Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers

Evgeny Sherkhonov

Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof.dr. D.C. van den Boom
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in
de Agnietenkapel
op vrijdag 12 februari 2016, te 10:00 uur

door

Evgeny Sherkhonov

geboren te Ulan-Ude, Rusland

SIKS Dissertation Series No. 2016-05
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 612.001.012 (DEX).

Cover by David Graus
Printed by Off Page, Amsterdam

ISBN: 978-94-6182-652-7

Acknowledgements

There are a number of people that have supported me in this four-year-long journey. Even if you are not mentioned in the list below, you should know that I am very grateful for all the good you have done for me.

Special thanks go to:

Maarten Marx, my supervisor. Thank you Maarten for the guidance, your open heart and remarkable creativity.

Balder ten Cate. Thank you Balder for having me in Santa Cruz, it was one of the greatest periods of my life. I am indebted to you for opening a totally new research direction to me.

Maarten Marx, Yoichi Hirai, Alessandro Facchini, Balder ten Cate, Cristina Civili, and Wang-Chiew Tan, co-authors of the chapters of this thesis. I am grateful for all the great ideas and discussions we have had together.

Maarten de Rijke, my promotor. Thank you for creating a very stimulating environment in the group.

Sergey P. Odintsov, my first supervisor back in Novosibirsk. Thank you Sergey Pavlovich for sparking my interest in logic.

The Netherlands Organisation for Scientific Research (NWO) for sponsoring my PhD project and the Dutch Research School for Information and Knowledge Systems (SIKS) for their additional support.

Former and current members of ILPS, for all the fun at coffee breaks, parties and elsewhere. I wish I could mention all your names, but ILPS is very large.

#Running and #Beachvolley teams at ILPS for sports during and after the work hours.

Katya and Abdo, my paranymphs.

Last but foremost my family for being there for me.

Evgeny Sherkhonov
Amsterdam, December 2015

Contents

1 Introduction

One of the main uses of computers is to store, manage, and retrieve information. Large amounts of data stored on computers and structured by a defined schema are called databases. Software that helps to manipulate and retrieve data (by means of *querying*) is called a database management system. *Database theory* is the field of computer science that studies the foundations underlying the design and use of such systems.

Static analysis for queries is one of the main themes in database theory. One of the main static analysis problems for query languages is *containment*. Ever since the creation of the relational model, and SQL query language, the containment problem has been a topic of considerable interest (Chandra and Merlin, 1977a; Chekuri and Rajaraman, 2000; Gottlob et al., 2001). With the creation of other data models such as (tree) XML, data trees, graph databases, and corresponding languages (e.g., XPath and SPARQL, graph patterns or path queries) the containment problem has received renewed interest (Kostylev et al., 2014; Miklau and Suciu, 2004; Wood, 2003). Roughly, the containment problem for a query language $\mathcal{L}$ over the data model $\mathcal{D}$ can be formalized as follows. Let Q_1 and Q_2 be two queries from $\mathcal{L}$. Then we say that Q_1 is *contained* in Q_2, denoted as

$$Q_1 \subseteq Q_2,$$

if for every $\mathcal{D}$-database I, the answers to Q_1 over I are among the answers to Q_2 over I. This problem has a number of important applications. We list a few of them.

Query optimization. The task of query optimization is to produce a query Q' for a given query Q such that the answers of the queries are the same (i.e., they are *equivalent* and denoted as $Q \equiv Q'$) and the "cost" