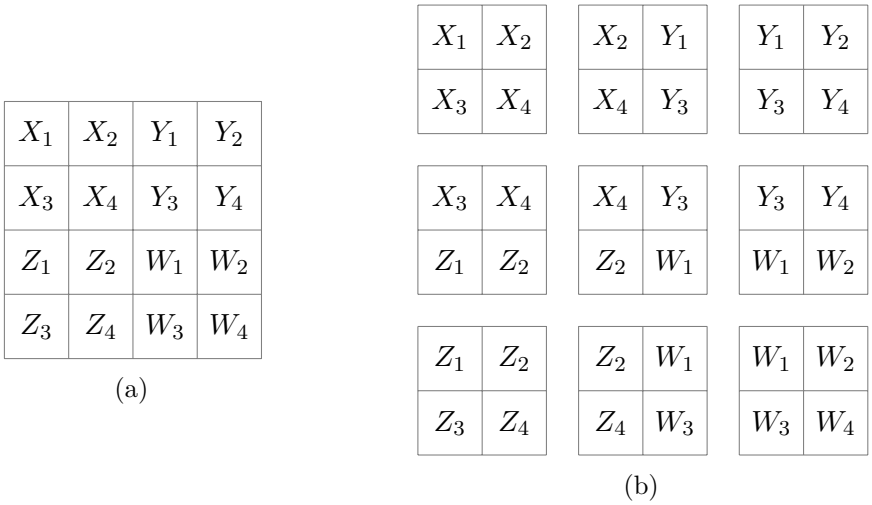

Figure B.1: Inductive construction of tilings.

- A 0-ary predicate Goal, which is derived whenever the predicates Existence and Tiling are derived.

Q_1 is defined as the query $(\mathbf{S}, \mathcal{O}_1, \mathsf{Goal})$, where $\mathcal{O}_1$ consists of the following TGDs:

- For each $i \in \{0, \ldots, k-1\}$ and $j \in \{1, \ldots, m\}$,

$$C_i^j \to C_i,$$

  and the TGD that checks for the existence property

$$C_0, \ldots, C_{k-1} \to \mathsf{Existence}.$$

- Generate the tiles:

$$\top \to \exists x_1 \ldots \exists x_m \, (\mathsf{Tile}_1(x_1) \wedge \cdots \wedge \mathsf{Tile}_m(x_m)).$$

- Generate the compatibility relations:
  For each $(i, j) \in H_1$,

$$\mathsf{Tile}_i(x), \mathsf{Tile}_j(y) \to H(x, y).$$

  For each $(i, j) \in V_1$,

$$\mathsf{Tile}_i(x), \mathsf{Tile}_j(y) \to V(x, y).$$

- Generate the $(2^n \times 2^n)$-tilings. The key idea is to inductively construct $(2^i \times 2^i)$-tilings from $(2^{i-1} \times 2^{i-1})$-tilings. It is easy to verify that the grid in Figure B.1a is

a $(2^i \times 2^i)$-tiling iff the nine subgrids of it, shown in Figure B.1b, are $(2^{i-1} \times 2^{i-1})$-tilings. This has been already observed in [67], where Datalog with complex values is studied.

First, we construct tilings of size 2×2 (the base case of the inductive construction):

$$H(x_1, x_2), H(x_3, x_4), V(x_1, x_3), V(x_2, x_4) \to \exists x\, T_1(x, x_1, x_2, x_3, x_4).$$

Then, we inductively construct tilings of larger size until we get tilings of size $2^n \times 2^n$. This is done using the following TGDs: for each $i \in \{2, \ldots, n\}$,

$$\begin{aligned}
&T_{i-1}(x_1, x_{11}, x_{12}, x_{21}, x_{22}), T_{i-1}(x_2, x_{12}, x_{13}, x_{22}, x_{23}), T_{i-1}(x_3, x_{13}, x_{14}, x_{23}, x_{24}),\\
&T_{i-1}(x_4, x_{21}, x_{22}, x_{31}, x_{32}), T_{i-1}(x_5, x_{22}, x_{23}, x_{32}, x_{33}), T_{i-1}(x_6, x_{23}, x_{24}, x_{33}, x_{34}),\\
&T_{i-1}(x_7, x_{31}, x_{32}, x_{41}, x_{42}), T_{i-1}(x_8, x_{32}, x_{33}, x_{42}, x_{43}), T_{i-1}(x_9, x_{33}, x_{34}, x_{43}, x_{44}) \to\\
&\qquad\qquad \exists x\, T_i(x, x_1, x_3, x_7, x_9).
\end{aligned}$$

- Extract from the $(2^n \times 2^n)$-tilings those tiles at positions $(0,0), (1,0), \ldots, (k-1, 0)$. This is done using the following TGDs:

$$\begin{aligned}
T_1(x, x_1, x_2, x_3, x_4) &\to \mathsf{Top}_1^0(x, x_1) \wedge \mathsf{Top}_1^1(x, x_2),\\
T_2(x, x_1, x_2, x_3, x_4), \mathsf{Top}_1^0(x_1, y_0), \mathsf{Top}_1^1(x_1, y_1) &\to \mathsf{Top}_2^0(x, y_0) \wedge \mathsf{Top}_2^1(x, y_1),\\
T_2(x, x_1, x_2, x_3, x_4), \mathsf{Top}_1^0(x_2, y_0), \mathsf{Top}_1^1(x_2, y_1) &\to \mathsf{Top}_2^2(x, y_0) \wedge \mathsf{Top}_2^3(x, y_1),\\
&\vdots\\
T_\ell(x, x_1, x_2, x_3, x_4),&\\
\mathsf{Top}_{\ell-1}^0(x_1, y_0), \ldots, \mathsf{Top}_{\ell-1}^{2^{\ell-1}-1}(x_1, y_{2^{\ell-1}-1}) &\to \mathsf{Top}_\ell^0(x, y_0) \wedge \cdots \wedge\\
&\qquad \mathsf{Top}_\ell^{2^{\ell-1}-1}(x, y_{2^{\ell-1}-1}),\\
T_\ell(x, x_1, x_2, x_3, x_4),&\\
\mathsf{Top}_{\ell-1}^0(x_1, y_0), \ldots, \mathsf{Top}_{\ell-1}^{k-2^{\ell-1}-1}(x_1, y_{k-2^{\ell-1}-1}) &\to \mathsf{Top}_\ell^{2^{\ell-1}}(x, y_0) \wedge \cdots \wedge\\
&\qquad \mathsf{Top}_\ell^{k-1}(x, y_{k-2^{\ell-1}-1}),
\end{aligned}$$

  where $\ell = \lceil \log k \rceil$. Moreover, for each $i \in \{\ell + 1, \ldots, n\}$:

$$\begin{aligned}
T_i(x, x_1, x_2, x_3, x_4), \mathsf{Top}_{i-1}^0(x_1, y_0), \ldots, \mathsf{Top}_{i-1}^{k-1}(x_1, y_{k-1}) \to\\
\mathsf{Top}_i^0(x, y_0) \wedge \cdots \wedge \mathsf{Top}_i^{k-1}(x, y_{k-1}).
\end{aligned}$$

- Check whether there exists a $(2^n \times 2^n)$-tiling that is compatible with the sequence of tilings $\bar{s}$:

  For each $i \in \{0, \ldots, k-1\}$ and $j \in \{1, \ldots, m\}$:

$$C_i^j, \mathsf{Tile}_j(x) \to \mathsf{Initial}_i(x),$$

  and the TGD

$$\mathsf{Top}_n^0(x, y_0), \mathsf{Initial}_0(y_0), \ldots, \mathsf{Top}_n^{k-1}(x, y_{k-1}), \mathsf{Initial}_{k-1}(y_{k-1}) \to \mathsf{Tiling}.$$

- Finally, we have the output TGD

$$\mathsf{Existence}, \mathsf{Tiling} \rightarrow \mathsf{Goal}.$$

This concludes the construction of Q_1.

The Query Q_2

The goal of the query Q_2 is twofold: (i) to check that the so-called *uniqueness property* of the input database, i.e., for every $i \in \{0, \ldots, k-1\}$, there exists at most one atom of the form C_i^j, is satisfied, and (ii) to check whether $\mathbb{T}_2 \coloneqq (n, m, H_2, V_2, \bar{s})$, where $\bar{s}$ is the sequence of tilings encoded in the input database, has a solution.

Q_2 mentions the same predicates as Q_1, and is defined as $Q_2 \coloneqq (\mathbf{S}, \mathcal{O}_2, \mathsf{Goal})$, where $\mathcal{O}_2$ consists of the following TGDs:

- For each $i \in \{0, \ldots, k-1\}$ and $j, \ell \in \{1, \ldots, m\}$ with $j < \ell$,

$$C_i^j, C_i^\ell \rightarrow \mathsf{Goal}.$$

- The rest of $\mathcal{O}_2$ encodes the tiling problem $\mathbb{T}_2$ in exactly the same way as $\mathcal{O}_1$ encodes the instance $\mathbb{T}_1$.

This concludes the construction of Q_2 and thus the proof of Theorem 4.13 □

B.2.4 Proof of Theorem 4.15

The coNExpTime upper bound, as well as the Π_2^{P}-hardness in case of predicates of bounded arity, are discussed in the main body of the thesis. Here, we show the coNExpTime-hardness. The proof proceeds in two steps:

(i) First, we show that $\mathsf{Cont}((\mathsf{FNR}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$ is coNExpTime-hard, where FNR denotes the class of *full* non-recursive sets of TGDs, i.e., non-recursive sets of tgds without existentially quantified variables. Notice thus that OMQs using sets of TGDs from FNR correspond to *non-recursive* Datalog queries.

(ii) Then, we reduce $\mathsf{Cont}((\mathsf{FNR}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$ to $\mathsf{Cont}((\mathsf{S}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$ by showing that (under some assumptions that are explained below) every OMQ from $(\mathsf{FNR}, \mathsf{CQ})$ can be rewritten into an OMQ from $(\mathsf{S}, \mathsf{CQ})$.

By Proposition 4.5, we immediately get that $\mathsf{Cont}((\mathsf{S}, \mathsf{CQ}), (\mathsf{L}, \mathsf{CQ}))$ is coNExpTime-hard, as needed.

Step 1: coNExpTime*-hardness of* $\mathsf{Cont}((\mathsf{FNR}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$

We show that $\mathsf{Cont}((\mathsf{FNR}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$ is coNExpTime-hard when we focus on *0-1-queries*, that is, queries Q with following property: for every database $\mathfrak{D}$, $Q(\mathfrak{D}) = Q(\mathfrak{D}_{01})$, where $\mathfrak{D}_{01} \subseteq \mathfrak{D}$ is the substructure of $\mathfrak{D}$ induced by the set $\{0, 1\}$. The proof is by reduction from the Exponential Tiling Problem, and is a non-trivial adaptation of

the one given in [29] for showing that containment of non-recursive Datalog queries is coNExpTime-hard.

Given an instance $\mathbb{T} = (n, m, H, V, \bar{s})$ of the Exponential Tiling Problem, we are going to construct an OMQ $Q_\mathbb{T}$ from $(\mathsf{FNR}, \mathsf{CQ})$, which is a 0-1-query with data schema $\mathbf{S}$, and an OMQ $Q'_\mathbb{T}$ from $(\mathsf{L}, \mathsf{UCQ})$, which is also a 0-1-query having data schema $\mathbf{S}$, such that $\mathbb{T}$ has a solution iff $Q_\mathbb{T} \not\subseteq Q'_\mathbb{T}$.

The data schema S The data schema $\mathbf{S}$ consists of:

- $2n$-ary predicates $\mathsf{TiledBy}_i$, for $i \leq m$. An atom $\mathsf{TiledBy}_i(x_1, \ldots, x_n, y_1, \ldots, y_n)$ indicates that the cell with coordinates $((x_1, \ldots, x_n), (y_1, \ldots, y_n)) \in \{0,1\}^n \times \{0,1\}^n$ is tiled by tile i. Notice that we use n-bit binary numbers to represent a coordinate, which is the key difference between our construction and the one of [29].

The query $Q_\mathbb{T}$ The goal of the query $Q_\mathbb{T}$ is to assert whether the input database encodes a candidate tiling, i.e., whether the entire grid is tiled, without taking into account the constraints, that is, the compatibility relations and the constraint on the initial part of the first row. To this end, the query $Q_\mathbb{T}$ will mention the following predicates:

- Unary predicate Bit. The atom $\mathsf{Bit}(x)$ simply says that x is a bit, i.e., $x \in \{0,1\}$.
- $2n$-ary predicates $\mathsf{TiledAboveCol}_i$, for each $i \leq n$. The atom $\mathsf{TiledAboveCol}_i(\bar{x}, \bar{y})$ says that for the row-coordinate $\bar{y}$ there are tiled cells with coordinates $(\bar{x}', \bar{y})$ for every col-coordinate $\bar{x}'$ that agrees with $\bar{x}$ on the first $i-1$ bits. In other words, for the row corresponding to $\bar{y}$, every column extending the first $i-1$ bits of $\bar{x}$ is tiled. In particular, $\mathsf{TiledAboveCol}_1(\bar{x}, \bar{y})$ says that the entire row $\bar{y}$ is tiled.
- $2n$-ary predicates $\mathsf{TiledAboveRow}_i$, for each $i \leq n$. The atom $\mathsf{TiledAboveRow}_i(\bar{y})$ says that for every $\bar{y}'$ that agrees with $\bar{y}$ on the first $i-1$ bits, the row $\bar{y}'$ is fully tiled.
- An n-ary predicate $\mathsf{RowTiled}$. The atom $\mathsf{RowTiled}(\bar{y})$ says that the row $\bar{y}$ is fully tiled.
- A 0-ary predicate $\mathsf{AllTiled}$, which asserts that the entire grid is tiled.
- A 0-ary predicate Goal, which is derived whenever the predicate $\mathsf{AllTiled}$ is derived.

$Q_\mathbb{T}$ is defined as $Q_\mathbb{T} \coloneqq (\mathbf{S}, \mathcal{O}, \mathsf{Goal})$, where $\mathcal{O}$ consists of the following rules:

- Generate the Bit-atoms:

$$\top \to \mathsf{Bit}(0), \quad \top \to \mathsf{Bit}(1).$$

- The relation $\mathsf{RowTiled}$ is defined as follows:

For each $j, k \leq m$,

$$\mathsf{TiledBy}_j(x_1, \ldots, x_{n-1}, 1, y_1, \ldots, y_n), \mathsf{TiledBy}_k(x_1, \ldots, x_{n-1}, 0, y_1, \ldots, y_n), \\ \mathsf{Bit}(x_1), \ldots, \mathsf{Bit}(x_{n-1}), \mathsf{Bit}(y_1), \ldots, \mathsf{Bit}(y_n), \mathsf{Bit}(w) \rightarrow \\ \mathsf{TiledAboveCol}_n(x_1, \ldots, x_{n-1}, w, y_1, \ldots, y_n)$$

For each $2 \leq i \leq n$,

$$\mathsf{TiledAboveCol}_i(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n, y_1, \ldots, y_n), \\ \mathsf{TiledAboveCol}_i(x_1, \ldots, x_{i-1}, 0, x'_{i+1}, \ldots, x'_n, y_1, \ldots, y_n), \\ \mathsf{Bit}(w_i), \ldots, \mathsf{Bit}(w_n) \rightarrow \\ \mathsf{TiledAboveCol}_{i-1}(x_1, \ldots, x_{i-1}, w_i, \ldots, w_n, y_1, \ldots, y_n).$$

A row is fully tiled:

$$\mathsf{TiledAboveCol}_1(x_1, \ldots, x_n, y_1, \ldots, y_n) \rightarrow \mathsf{RowTiled}(y_1, \ldots, y_n)$$

- The relation $\mathsf{AllTiled}$:

$$\mathsf{RowTiled}(y_1, \ldots, y_{n-1}, 1), \mathsf{RowTiled}(y_1, \ldots, y_{n-1}, 0), \mathsf{Bit}(w) \rightarrow \\ \mathsf{TiledAboveRow}_n(y_1, \ldots, y_{n-1}, w).$$

For each $2 \leq i \leq n$,

$$\mathsf{TiledAboveRow}_i(y_1, \ldots, y_{i-1}, 1, y_{i+1}, \ldots, y_n), \\ \mathsf{TiledAboveRow}_i(y_1, \ldots, y_{i-1}, 0, y'_{i+1}, \ldots, y'_n), \\ \mathsf{Bit}(w_i), \ldots, \mathsf{Bit}(w_n) \rightarrow \\ \mathsf{TiledAboveRow}_{i-1}(y_1, \ldots, y_{i-1}, w_i, \ldots, w_n).$$

The entire grid is tiled:

$$\mathsf{TiledAboveRow}_1(y_1, \ldots, y_n) \rightarrow \mathsf{AllTiled}, \\ \mathsf{AllTiled} \rightarrow \mathsf{Goal}.$$

This concludes the construction of the query $Q_{\mathbb{T}}$.

The query $Q'_{\mathbb{T}}$ $Q'_{\mathbb{T}}$ is defined in such a way that $Q'_{\mathbb{T}}(\mathfrak{D})$ is non-empty exactly when the input database $\mathfrak{D}$ encodes an invalid tiling, i.e., when one of the constraints on the tiles is violated. The query $Q'_{\mathbb{T}}$ will mention the following intensional predicates:

- A unary predicate Bit. As above, $\mathsf{Bit}(x)$ says that x is a bit.
- $2i$-ary predicates $\mathsf{LastFirst}_i$, for each $1 \leq i \leq n$. The atom

$$\mathsf{LastFirst}_i(x_1, \ldots, x_i, y_1, \ldots, y_i)$$

says that $(x_1, \ldots, x_i) = (1, \ldots, 1)$ and $(y_1, \ldots, y_i) = (0, \ldots, 0)$.

- $2i$-ary predicates Succ_i, for each $1 \leq i \leq n$. The atom $\mathsf{Succ}_i(\bar{x}, \bar{y})$ says that the i-bit binary number $\bar{y}$ is the successor of the i-bit binary number $\bar{x}$.
- A 0-ary predicate Goal.

$Q'_{\mathbb{T}}$ is defined as the query $Q'_{\mathbb{T}} \coloneqq (\mathbf{S}, \mathcal{O}', q')$, where $\mathcal{O}'$ and q' are specified as follows. The set $\mathcal{O}'$ consists of the following linear TGDs:

- Generate the Bit-atoms:

$$\top \to \mathsf{Bit}(0), \quad \top \to \mathsf{Bit}(1).$$

- Generate the successor predicates:

$$\top \to \mathsf{Succ}_1(0,1), \quad \top \to \mathsf{LastFirst}_1(1,0).$$

For each $1 \leq i \leq n-1$,

$$\begin{aligned}
\mathsf{Succ}_i(x_1, \ldots, x_i, y_1, \ldots, y_i) &\to \mathsf{Succ}_{i+1}(0, x_1, \ldots, x_i, 0, y_1, \ldots, y_i),\\
\mathsf{Succ}_i(x_1, \ldots, x_i, y_1, \ldots, y_i) &\to \mathsf{Succ}_{i+1}(1, x_1, \ldots, x_i, 1, y_1, \ldots, y_i),\\
\mathsf{LastFirst}_i(x_1, \ldots, x_i, y_1, \ldots, y_i) &\to \mathsf{Succ}_{i+1}(0, x_1, \ldots, x_i, 1, y_1, \ldots, y_i),\\
\mathsf{LastFirst}_i(x_1, \ldots, x_i, y_1, \ldots, y_i) &\to \mathsf{LastFirst}_{i+1}(1, x_1, \ldots, x_i, 0, y_1, \ldots, y_i).
\end{aligned}$$

The UCQ q' consists of the following (Boolean) CQs. For brevity, the existential quantifiers in front of the CQs are omitted:

- Tile consistency: for each $i \neq j \leq m$,

$$\begin{aligned}
&\mathsf{TiledBy}_i(x_1, \ldots, x_n, y_1, \ldots, y_n) \wedge \mathsf{TiledBy}_j(x_1, \ldots, x_n, y_1, \ldots, y_n) \wedge\\
&\qquad \mathsf{Bit}(x_1) \wedge \cdots \wedge \mathsf{Bit}(x_n) \wedge \mathsf{Bit}(y_1) \wedge \cdots \wedge \mathsf{Bit}(y_n).
\end{aligned}$$

- Tile compatibility:
  For each $(i,j) \notin V$,

$$\begin{aligned}
&\mathsf{Succ}_n(x_1, \ldots, x_n, y_1, \ldots, y_n) \wedge\\
&\quad \mathsf{TiledBy}_i(w_1, \ldots, w_n, x_1, \ldots, x_n) \wedge \mathsf{TiledBy}_i(w_1, \ldots, w_n, y_1, \ldots, y_n) \wedge\\
&\qquad \mathsf{Bit}(w_1) \wedge \cdots \wedge \mathsf{Bit}(w_n).
\end{aligned}$$

  For each $(i,j) \notin H$,

$$\begin{aligned}
&\mathsf{Succ}_n(x_1, \ldots, x_n, y_1, \ldots, y_n) \wedge\\
&\quad \mathsf{TiledBy}_i(x_1, \ldots, x_n, w_1, \ldots, w_n) \wedge \mathsf{TiledBy}_i(y_1, \ldots, y_n, w_1, \ldots, w_n) \wedge\\
&\qquad \mathsf{Bit}(w_1) \wedge \cdots \wedge \mathsf{Bit}(w_n).
\end{aligned}$$

- Tiling of the first row:

  For each $j \leq n$, let $f_j \colon \{1, \ldots, n\} \to \{0, 1\}$ be such that $f_j(1) \cdots f_j(n)$ is the number j in binary representation, and let $k \in \{1, \ldots, m\}$ be different to s_j (recall that $\bar{s}$ is a sequence of numbers from $\{1, \ldots, m\}$ that represents a constraint on the initial part of the first row of the grid). Then, we have the CQ

  $$\mathsf{TiledBy}_k(x_1, \ldots, x_n, \underbrace{z, \ldots, z}_{n}) \wedge \mathsf{Succ}_1(z, o),$$

  where, for each $i \in \{1, \ldots, n\}$, $x_i = z$ if $f_j(i) = 0$, and $x_i = o$ if $f_j(i) = 1$.

This concludes the definition of the query $Q'_{\mathbb{T}}$, and hence the proof that the problem $\mathsf{Cont}((\mathsf{FNR}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$ is coNExpTime-hard.

Step 2: coNExpTime*-hardness of* $\mathsf{Cont}((\mathsf{S}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$

Let F denote the class of finite sets of full TGDs, i.e, those that do not contain rules with existential quantifiers in their heads. Our goal is show that every 0-1-query $Q = (\mathbf{S}, \mathcal{O}, q)$ from $(\mathsf{F}, \mathsf{CQ})$ can be equivalently rewritten as a 0-1-query $Q' = (\mathbf{S}, \mathcal{O}', q')$, where all the TGDs of $\mathcal{O}'$ are *lossless*, i.e., all the body-variables appear also in the head, which in turn implies that $\mathcal{O}'$ is sticky.

Proposition B.6. *Consider a 0-1-query $Q \in (\mathsf{F}, \mathsf{CQ})$. We can construct in polynomial time a 0-1-query $Q' \in (\mathsf{S}, \mathsf{CQ})$ such that $Q \equiv Q'$.*

Proof. Let $Q = (\mathbf{S}, \mathcal{O}, q)$, and assume that n is the maximum number of variables occurring in the body of a TGD of $\mathcal{O}$. We are going to construct in polynomial time a 0-1-query $Q' = (\mathbf{S}, \mathcal{O}', q')$ from $(\mathsf{S}, \mathsf{CQ})$ such that $Q \equiv Q'$.

The set $\mathcal{O}'$ consists of the following TGDs:

- Initialization rules:

  We first transform every database atom of the form $R(\bar{c})$ into an atom

  $$R'(\bar{c}, \underbrace{0, \ldots, 0}_{n}, 0, 1).$$

  This is done by adding the following rules to $\mathcal{O}'$:

  $$\top \to \mathsf{Bit}(0), \quad \top \to \mathsf{Bit}(1)$$

  and, for each k-ary predicate $R \in \mathbf{S}$, we have the lossless TGD

  $$R(x_1, \ldots, x_k), \mathsf{Bit}(x_1), \ldots, \mathsf{Bit}(x_k) \to R'(x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{n}).$$

  Notice that we can safely force the variables $x_1, \ldots, x_k$ to take only values from $\{0, 1\}$ due to Q being a 0-1-query.

- Transformation into lossless TGDs:
  For each TGD $\tau \in \mathcal{O}$ of the form

  $$R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k) \to R_0(\bar{x}_0),$$

  we have the lossless TGD

  $$R'_1(\bar{x}_1, \underbrace{0, \ldots, 0}_{n}), \ldots, R'_k(\bar{x}_k, \underbrace{0, \ldots, 0}_{n}) \to R'_0(\bar{x}_0, y_1, \ldots, y_n),$$

  where, if $\{v_1, \ldots, v_\ell\}$, for $\ell \in \{1, \ldots, n\}$, is the set of variables occurring in the body of τ (the order is not relevant), then $y_i = v_i$, for each $i \in \{1, \ldots, \ell\}$, and $y_j = v_1$, for each $j \in \{\ell + 1, \ldots, n\}$.
- Finalization rules:
  Observe that each atom obtained during the chase due to one of the lossless TGDs introduced above is of the form $R'(\bar{x}, \bar{y})$, where $\bar{y} \in \{0, 1\}^n$. If $\bar{y} \neq (0, \ldots, 0)$, then we need to ensure that eventually the atom

  $$R'(\bar{x}, \underbrace{0, \ldots, 0}_{n})$$

  will be inferred. This is achieved by adding to $\mathcal{O}'$ the following TGDs. For each k-ary predicate R occurring in $\mathcal{O}$, and for each $1 \leq i \leq n$, we have the rule:

  $$R'(x_1, \ldots, x_k, y_1, \ldots, y_{i-1}, 1, y_{i+1}, \ldots, y_n) \to R'(x_1, \ldots, x_k, y_1, \ldots, y_{i-1}, 0, y_{i+1}, \ldots, y_n).$$

This concludes the definition of $\mathcal{O}'$.

The CQ q' is defined analogously. More precisely, assuming that q is of the form (the existential quantifiers are omitted)

$$R_1(\bar{x}_1) \wedge \cdots \wedge R_k(\bar{x}_k),$$

then the CQ q' is defined as

$$R'_1(\bar{x}_1, \underbrace{0, \ldots, 0}_{n}) \wedge \cdots \wedge R'_k(\bar{x}_k, \underbrace{0, \ldots, 0}_{n}).$$

It is easy to verify that $\mathcal{O}'$ consists of lossless TGDs, and thus $Q' \in (\mathsf{S}, \mathsf{CQ})$. It also not difficult to see that, for every $\mathbf{S}$-database $\mathfrak{D}$, $Q(\mathfrak{D}_{01}) = Q'(\mathfrak{D}_{01})$. Thus, since Q is a 0-1-query, $Q(\mathfrak{D}) = Q'(\mathfrak{D})$, and the claim follows. □

By the fact that $\mathsf{Cont}((\mathsf{FNR}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$ is coNEXPTIME-hard and Proposition B.6, we immediately get that $\mathsf{Cont}((\mathsf{S}, \mathsf{CQ}), (\mathsf{L}, \mathsf{UCQ}))$ is coNEXPTIME-hard, as required. □

B.2.5 Proof of Lemma 4.49

The fact that item (ii) implies item (i) is immediate, we thus focus on the other direction for which we mainly follow [18].

Let h be a homomorphism from $q(\bar{x})$ to $\mathfrak{A}$ that witnesses $\mathfrak{A} \models q(\bar{a})$. Let $\delta = (\mathcal{T}, (X_v)_{v \in T})$ be a tree decomposition that witnesses that $\mathfrak{A}$ is acyclic. Since δ is guarded, we can pick for each $v \in T$ a fact β_v such that (i) β_v contains all elements from X_v as arguments, and (ii) $\mathfrak{A}_\delta(v) \models \beta_v$. Moreover, for each atom α of $q(\bar{x})$, there is a node v_α such that $\mathfrak{A}_\delta(v_\alpha), h \models \alpha$. Let W be the set of all these v_α closed off under greatest lower bounds w.r.t. $\preceq_{\mathcal{T}}$. Consider the set $Q^+ \coloneqq h(q) \cup \{\beta_v \mid v \in W\}$ (Q^+ may also contain equality atoms). Notice that at least half of the nodes in W are of the form v_α – hence, $|Q^+| \leq 3|q|$. Now from Q^+ we can construct an acyclic CQ q^+ by identifying elements named in the atoms of Q^+ by variables. In particular, we ensure that the elements $h(\bar{a})$ are renamed to $\bar{x}$. We existentially quantify over every variable in q^+ except those among $\bar{x}$. It may be the case that not all of the variables of $\bar{x}$ are answer variables of q^+, as h may not be injective on $[\bar{x}]$. In this case, for each variable x_i among $\bar{x}$ that is not an an answer variable of q^+ we add a conjunct $x_i = v$, where v is an arbitrary answer variable (notice that v exists by assumption). This step is repeated exhaustively for all such x_i, call the resulting CQ $p(\bar{x})$. It is clear that $|p| \leq 3|q| + |\bar{x}|$ and that $\mathfrak{A} \models p(\bar{a})$ by construction. Moreover, it is also easy to see that $p(\bar{x}) \models q(\bar{x})$. □

B.2.6 Proof of Lemma 4.50

Throughout the proof, let $\bar{x} \coloneqq x_1, \ldots, x_n$ and $\bar{a} \coloneqq a_1, \ldots, a_n$.

Suppose first that $\mathfrak{A} \models q(\bar{a})$, and let h be a homomorphism witnessing this fact. Let $\mathcal{T}_1, \ldots, \mathcal{T}_k$ enumerate the subtrees rooted at the children of the root ε of $\mathcal{T}$. For $i = 1, \ldots, k$, let

$$\mathfrak{A}_0 \coloneqq \mathfrak{C} \quad \text{and} \quad \mathfrak{A}_i \coloneqq \bigcup_{v \in T_i} \mathfrak{A}_\delta(v).$$

Let $\alpha_1, \ldots, \alpha_n$ be an enumeration of all the atoms from $q(\bar{x})$, and let $h(\alpha_1), \ldots, h(\alpha_n)$ denote the result of these atoms by applying h as a substitution to them – hence, $h(\alpha_i)$ may also be an equality atom of the form $c = c$, where $c \in \mathsf{dom}(\mathfrak{A})$. Let $A_0, A_1, \ldots, A_k$ be a partition of the atoms $h(\alpha_1), \ldots, h(\alpha_n)$ so that $h(\alpha_i) \in A_j$ implies that α_i is true in $\mathfrak{A}_j$ under the assignment h. For $i = 0, 1, \ldots, k$, let q_i be the CQ that results from A_i by taking the conjunction of all atoms contained in it, and by replacing the elements of $\mathsf{dom}(\mathfrak{A})$ that also occur in A_i by variables in such a way that $\bar{a}$ is renamed to $\bar{x}$. Let $\ell \in \{0, 1, \ldots, k\}$ be such that the h-image of the answer guard of $q(\bar{x})$ is contained in A_ℓ. In q_ℓ we declare all the variables that are among $\bar{x}$ as answer-variables, while for $i \neq \ell$, we close q_i off under existential quantification so that q_i is boolean. Thus, only q_ℓ has answer variables at all, provided $q(\bar{x})$ has some. However, it may be the case that q_ℓ does not have exactly the variables $\bar{x}$ as answer variables. Suppose x_i occurs in $\bar{x}$ but not in q_ℓ. Then we pick a $v \in \mathsf{free}(q_\ell)$ and add to q_ℓ the conjunct $v = x_i$. We do this exhaustively for all such variables x_i. For $i = 0, 1, \ldots, k$, let $\bar{x}_i$ denote the answer variables of q_i.

Now it is easy to check that, for $i = 1, \dots, k$, $\mathfrak{A}_i$ is acyclic. Hence, for $i = 1, \dots, k$, we now invoke Lemma 4.49 in order to obtain a strictly acyclic $p_i(\bar{x}_i)$ over $\mathbf{T}$ such that $p_i(\bar{x}_i) \models q_i(\bar{x}_i)$ and $\mathfrak{A}_i \models p_i(h(\bar{x}_i))$. Let h_i be a homomorphism witnessing that $\mathfrak{A}_i \models p_i(h(\bar{x}_i))$, and let h_0 be a homomorphism witnessing that $\mathfrak{A}_0 \models q_0(h(\bar{x}_0))$. We claim that $\theta \coloneqq (q_0, p_1, \dots, p_k)$ is a squid decomposition of $q(\bar{x})$ with the desired properties, and that $h_0, h_1, \dots, h_k$ are the desired homomorphisms.

By construction, $\theta = q_0 \wedge p_1 \wedge \cdots \wedge p_k$ has the same free variables as $q(\bar{x})$, and one of them has exactly the variables $\bar{x}$ as answer variables (namely q_0 in case $\ell = 0$ and p_ℓ otherwise). Moreover, it is easy to check that $\theta(\bar{x}) \models q(\bar{x})$ and that all sub-items of item (ii) are satisfied by construction. Also note that $|q_0| \leq |q|$ and $|\mathsf{var}(q_0)| \leq |\mathsf{var}(q)|$. Now Lemma 4.49 tells us that $|p_i| \leq 3|q_i| + |\bar{x}_i|$ and we also know that $|\bar{x}_i| = 0$ for all $i \neq \ell$. From this and

$$|q_0| + |q_1| + \cdots + |q_k| \leq |q| + |\bar{x}|,$$

(recall here again that we added equality atoms to q_ℓ in order to accommodate missing answer variables) we obtain

$$\begin{aligned} |q_0| + |p_1| + \cdots + |p_k| &\leq |q_0| + 3|q_1| + \cdots + 3|q_k| + |\bar{x}_\ell| \\ &\leq 3(|q| + |\bar{x}|) + |\bar{x}_\ell| \\ &= 3|q| + 4|\bar{x}|. \end{aligned}$$

Thus, θ satisfies the required size bounds, and θ is the squid decomposition of $q(\bar{x})$ we are looking for.

The fact that item (ii) implies item (i) is immediate since item (ii) implies that $\mathfrak{A} \models \theta(\bar{a})$, whence $\theta(\bar{x}) \models q(\bar{x})$ gives $\mathfrak{A} \models q(\bar{a})$. □

B.2.7 Proof of Lemma 4.60

We are going to construct $\mathfrak{A}_q = (S, \Gamma_{\mathbf{S}}, \{0, \updownarrow\}, \delta, s_0, \Omega)$. Let $x_1, \dots, x_n$ be the variables of $\mathsf{var}_{\geq 1}(q)$ and fix a total order $x_1 \prec x_2 \prec \cdots \prec x_n$ among them. Define the state set S to be

$$S \coloneqq \{s_{y,\theta} \mid \theta \colon V \to U_{\mathbf{S}}, V \subseteq \mathsf{var}_{\geq 2}(q), y \in \mathsf{var}_{\leq 1}(q) \cup \{\sharp\}\},$$

where $\sharp \notin \mathsf{var}(q)$ is a special symbol. Notice that $|S| = O(|\mathsf{var}_{\leq 1}(q)| \cdot \mathsf{wd}(\mathbf{S})^{|\mathsf{var}_{\geq 2}(q)|})$. Let us set $\Omega(s) \coloneqq 1$ for all $s \in S$, and $s_0 \coloneqq s_{\sharp,\varnothing}$ where $\varnothing$ denotes the empty substitution.

For brevity, let us set $X \coloneqq \mathsf{var}_{\geq 2}(q)$ in the following. For the presentation of δ, let us introduce some auxiliary notation. Given a set of atoms A whose arguments are elements from $U_{\mathbf{S}}$ and variables from $\mathsf{var}(q)$ and a symbol $\rho \in \Gamma_{\mathbf{S}}$, a *homomorphism* from A to ρ is a function $h \colon \mathsf{var}(A) \cup U_{\mathbf{S}} \to U_{\mathbf{S}}$ that is the identity on $U_{\mathbf{S}}$ such that if $R(t_1, \dots, t_k) \in A$, then also $R_{h(t_1),\dots,h(t_k)} \in \rho$. Abusing notation, we write $\rho \models A$ if there is a homomorphism from A to ρ.

Now, for $\rho \in \Gamma_{\mathbf{S}}$ and $s_{y,\theta} \in S$, we define $\delta(s_{y,\theta}, \rho)$ by case distinction as follows:

(i) If $y = \sharp$, we distinguish the following sub-cases:

(a) Suppose first that there is an atom $\alpha(\bar{v})$ occurring in q such that $Y \coloneqq \mathsf{dom}(\theta) \cap [\bar{v}] \cap X \neq \varnothing$ and $\theta(Y) \cap U_{\mathbf{S}} \not\subseteq \mathsf{names}(\rho)$. Then $\delta(s_{\sharp,\theta}, \rho) \coloneqq \mathsf{false}$.

(b) Otherwise, if $\mathsf{dom}(\theta) = X$, then we set $\delta(s_{\sharp,\theta}, \rho) \coloneqq \langle 0 \rangle s_{x_1,\theta}$.

(c) Otherwise, we set

$$\delta(s_{\sharp,\theta}, \rho) \coloneqq \bigvee \{ \langle 0 \rangle s_{\sharp,\eta} \mid \eta \supseteq \theta, \rho \models \eta(\mathsf{body}(q)) \setminus \theta(\mathsf{body}(q)) \} \vee \langle \updownarrow \rangle s_{\sharp,\theta}.$$

(ii) Suppose now that $y = x_i$ for some $i = 1, \ldots, n$. Let α_i denote the unique body atom of q that has an occurrence of x_i. Let $\alpha_{i,\theta}$ be the atom resulting from α_i by applying the substitution θ to its arguments. We set

$$\delta(s_{x_i,\theta}, \rho) \coloneqq \begin{cases} \langle 0 \rangle s_{x_{i+1},\theta}, & \text{if } \rho \models \{\alpha_{i,\theta}\} \text{ and } i < n, \\ \mathsf{true}, & \text{if } \rho \models \{\alpha_{i,\theta}\} \text{ and } i = n, \\ \langle \updownarrow \rangle s_{x_i,\theta}, & \text{otherwise.} \end{cases}$$

Intuitively, the automaton works in two passes. The first pass consists of the runs working on states of the form $s_{\sharp,\theta}$. In this pass, the automaton tries to find an assignment for the variables in the query that appear in at least two distinct atoms. When a candidate assignment θ is found, the automaton changes to state $s_{x_1,\theta}$ which is the beginning of the second pass. A state of the form $s_{x_i,\theta}$ means that the assignment θ can be extended to all variables $x \prec x_i$ and, in this state, the automaton tries to extend θ to cover the variable x_i. The automaton makes use of the fact that the variable x_i occurs in only one atom α_i. Therefore, to extend θ to x_i, it suffices to find a single value from $U_{\mathbf{S}}$ that can be assigned to x_i in such a way that the resulting fact is named in a label of the input tree. The automaton accepts iff it is able to extend the candidate assignment θ to all variables $x_1, \ldots, x_n$. □

B.3 Proofs for Chapter 5

B.3.1 Proof of Theorem 5.3

The only missing part of the proof of Theorem 5.3 is the 2ExpTime lower bound for the problem $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{AQ})$. We are going to show in the following that actually $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ is 2ExpTime-hard. Recall that in [23] it is shown that the problem of deciding whether an OMQ from $(\mathsf{G}, \mathsf{BAQ})$ is contained in a Boolean acyclic CQ is 2ExpTime-hard. As we mentioned in Section 4.3, we can view a Boolean acyclic CQ as an OMQ from $(\mathsf{G}, \mathsf{BAQ})$ that is even first-order rewritable. Hence, the following result is implicit in [23]:

Theorem B.7. *The problem of deciding whether or not an OMQ $Q_1 = (\mathbf{S}, \mathcal{O}, G_1)$ from $(\mathsf{G}, \mathsf{AQ}_0)$ is contained in an OMQ $Q_2 = (\mathbf{S}, \mathcal{O}, G_2)$ is hard for* 2ExpTime. *This is true even for the case where Q_2 is first-order rewritable.*

To prove that $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ is hard for 2ExpTime, we are going to reduce the problem mentioned in Theorem B.7 to $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$.

Let $Q_1 = (\mathbf{S}, \mathcal{O}_1, G_1)$ and $Q_2 = (\mathbf{S}, \mathcal{O}_2, G_2)$ be as in the hypothesis of Theorem B.7. Without loss of generality, we may assume that the predicates Q_1 and Q_2 use and that do not appear in $\mathbf{S}$ are distinct. We are going to construct an OMQ Q' that falls into $(\mathsf{G}, \mathsf{BAQ})$ such that Q' is first-order rewritable iff Q_1 is contained in Q_2.

Let $Q' = (\mathbf{S}, \mathcal{O}', G_2)$, where

- $\mathbf{S}' \coloneqq \mathbf{S} \cup \{R/2, A/1, B/1\}$.
- $\mathcal{O}'$ is the union of $\mathcal{O}_1$ and $\mathcal{O}_2$ plus the rules

$$R(x, y), A(y) \to A(x),$$
$$A(x), B(x), G_1 \to G_2.$$

Notice that G_2 is also the query component of Q'.

Lemma B.8. *Q_1 is contained in Q_2 iff Q' is first-order rewritable.*

Proof. Assume first that Q_1 is not contained in Q_2. Then there is an $\mathbf{S}$-database $\mathfrak{D}$ such that $\mathfrak{D} \models Q_1$ and $\mathfrak{D} \not\models Q_2$. By Lemma 5.31, there is a $\mathfrak{B}$ of tree-width at most $\mathsf{wd}^*(\mathbf{S})$ such that $\mathfrak{B} \models Q_1$ and a weak homomorphism from $\mathfrak{B}$ to $\mathfrak{D}$. Since Q_2 is closed under weak homomorphisms, we must also have $\mathfrak{B} \not\models Q_2$. For each $k > 0$, let $\mathfrak{D}_k$ be the $\mathbf{S}'$-database extending $\mathfrak{B}$ by the facts

$$B(a_0), R(a_0, a_1), \dots, R(a_{k-1}, a_k), A(a_k),$$

where $a_0, \dots, a_k$ do not occur in $\mathsf{dom}(\mathfrak{B})$. It is easy to check that $\mathfrak{D}_k \models Q'$ for all $k > 0$. Moreover, no proper substructure of $\mathfrak{D}_k$ satisfies Q'. By virtue of Proposition 5.7, Q' is thus not first-order rewritable.

Conversely, suppose that Q_1 is contained in Q_2. Since Q_2 is first-order rewritable, there is a UBCQ q over $\mathbf{S}$ that is equivalent to Q_2. We claim that q is a UCQ-rewriting of Q' as well. Indeed, suppose first that $\mathfrak{D} \models q$ for some $\mathbf{S}'$-database $\mathbf{S}$. Since q uses only symbols from $\mathbf{S}$, we obtain that $\mathfrak{D} \restriction \mathbf{S} \models q$ as well. Since q is equivalent to Q_2, we get $\mathfrak{D} \restriction \mathbf{S} \models Q_2$ and, by construction of Q', so $\mathfrak{D} \models Q'$. Suppose now that $\mathfrak{D} \models Q'$ for some $\mathbf{S}'$-database $\mathfrak{D}$. By construction of Q', we must then have $\mathfrak{D} \models Q_2$ or $\mathfrak{D} \models Q_1$. In the former case, we are done since Q_2 and q are equivalent. In the latter case, we get $\mathfrak{D} \restriction \mathbf{S} \models Q_1$, whence $\mathfrak{D} \restriction \mathbf{S} \models Q_2$, since Q_1 is contained in Q_2. Therefore also $\mathfrak{D} \restriction \mathbf{S} \models q$ and thus $\mathfrak{D} \models q$. This proves the claim. □

It is clear that Q' can be constructed from Q_1 and Q_2 in polynomial time. Therefore, 2ExpTime-hardness for $\mathsf{FO}^{\Leftarrow}(\mathsf{G}, \mathsf{BAQ})$ follows by virtue of Lemma B.8. □

B.4 Proofs for Chapter 6

B.4.1 Proof of Theorem 6.6

Let us call a *branch of the unfolding of q with $\mathcal{O}$* a sequence of CQs $q_0, q_1, \dots, q_n$, where $q = q_0$, while, for each $i \in [n]$, q_i is a σ-resolvent of q_{i-1} for some $\sigma \in \mathcal{O}$. It is not difficult to see that the following statements are equivalent:

- There exists a proof tree $\mathcal{P}$ of q w.r.t. $\mathcal{O}$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$.
- There exists a branch $q_0, q_1, \ldots, q_n$, for some $n \geq 0$, of the unfolding of q with $\mathcal{O}$ such that $\mathfrak{D} \models q_n(\bar{a})$.

Thus, to establish Theorem 6.6, it suffices to show that the following statements are equivalent:

(a) $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$.

(b) There exists a branch $q_0, q_1, \ldots, q_n$, for some $n \geq 0$, of the unfolding of q with $\mathcal{O}$ such that $\mathfrak{D} \models q_n(\bar{a})$.

Proof of (a) $\Rightarrow$ (b)

We first establish the following auxiliary lemma:

Lemma B.9. *Consider a CQ $q(\bar{x})$, and an initial segment of a chase sequence*

$$\mathfrak{J}_0, \mathfrak{J}_1, \ldots, \mathfrak{J}_m, \quad \text{for } m \geq 0,$$

for $\mathfrak{D}$ and $\mathcal{O}$ such that $\mathfrak{J}_m \models q(\bar{a})$. Then, there exists a sequence

$$q_0, q_1, \ldots, q_m$$

of CQs such that $q_0 = q$, q_i is q_{i-1} or a σ_{m-i}-resolvent of q_{i-1}, for each $i \in [m]$, and $\mathfrak{J}_{m-i} \models q_i(\bar{a})$, for all $i = 0, 1, \ldots, m$.

Proof. By induction on the length of the the initial segment of the chase sequence at hand.

Base case. The statement holds trivially since $\mathfrak{J}_0 \models q(\bar{a})$. In other words, the desired sequence of CQs consists only of q.

Induction step. Assume that there exists a chase sequence $\mathfrak{J}_0, \mathfrak{J}_1, \ldots, \mathfrak{J}_{m+1}$ such that $\mathfrak{J}_{m+1} \models q(\bar{a})$. Suppose, for $i = 1, \ldots, m+1$, that $\mathfrak{J}_i$ results by the application of (h_i, σ_i).

Clearly, there exists a homomorphism h such that $h(\mathsf{body}(q)) \subseteq \mathfrak{J}_{m+1}$ and $h(\bar{x}) = \bar{a}$. Let H be the set of facts $h'_m(\mathsf{head}(\sigma_m))$, where h'_m is an extension of h_m that maps the existentially quantified variables of the head of σ_m to fresh nulls. It is clear that $H \subseteq \mathfrak{J}_{m+1}$. We proceed by considering two cases:

Case 1. Assume that $H \cap h(\mathsf{body}(q)) = \varnothing$. This implies that $\mathfrak{J}_m \models q(\bar{a})$. Thus, by induction hypothesis, there exists a sequence of CQs $q'_0, \ldots, q'_m$, where $q'_0 = q$, that enjoys the desired properties. Thus, the claim follows by the existence of the sequence of CQs $q, q, q'_1, \ldots, q'_m$.

Case 2. The interesting case is when $H \cap h(\mathsf{body}(q)) \neq \varnothing$. Let $S \subseteq \mathsf{body}(q)$ be a set of atoms such that $h(S) \subseteq H$, while $H \cap h(\mathsf{body}(q) \setminus S) = \varnothing$. In other words, S is the maximal subset of $\mathsf{body}(q)$ that is mapped to H via μ. Let S' be a set of atoms of $\mathsf{head}(\sigma_m)$ such that $h'_m(S') = h(S)$. It is easy to verify that (S, S', γ), where $\gamma := h'_m \cup h$ is a chunk unifier of q with σ_m – we assume, w.l.o.g., that σ_m and q do not share variables, and thus, γ is a well-defined substitution. Indeed, for every $x \in \mathsf{var}(S') \cap \mathsf{var}_{\exists}(\sigma_m)$, $\gamma(x)$

is not a constant since, by construction $h'_m(x)$ is a null, and $\gamma(x) = \gamma(y)$ implies that y occurs in S and is not shared. By contradiction, assume that y is shared, which means that it occurs in $\mathsf{body}(q) \setminus S$. Observe that, by definition of the set S, $h(\mathsf{body}(q) \setminus S) \subseteq \mathfrak{J}_m$, and thus, $h(y) = \gamma(y)$ is a null occurring in $\mathfrak{J}_m$, which is a contradiction, since $h(y)$ is fresh in H, and thus, it occurs only in $\mathsf{dom}(\mathfrak{J}_{m+1}) \setminus \mathsf{dom}(\mathfrak{J}_m)$. Since (S, S', γ) is a chunk unifier of q with σ_m, there exists a most general one, say $(S, S', \hat{\gamma})$. We define $\hat{q}$ in such a way that $\mathsf{body}(\hat{q}) = \hat{\gamma}((\mathsf{body}(q) \setminus S) \cup \mathsf{body}(\sigma_m))$ i.e., as a σ_m-resolvent of q, while its free variables are $\hat{\gamma}(\bar{x})$. We can show that $\mathfrak{J}_m \models \hat{q}(\bar{a})$, i.e., there exists a homomorphism h'' such that $h''(\mathsf{body}(\hat{q})) \subseteq \mathfrak{J}_m$ and $h''(\hat{\gamma}(\bar{x})) = \bar{a}$. By definition of the MGU, $\gamma = \theta \circ \hat{\gamma}$ for some substitution θ. It is clear that θ maps $\mathsf{body}(\hat{q})$ to $\mathfrak{J}_m$ since $\gamma(\mathsf{body}(\sigma_m)) \subseteq \mathfrak{J}_m$ and $\gamma(\mathsf{body}(q) \setminus S) \subseteq \mathfrak{J}_m$. Moreover, $\theta(\hat{\gamma}(\bar{x})) = \gamma(\bar{x}) = \bar{a}$. Thus, $\mathfrak{J}_m \models \hat{q}(\bar{a})$ as claimed above. By induction hypothesis, there exists a sequence of CQs $q'_0, \ldots, q'_m$, where $q'_0 = \hat{q}$, that enjoys the desired properties. Thus, the claim follows by taking the sequence of CQs $q, \hat{q}, q'_1, \ldots, q'_m$. □

We can now complete the proof of the statement (a) ⇒ (b). By hypothesis, $\bar{c} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$, and thus, there exists an initial segment $\mathfrak{J}_0, \mathfrak{J}_1, \ldots, \mathfrak{J}_m$ of a chase sequence for $\mathfrak{D}$ and $\mathcal{O}$ such that $\mathfrak{J}_m \models q(\bar{a})$. By Lemma B.9 there exists a sequence of CQs $q_0, q_1, \ldots, q_m$, where (i) $q_0 = q$, (ii) q_i is either q_{i-1} or a σ-resolvent of q_{i-1}, where $\sigma \in \mathcal{O}$, for each $i \in [m]$, and (iii) $\mathfrak{D} \models q_m(\bar{a})$. However, strictly speaking, $q_0, q_1, \ldots, q_m$ is not a branch of the unfolding of q with $\mathcal{O}$ due to the fact that some CQs are repeated. Indeed, there may be an $i \in [m]$ such that q_i is not a resolvent of q_{i-1} but is rather identical to q_{i-1}. We can easily convert $q_0, q_1, \ldots, q_m$ into a proper branch of the unfolding of q with $\mathcal{O}$ of length $n \leq m$ by simply removing the repeated CQs from $q_0, q_1, \ldots, q_m$.

Proof of (b) ⇒ (a)

We first establish the following auxiliary lemma:

Lemma B.10. *Consider a branch $q_0, q_1, \ldots, q_n$, for $n \geq 0$, of the unfolding of q with $\mathcal{O}$. For every $i = 0, 1, \ldots, n$, $\bar{a} \in \mathsf{cert}_{q_i,\mathcal{O}}(\mathfrak{D})$ implies $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$.*

Proof. We proceed by induction on $i \geq 0$.

Base case. Clearly, $\bar{a} \in \mathsf{cert}_{q_0,\mathcal{O}}(\mathfrak{D})$ implies $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$ holds trivially since, by definition, $q_0 = q$.

Inductive step. Suppose now that $\bar{a} \in \mathsf{cert}_{q_i,\mathcal{O}}(\mathfrak{D})$, for $i > 0$. To show that $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$, by induction hypothesis, it suffices to show that $\bar{a} \in \mathsf{cert}_{q_{i-1},\mathcal{O}}(\mathfrak{D})$, i.e., there exists a homomorphism h that maps $\mathsf{body}(q_{i-1})$ to $\mathsf{chase}(\mathfrak{D}, \mathcal{O})$ such that $h(\bar{x}_{i-1}) = \bar{a}$, where $\bar{x}_{i-1}$ are the answer variables of q_{i-1}.

Since, by hypothesis, $\bar{a} \in \mathsf{cert}_{q_i,\mathcal{O}}(\mathfrak{D})$, we conclude that there exists a homomorphism h' such that $h'(\mathsf{body}(q_i)) \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$ and $h'(\bar{x}_i) = \bar{a}$ with $\bar{x}_i$ being the output variables of q_i. Recall that q_i is a σ-resolvent of q_{i-1} for some $\sigma \in \mathcal{O}$, i.e., q_i is such that $\mathsf{body}(q_i) = \gamma((\mathsf{body}(q_{i-1}) \setminus S) \cup \mathsf{body}(\sigma))$, while its free variables are $\gamma(\bar{x}_{i-1})$, for an MGCU (S, S', γ) of q_{i-1} with σ. Let $\mu \coloneqq h' \circ \gamma$. Observe that $\mu(\mathsf{body}(\sigma)) \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$. Thus, $\mu'(\mathsf{head}(\sigma)) \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$, where $\mu' \supseteq \mu$ is an appropriate homomorphism that

maps each existentially quantified variable of the head of σ to a fresh null. We define the substitution

$$h'' \coloneqq h' \restriction \mathsf{var}(q_i) \cup \{\gamma(z) \mapsto \mu'(z)\}_{z \in \mathsf{var}(S') \cap \mathsf{var}_\exists(\sigma)}$$

We proceed to show that

- h'' is a well-defined substitution.
- The desired homomorphism h such that $h(\mathsf{body}(q_{i-1})) \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$, as well as $h(\bar{x}_{i-1}) = \bar{a}$, is $h' \circ \gamma$.

To show that h'' is a well-defined substitution, it suffices to show that, for each $z \in \mathsf{var}(S') \cap \mathsf{var}_\exists(\sigma)$, $\gamma(z)$ is not a constant, and $\gamma(z)$ does not occur in the domain of $h' \restriction \mathsf{var}(q_i)$. By contradiction, assume that $\gamma(z)$ is either a constant, or is in the domain of h'. It is easy to verify that in this case there exists a $z \in \mathsf{var}(S') \cap \mathsf{var}_\exists(\sigma)$ such that $\gamma(z)$ is a constant, or $\gamma(z) = \gamma(y)$ for a variable y that is in S', or in S but shared. This contradicts the fact that (S, S', γ) is a chunk unifier.

We proceed to show that the desired homomorphism h is $h'' \circ \gamma$. Clearly, it holds that $h''(\gamma(\mathsf{body}(q_{i-1}) \setminus S)) \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$. Moreover, $h''(\gamma(S)) = h''(\gamma(S')) = \mu'(S') \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$, since all atoms in S' occur in $\mathsf{head}(\sigma)$ and $\mu'(\mathsf{head}(\sigma)) \subseteq \mathsf{chase}(\mathfrak{D}, \mathcal{O})$. Finally, since $\gamma(\bar{x}_{i-1}) = \bar{x}_i$ and $h'(\bar{x}_i) = \bar{a}$, we get that $h''(\gamma(\bar{x}_{i-1})) = \bar{a}$, and the claim follows. □

Having the Lemma B.10 in place, we can now show that (b) ⇒ (a). By hypothesis, there exists a branch $q_0, q_1, \ldots, q_n$, for some $n \geq 0$, of the unfolding of q with $\mathcal{O}$ such that $\mathfrak{D} \models q_n(\bar{a})$. Thus, $\bar{a} \in \mathsf{cert}_{q_n,\mathcal{O}}(\mathfrak{D})$ due to the monotonicity of CQs. By Lemma B.10 we get that $\bar{a} \in \mathsf{cert}_{q,\mathcal{O}}(\mathfrak{D})$, and the claim follows. □

B.4.2 Proof of Lemma 6.15

For a sequence of variables $\bar{v}$ that are all among $\mathsf{var}(q)$, we let $\sim_{h,\bar{v}}$ be the equivalence relation defined by

$$v_i \sim_{h,\bar{v}} v_j \iff h(v_i) = h(v_j),$$

and we let $\pi_{h,\bar{v}}$ be the partition of the variables $\bar{v}$ given by the set of equivalence classes of $\sim_{h,\bar{v}}$.

We show the following lemma which is a slightly more general statement than that of Lemma 6.15:

Lemma B.11. *If there is a (linear) chase tree $\mathcal{C}$ for $h(\mathsf{body}(q))$ w.r.t. $\mathcal{G}_\Theta^{\mathfrak{D},\mathcal{O}}$ such that $\mathsf{nwd}(\mathcal{T}) \leq m$, then there is a (linear) proof tree $\mathcal{P}$ of $q(\bar{x})$ w.r.t. $\mathcal{O}$ such that*

(i) *$\mathcal{P}$ has equality type $\pi_{h,\bar{x}}$,*

(ii) *$\mathsf{nwd}(\mathcal{P}) \leq m$, and*

(iii) *$\mathfrak{D} \models q_\mathcal{P}(\bar{a})$.*

PROOF. Let $\alpha_1, \ldots, \alpha_s$ be an enumeration of the body atoms of $q(\bar{x})$. We proceed by induction on the depth of $\mathcal{C}$ (i.e., the longest among all branches).

Suppose first that the depth of $\mathcal{C}$ equals 1, that is, $\mathcal{T}$ consists of just a single node v_0 whose label equals $h(\mathsf{body}(q))$. Then the proof tree $\mathcal{P}$ that has equality type $\pi_{h,x}$ and that just consists of a single node labeled with

$$q(\mathsf{eq}_{\pi_{h,\bar{x}}}(x_1, \ldots, x_n)) \leftarrow \mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_1), \ldots, \mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_s),$$

is obviously a proof tree for $q(\bar{x})$ w.r.t. $\mathcal{O}$ such that $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$.

Suppose now that the depth of $\mathcal{C}$ is larger than one such that $\mathsf{nwd}(\mathcal{C}) \leq m$. We distinguish cases.

Case 1. Suppose first that the children of the root v_0 of $\mathcal{C}$, whose label equals $h(\mathsf{body}(p))$, result from a decomposition step. Assume that v_0 has exactly two children, say v_1 and v_2, that are respectively labeled Θ_1 and Θ_2 – the case with any other number of children is treated analogously. Therefore, $\{\Theta_1, \Theta_2\}$ must be a decomposition of $h(\mathsf{body}(p))$ such that $h(\mathsf{body}(p)) = \Theta_1 \cup \Theta_2$, and Θ_1 and Θ_2 do not share any labeled null. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively be the subtrees rooted in v_1 and v_2, and notice that $\mathsf{nwd}(\mathcal{C}_1) \leq m$ and $\mathsf{nwd}(\mathcal{C}_2) \leq m$. Moreover, let $\bar{y} = y_1, \ldots, y_k$ enumerate all variables from $\mathsf{var}(q) \setminus \{x_1, \ldots, x_n\}$ such that (i) $h(y_i)$ is a constant, and (ii) y_i occurs in atoms α and β of $q(\bar{x})$, but neither $h(\{\alpha, \beta\}) \subseteq \Theta_1$, nor $h(\{\alpha, \beta\}) \subseteq \Theta_2$ holds.

Let $\mathcal{P}$ be the proof tree with equality type $\pi_{h,\bar{x}}$ whose root v_0 is labeled with

$$q(\mathsf{eq}_{\pi_{h,\bar{x}}}(\bar{x})) \leftarrow \mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_1), \ldots, \mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_s), \tag{B.2}$$

and that has exactly one child v' whose label is

$$q'(\mathsf{eq}_{\pi_{h,\bar{x},\bar{y}}}(\bar{x}, \bar{y})) \leftarrow \mathsf{eq}_{\pi_{h,\bar{x},\bar{y}}}(\alpha_1), \ldots, \mathsf{eq}_{\pi_{h,\bar{x},\bar{y}}}(\alpha_s). \tag{B.3}$$

It is easy to check that (B.3) results from a specialization step from (B.2). Now let $\beta_{i_1}, \ldots, \beta_{i_l}$ enumerate those atoms from $\mathsf{eq}_{\pi_{h,\bar{x},\bar{y}}}(\alpha_1), \ldots, \mathsf{eq}_{\pi_{h,\bar{x},\bar{y}}}(\alpha_s)$ whose image under h lies in Θ_1, and $\beta_{j_1}, \ldots, \beta_{j_r}$ those whose image under h lies in Θ_2. Let $\bar{z}$ be the restriction of $\mathsf{eq}_{\pi_{h,\bar{x},\bar{y}}}(\bar{x}, \bar{y})$ to $\mathsf{var}(\{\beta_{i_1}, \ldots, \beta_{i_l}\})$, and let $\bar{w}$ be the restriction of $\mathsf{eq}_{\pi_{h,\bar{x},\bar{y}}}(\bar{x}, \bar{y})$ to $\mathsf{var}(\{\beta_{j_1}, \ldots, \beta_{j_r}\})$. We let v_1 and v_2 be children of v' in $\mathcal{P}$ whose labels are respectively $q_1(\bar{z}) \leftarrow \beta_{i_1}, \ldots, \beta_{i_l}$ and $q_2(\bar{w}) \leftarrow \beta_{j_1}, \ldots, \beta_{j_r}$. Then these two queries result from a decomposition step from (B.3). Since $h(\{\beta_{i_1}, \ldots, \beta_{i_l}\}) = \Theta_1$ and $h(\{\beta_{j_1}, \ldots, \beta_{j_r}\}) = \Theta_2$, by induction hypothesis, there are proof trees $\mathcal{P}_1$ and $\mathcal{P}_2$ that have equality types $\pi_{h,\bar{z}}$ and $\pi_{h,\bar{w}}$, respectively, such that $\mathsf{nwd}(\mathcal{P}_1) \leq m$ and $\mathsf{nwd}(\mathcal{P}_2) \leq m$. Furthermore, we have $\mathfrak{D} \models q_{\mathcal{P}_1}(h(\bar{z}))$ and $\mathfrak{D} \models q_{\mathcal{P}_2}(h(\bar{w}))$. Notice that $\mathsf{eq}_{\pi_{h,\bar{z}}}(\bar{z}) = \bar{z}$ and $\mathsf{eq}_{\pi_{h,\bar{w}}}(\bar{w}) = \bar{w}$ by construction. Moreover, $\exists \bar{v}\, (q_{\mathcal{P}_1}(\bar{z}) \wedge q_{\mathcal{P}_2}(\bar{w})) \equiv q_{\mathcal{P}}$, where $\bar{v}$ is the sequence of variables that appear in the head of (B.3), but not in the head of (B.2). Hence, $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$, and $\mathcal{P}$ is thus the proof tree of $q(\bar{x})$ w.r.t. $\mathcal{O}$ we are looking for.

Case 2. Suppose now that the root v_0 of $\mathcal{C}$ has exactly one child, say v', that is labeled with Θ' which results from $h(\mathsf{body}(q))$ by unfolding. Let $\tau \in \mathcal{O}$, h_0, $\beta_1, \ldots, \beta_k$, and $\alpha \in h(\mathsf{body}(q))$ be such that $\beta_1, \ldots, \beta_k \Rightarrow_{\tau,h_0} \alpha$ and $\Theta' = (h(\mathsf{body}(q)) \setminus \{\alpha\}) \cup \{\beta_1, \ldots, \beta_k\}$. Thus, τ is of the form

$$R_{\beta_1}(\bar{x}_1), \ldots, R_{\beta_k}(\bar{x}_k) \rightarrow \exists w_{i_1}, \ldots, w_{i_l}\, R_\alpha(w_1, \ldots, w_r),$$

for some predicates $R_{\beta_1}, \ldots, R_{\beta_k}, R_\alpha$. Moreover, $q(\bar{x})$ contains an atom of the form $R_\alpha(t_1, \ldots, t_r)$ such that

$$h(R_\alpha(t_1, \ldots, t_r)) = \alpha = h_0(R_\alpha(w_1, \ldots, w_r))$$

(the $t_1, \ldots, t_r$ are terms each of which is either a variable or a constant). Now let τ_{v_0} be a copy of τ, where every variable occurrence x is renamed to x_{v_0}. Let h' be the homomorphism defined by $h' \colon x_{v_0} \mapsto h_0(x)$. Moreover, let $\mathsf{eq}_{\pi_{h,\bar{x}}}(t_1, \ldots, t_r) = s_1, \ldots, s_r$. Notice that $\{t_1, \ldots, t_r\} \subseteq \{s_1, \ldots, s_r\}$ and observe that $h(R_\alpha(s_1, \ldots, s_r)) = \alpha = h'(R_\alpha(w_{1,v_0}, \ldots, w_{r,v_0}))$. Let γ be a substitution such that, for all $i, j \in [r]$,

$$\gamma(z_i) = \gamma(w_{j,v_0}) \coloneqq v_t \iff t = h(z_i) = h'(w_{j,v_0}),$$

where the v_t are newly chosen variable names (for the other variables not mentioned, γ is simply the identity). In particular, $\gamma(x_i) = \gamma(x_j)$ iff $x_i \sim_{h,\bar{x}} x_j$, for all answer variables x_i and x_j of $q(\bar{x})$ that are among $\{s_1, \ldots, s_r\}$. Let γ_0 be an MGU such that $\gamma = \eta \circ \gamma_0$ for some substitution η. Notice that, if x_i and x_j are answer variables of $q(\bar{x})$ among $\{s_1, \ldots, s_r\}$, then $\gamma_0(x_i) = \gamma_0(x_j)$ implies $x_i \sim_{h,\bar{x}} x_j$. On the other hand, for $\hat{x}_i \coloneqq \mathsf{eq}_{\pi_{h,\bar{x}}}(x_i)$ $(i = 1, \ldots, n)$, if $x_i \sim_{h,\bar{x}} x_j$, then there is exactly one $k \in [n]$ such that $h(\hat{x}_k) = h(x_i) = h(x_j)$. Thus, γ_0 is bijective when restricted to the (representatives of the) equivalence classes of $\sim_{h,\bar{x}}$ and we can henceforth assume w.l.o.g. that $\gamma_0(\hat{x}_i) = \hat{x}_i$ for all $i = 1, \ldots, n$.

Let $\mathcal{P}$ be the proof tree with equality type $\pi_{h,\bar{x}}$ whose root v_0 is labeled with

$$q(\hat{x}_1, \ldots, \hat{x}_n) \leftarrow \mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_1), \ldots, \mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_s). \tag{B.4}$$

We introduce a new node v' in $\mathcal{P}$ that is a child of v_0 and whose label is

$$q'(\hat{x}_1, \ldots, \hat{x}_n) \leftarrow \gamma_0(A), \tag{B.5}$$

where

$$A \coloneqq (\mathsf{eq}_{\pi_{h,\bar{x}}}(\{\alpha_1, \ldots, \alpha_s\}) \setminus \{R_\alpha(s_1, \ldots, s_r)\}) \cup \{R_{\beta_1}(\bar{x}_{1,v_0}), \ldots, R_{\beta_k}(\bar{x}_{k,v_0})\}.$$

It is clear that (B.5) is a σ_{v_0}-resolvent of (B.4). Notice in particular that the variables occurring in $\mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_1), \ldots, \mathsf{eq}_{\pi_{h,\bar{x}}}(\alpha_s)$ that unify with some existential variable from the head of τ_{v_0} cannot be shared, since the application of $\beta_1, \ldots, \beta_k \Rightarrow_{\sigma,h_0} \alpha$ is not blocked in $h(\mathsf{body}(p))$. Moreover, the resolvent must be an IDO-resolvent, since γ_0 is the identity on $\{\hat{x}_1, \ldots, \hat{x}_n\}$.

Let us write $q'(\hat{x}_1, \ldots, \hat{x}_n)$ for the CQ (B.5). Now let h'' be the homomorphism that extends h so that h'' maps q' to Θ' and $h''(\hat{x}_1, \ldots, \hat{x}_n) = \bar{a}$. Notice that h'' exists by construction. Now the subtree of $\mathcal{C}$ that is rooted at v', call it $\mathcal{C}'$, has smaller depth than $\mathcal{C}$, whence by induction hypothesis it follows that there is a proof tree $\mathcal{P}'$ of q' w.r.t. $\mathcal{O}$ such that (i) $\mathcal{P}'$ has equality type $\pi_{h'',\hat{x}_1,\ldots,\hat{x}_n}$, (ii) $\mathsf{nwd}(\mathcal{P}') \leq m$, and (iii) $\mathfrak{D} \models q_{\mathcal{P}'}(\bar{a})$. We can thus simply declare that $\mathcal{P}'$ becomes a subtree of $\mathcal{P}$ rooted at the node v' of $\mathcal{P}$. Then $\mathcal{P}$ is a proof tree for $q(\bar{x})$ that has equality type $\pi_{h,\bar{x}}$ such that $\mathsf{nwd}(\mathcal{P}) \leq m$. Moreover, we must have $\mathfrak{D} \models q_{\mathcal{P}}(\bar{a})$, since $\mathcal{P}$ and $\mathcal{P}'$ have the same leaf nodes.

Notice that the constructions performed in the all cases above yields a linear $\mathcal{P}$ whenever $\mathcal{T}$ is linear, and thus Lemma B.11 follows. □

List of Theorems

Index

Bibliography

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

[2] Foto N. Afrati, Manolis Gergatsoulis, and Francesca Toni. Linearisability on datalog programs. *Theor. Comput. Sci.*, 308(1-3):199–226, 2003.

[3] Miklós Ajtai and Yuri Gurevich. Datalog vs First-Order Logic. *J. Comput. Syst. Sci.*, 49(3):562–588, 1994.

[4] Antoine Amarilli, Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. Query Answering with Transitive and Linear-Ordered Data. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI '16, pages 893–899, 2016.

[5] Hajnal Andréka, István Németi, and Johan van Benthem. Modal Languages and Bounded Fragments of Predicate Logic. *J. Philosophical Logic*, 27(3):217–274, 1998.

[6] Marcelo Arenas, Richard Hull, Wim Martens, Tova Milo, and Thomas Schwentick. Foundations of Data Management (Dagstuhl Perspectives Workshop 16151). *Dagstuhl Reports*, 6(4):39–56, 2016.

[7] Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The iBench Integration Metadata Generator. *PVLDB*, 9(3):108–119, 2015.

[8] Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

[9] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[10] Franz Baader. Least Common Subsumers and Most Specific Concepts in a Description Logic with Existential Restrictions and Terminological Cycles. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, IJCAI '03, pages 319–324, 2003.

[11] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL Envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, IJCAI '05, pages 364–369, 2005.

[12] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[13] Jean-François Baget. Improving the forward chaining algorithm for conceptual graphs rules. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference*, KR '04, pages 407–414, 2004.

[14] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[15] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, IJCAI '11, pages 712–717, 2011.

[16] Jean-François Baget, Marie-Laure Mugnier, and Michaël Thomazo. Towards Farsighted Dependencies for Existential Rules. In *Proceedings of the Web Reasoning and Rule Systems - 5th International Conference*, RR '11, pages 30–45, 2011.

[17] Vince Bárány, Michael Benedikt, and Balder ten Cate. Rewriting Guarded Negation Queries. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium*, MFCS '13, pages 98–110, 2013.

[18] Vince Bárány, Georg Gottlob, and Martin Otto. Querying the Guarded Fragment. *Logical Methods in Computer Science*, 10(2), 2014.

[19] Vince Bárány, Balder ten Cate, and Martin Otto. Queries with Guarded Negation. *PVLDB*, 5(11):1328–1339, 2012.

[20] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded Negation. *J. ACM*, 62(3):22:1–22:26, 2015.

[21] Pablo Barceló, Gerald Berger, Carsten Lutz, and Andreas Pieris. First-Order Rewritability of Frontier-Guarded Ontology-Mediated Queries. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI '18, pages 1707–1713, 2018.

[22] Pablo Barceló, Gerald Berger, and Andreas Pieris. Containment for Rule-Based Ontology-Mediated Queries. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '18, pages 267–279, 2018.

[23] Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. Does query evaluation tractability help query containment? In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 188–199, 2014.

[24] Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In *Proc. of the ICALP*, pages 73–85, 1981.

[25] Catriel Beeri and Moshe Y. Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.

[26] Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. Swift Logic for Big Data and Knowledge Graphs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI '17, pages 2–10, 2017.

[27] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *PVLDB*, 11(9):975–987, 2018.

[28] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A Step Up in Expressiveness of Decidable Fixpoint Logics. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 817–826, 2016.

[29] Michael Benedikt and Georg Gottlob. The Impact of Virtual Views on Containment. *PVLDB*, 3(1):297–308, 2010.

[30] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, pages 37–52, 2017.

[31] Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. The Complexity of Boundedness for Guarded Logics. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '15, pages 293–304, 2015.

[32] Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The Space-Efficient Core of Vadalog, 2019. To appear in PODS '19.

[33] Gerald Berger and Andreas Pieris. Ontology-Mediated Queries Distributing over Components. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI '16, pages 943–949, 2016.

[34] Meghyn Bienvenu, Peter Hansen, Carsten Lutz, and Frank Wolter. First Order-Rewritability and Containment of Conjunctive Queries in Horn Description Logics. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI '16, pages 965–971, 2016.

[35] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. Query Containment in Description Logics Reconsidered. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference*, KR '12, 2012.

[36] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-Order Rewritability of Atomic Queries in Horn Description Logics. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 754–760, 2013.

[37] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-Based Data Access: A Study through Disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[38] Achim Blumensath, Martin Otto, and Mark Weyer. Decidability Results for the Boundedness Problem. *Logical Methods in Computer Science*, 10(3), 2014.

[39] Peter Van Emde Boas. The Convenience of Tilings. In *Complexity, Logic, and Recursion Theory*, pages 331–363. Marcel Dekker Inc, 1997.

[40] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable Highly Expressive Query Languages - IJCAI-15 Distinguished Paper (Honorary Mention). In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '15, pages 2826–2832, 2015.

[41] Pierre Bourhis, Marco Manna, Michael Morak, and Andreas Pieris. Guarded-Based Disjunctive Tuple-Generating Dependencies. *ACM Trans. Database Syst.*, 41(4):27:1–27:45, 2016.

[42] Pierre Bourhis, Michael Morak, and Andreas Pieris. Acyclic Query Answering under Guarded Disjunctive Existential Rules and Consequences to DLs. In *Informal Proceedings of the 27th International Workshop on Description Logics*, pages 100–111, 2014.

[43] Luca Cabibbo. The Expressive Power of Stratified Logic Programs with Value Invention. *Inf. Comput.*, 147(1):22–56, 1998.

[44] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

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

[46] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*, LICS '10, pages 228–242, 2010.

[47] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[48] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[49] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Decidable containment of recursive queries. *Theor. Comput. Sci.*, 336(1):33–56, 2005.

[50] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Trans. Knowl. Data Eng.*, 1(1):146–166, 1989.

[51] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Surveys in computer science. Springer, 1990.

[52] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[53] Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. Embedded Implicational Dependencies and their Inference Problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '81, pages 342–354, 1981.

[54] Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, 1977.

[55] Surajit Chaudhuri and Moshe Y. Vardi. On the Equivalence of Recursive and Nonrecursive Datalog Programs. *J. Comput. Syst. Sci.*, 54(1):61–78, 1997.

[56] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

[57] Thomas Colcombet. A Safra-like construction for regular cost functions over finite words.

[58] Thomas Colcombet. The Theory of Stabilisation Monoids and Regular Cost Functions. In *Automata, Languages and Programming, 36th Internatilonal Colloquium*, ICALP '09, pages 139–150, 2009.

[59] Thomas Colcombet and Nathanaël Fijalkow. The Bridge Between Regular Cost Functions and Omega-Regular Languages. In *43rd International Colloquium on Automata, Languages, and Programming*, ICALP '16, pages 126:1–126:13, 2016.

[60] Thomas Colcombet and Christof Löding. Regular Cost Functions over Finite Trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*, LICS '10, pages 70–79, 2010.

[61] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[62] Stavros Cosmadakis, Haim Gaifman, Paris Kanellakis, and Moshe Vardi. Decidable Optimization Problems for Database Logic Programs. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 477–490, 1988.

[63] Stavros S. Cosmadakis and Paris C. Kanellakis. Functional and Inclusion Dependencies. *Advances in Computing Research*, 3:163–184, 1986.

[64] Oliver Costich. A Medvedev Characterization of Sets Recognized by Generalized Finite Automata. *Mathematical Systems Theory*, 6(3):263–267, 1972.

[65] Bruno Courcelle. The Monadic Second-Order Logic of Graphs, II: Infinite Graphs of Bounded Width. *Mathematical Systems Theory*, 21(4):187–221, 1989.

[66] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[67] Evgeny Dantsin and Andrei Voronkov. Complexity of Query Answering in Logic Databases with Complex Values. In *Logical Foundations of Computer Science, 4th International Symposium*, LFCS '97, pages 56–66, 1997.

[68] Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability Is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018.

[69] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '08, pages 149–158, 2008.

[70] John Doner. Tree Acceptors and Some of Their Applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[71] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

[72] Thomas Eiter, Thomas Lukasiewicz, and Livia Predoiu. Generalized Consistent Query Answering under Existential Rules. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference*, KR '16, pages 359–368, 2016.

[73] E. Allen Emerson and Charanjit S. Jutla. Tree Automata, Mu-Calculus and Determinacy (Extended Abstract). In *32nd Annual Symposium on Foundations of Computer Science*, FOCS '91, pages 368–377, 1991.

[74] Ronald Fagin. A Normal Form for Relational Databases That Is Based on Domians and Keys. *ACM Trans. Database Syst.*, 6(3):387–415, 1981.

[75] Ronald Fagin. Inverting schema mappings. *ACM Trans. Database Syst.*, 32(4):25, 2007.

[76] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[77] Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[78] Haim Gaifman, Harry G. Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable Optimization Problems for Database Logic Programs. *J. ACM*, 40(3):683–713, 1993.

[79] Georg Gottlob, Erich Grädel, and Helmut Veith. Datalog LITE: a deductive query language with linear time model checking. *ACM Trans. Comput. Log.*, 3(1):42–79, 2002.

[80] Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artif. Intell.*, 213:42–59, 2014.

[81] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003.

[82] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Query Rewriting and Optimization for Ontological Databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014.

[83] Georg Gottlob and Christos H. Papadimitriou. On the complexity of single-rule datalog queries. *Inf. Comput.*, 183(1):104–122, 2003.

[84] Georg Gottlob and Andreas Pieris. Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '15, pages 2999–3007, 2015.

[85] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. Expressiveness of guarded existential rule languages. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 27–38, 2014.

[86] Erich Grädel. On The Restraining Power of Guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.

[87] Erich Grädel. Guarded fixed point logics and the monadic theory of countable trees. *Theor. Comput. Sci.*, 288(1):129–152, 2002.

[88] Erich Grädel and Martin Otto. The Freedoms of (Guarded) Bisimulation. In *Johan van Benthem on Logic and Information Dynamics*, pages 3–31. Springer, 2014.

[89] Erich Grädel and Igor Walukiewicz. Guarded Fixed Point Logic. In *14th Annual IEEE Symposium on Logic in Computer Science*, LICS '99, pages 45–54, 1999.

[90] Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *J. Artif. Intell. Res.*, 47:741–808, 2013.

[91] Peter Hansen and Carsten Lutz. Computing FO-Rewritings in EL in Practice: from Atomic to Conjunctive Queries. In *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017.*, 2017.

[92] Lane A. Hemachandra. The Strong Exponential Hierarchy Collapses. *J. Comput. Syst. Sci.*, 39(3):299–322, 1989.

[93] Neil Immerman. Expressibility as a complexity measure: results and directions. In *Proceedings of the 2nd Annual Conference on Structure in Complexity Theory*, 1987.

[94] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[95] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. Efficiently answering reachability queries on very large directed graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 595–608, 2008.

[96] David S. Johnson. A Catalog of Complexity Classes. In *Handbook of Theoretical Computer Science*, pages 67–161. MIT Press Cambridge, MA, USA, 1990.

[97] David S. Johnson and Anthony C. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[98] Valerie King. Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. In *40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 81–91, 1999.

[99] Mélanie König, Michel Leclère, Marie-Laure Mugnier, and Michaël Thomazo. Sound, complete and minimal UCQ-rewriting for existential rules. *Semantic Web*, 6(5):451–475, 2015.

[100] Markus Krötzsch and Sebastian Rudolph. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, IJCAI '11, pages 963–968, 2011.

[101] Michel Leclère, Marie-Laure Mugnier, Michaël Thomazo, and Federico Ulliana. A Single Approach to Decide Chase Termination on Linear Existential Rules. In *22nd International Conference on Database Theory*, ICDT '19, pages 18:1–18:19, 2019.

[102] Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[103] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. From Classical to Consistent Query Answering under Existential Rules. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI '15, pages 1546–1552, 2015.

[104] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[105] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 13–22, 2009.

[106] Donald A. Martin. Borel Determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

[107] Michael Meier, Michael Schmidt, and Georg Lausen. On Chase Termination Beyond Stratification. *PVLDB*, 2(1):970–981, 2009.

[108] Michael Morak. *The impact of disjunction on reasoning under existential rules.* PhD thesis, University of Oxford, 2014.

[109] Andrzej Wlodzimierz Mostowski. Hierarchies of Weak Automata and Weak Monadic Formulas. *Theor. Comput. Sci.*, 83(2):323–335, 1991.

[110] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54:267–276, 1987.

[111] Jeffrey F. Naughton. Data Independent Recursion in Deductive Databases. In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '86, pages 267–279, 1986.

[112] Jeffrey F. Naughton. Data Independent Recursion in Deductive Databases. *J. Comput. Syst. Sci.*, 38(2):259–289, 1989.

[113] Jeffrey F. Naughton and Yehoshua Sagiv. A decidable class of bounded recursions. In *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '87, pages 227–236, 1987.

[114] Adrian Onet. *The chase procedure and its applications*. PhD thesis, Concordia University, Canada, 2012.

[115] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[116] Sushant Patnaik and Neil Immerman. Dyn-FO: A Parallel, Dynamic Complexity Class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997.

[117] Andreas Pieris. *Ontological Query Answering: New Languages, Algorithms and Complexity*. PhD thesis, University of Oxford, 2011.

[118] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking Data to Ontologies. *J. Data Semantics*, 10:133–173, 2008.

[119] Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, Massachusetts, 1967.

[120] Michael O. Rabin and Dana S. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[121] Raymond Reiter. Deductive Question-Answering on Relational Data Bases. In *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977.*, pages 149–177, 1977.

[122] Svan Rocher. *Querying Existential Rule Knowledge Bases: Decidability and Complexity*. PhD thesis, University of Montpellier, France, 2016.

[123] Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008.

[124] Sebastian Rudolph and Michaël Thomazo. Characterization of the Expressivity of Existential Rule Queries. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, IJCAI '15, pages 3193–3199, 2015.

[125] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences Among Relational Expressions with the Union and Difference Operators. *J. ACM*, 27(4):633–655, 1980.

[126] Oded Shmueli. Equivalence of DATALOG Queries is Undecidable. *J. Log. Program.*, 15(3):231–241, 1993.

[127] Mantas Simkus. *Nonmonotonic Logic Programs with Function Symbols*. PhD thesis, Technische Universität Wien, 2010.

[128] Giora Slutzki. Alternating Tree Automata. *Theor. Comput. Sci.*, 41:305–318, 1985.

[129] Johan van Benthem. *Modal correspondence theory*. PhD thesis, University of Amsterdam, 1976.

[130] Moshe Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, STOC '82, pages 137–146, 1982.

[131] Moshe Y. Vardi. Automata Theory for Database Theoreticans. In *Theoretical Studies in Computer Science, to Seymour Ginsburg on the occasion of his 2^6. birthday*, pages 153–180, 1992.

[132] Moshe Y. Vardi. Why is Modal Logic So Robustly Decidable? In *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996*, pages 149–184, 1996.

[133] Moshe Y. Vardi. Reasoning about The Past with Two-Way Automata. In *Automata, Languages and Programming, 25th International Colloquium*, ICALP '98, pages 628–641, 1998.

[134] Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Belg. Math. Soc. Simon Stevin*, 8(2):359–391, 2001.

[135] Mihalis Yannakakis. Algorithms for Acyclic Database Schemes. In *Very Large Data Bases, 7th International Conference*, VLDB '81, pages 82–94, 1981.

Gerald Berger

Curriculum Vitae

Personal Information

Birth date May 21, 1990 in Wels, Austria
Citizenship Austria
Languages German (native tongue), English (fluent)

Education

from April 2015 **PhD studies in Computer Science**, *TU Wien*.
Thesis Static Analysis for Ontology-Mediated Querying
Adviser Prof. Georg Gottlob
Student in the doctoral college *Logical Methods in Computer Science* (LogiCS).

2013–2015 **Master studies in Computational Intelligence**, *TU Wien*.
Graduation with highest distinction.
Thesis Provability Interpretations of a Many-Sorted Polymodal Logic.
Advisers Prof. Hans Tompits
Prof. Lev D. Beklemishev (Steklov Mathematical Institute, Moskow)

2010–2013 **Bachelor studies in Software & Information Engineering**, *TU Wien*.
Graduation with highest distinction.
Thesis On Axiomatic Rejection for the Description Logic $\mathcal{ALC}$.
Adviser Prof. Hans Tompits

2004–2009 **High school diploma**, *HTL Grieskirchen*, Austria.
Graduation ("Matura") with highest distinction.
Specialization IT and Business.

Professional Activities

Sept 2016–July 2018 **External Lecturer**, *University of Vienna*.

from Sept 2016 **DOC fellowship holder**, *Austrian Academy of Sciences*.

Abelegasse 26/2/11 – 1160 Wien, Austria
+43 660 123 4899 • gberger@dbai.tuwien.ac.at

Nov 2015–Sept 2016 **University Assistant**, *Institute for Information Systems, TU Wien*.

Jun 2015 to Nov 2015 **Senior Lecturer**, *Faculty of Informatics, TU Wien*.

Teaching Duties

Sept 2016–July 2018 **External Lecturer**, *University of Vienna*.
- Course *Theoretical Computer Science*.

Jun 2015–Sept 2016 **Senior Lecturer and University Assistant**, *TU Wien*.
- Courses *Object-oriented Modeling*, *Formal Modeling*, and *Data Modeling*.

2011–2015 **Tutor and Teaching Assistant**, *TU Wien*.
- Numerous teaching activities for undergraduate courses of computer science curricula.

Reviews for Conferences

- 30th International Conference on Logic Programming (ICLP'14)
- 13th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)
- 21st European Conference on Artificial Intelligence (ECAI'14)
- 23rd Workshop on Functional and Constraint Logic Programming (WFLP'14)
- 25th International Joint Conference on Artificial Intelligence (IJCAI'16)
- 44th International Colloquium on Automata, Languages and Programming (ICALP'17)
- 27th International Conference on Automated Deduction (CADE'19)

Prizes and Grants

- PhD fellowship from the Austrian Academy of Sciences, 2016. Total amount of funding: € 111k. Duration: three years.
- Prize for outstanding studies by the Austrian Federal Ministry for Science, Research, and Economics ("Staatspreis"), 2015.
- Scholarships from the Faculty of Informatics of TU Wien in the years 2011 to 2014.
- Scholarships from TU Wien in the years 2011 to 2013.

Miscellaneous

- Certificates:
  - Oracle Certified Associate: PL/SQL-Developer
  - CISCO CCNA Exploration: Network Fundamentals
  - CISCO CCNA Exploration: LAN Switching and Wireless
  - CISCO CCNA Exploration: Routing Protocols and Concepts
- Student speaker at the doctoral program LogiCS from 2016 to 2017.

Publications

Pablo Barceló, Gerald Berger, Carsten Lutz, and Andreas Pieris. First-Order Rewritability of Frontier-Guarded Ontology-Mediated Queries. In *IJCAI*, pages 1707–1713, 2018.

Pablo Barceló, Gerald Berger, and Andreas Pieris. Containment for Rule-Based Ontology-Mediated Queries. In *PODS*, pages 267–279, 2018.

Gerald Berger, Lev D. Beklemishev, and Hans Tompits. A many-sorted variant of Japaridze's polymodal provability logic. *Logic Journal of the IGPL*, 26(5):505–538, 2018.

Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The Space-Efficient Core of Vadalog. In *PODS*, 2019. To appear.

Gerald Berger, Martin Otto, Andreas Pieris, Dimitri Surinx, and Jan Van den Bussche. Additive first-order queries. In *ICDT*, pages 19:1–19:14, 2019.

Gerald Berger and Andreas Pieris. Ontology-Mediated Queries Distributing over Components. In *IJCAI*, pages 943–949, 2016.

Gerald Berger and Hans Tompits. On Axiomatic Rejection for the Description Logic $\mathcal{ALC}$. In *Proceedings of the KDPD 2013*, pages 65–82, 2013.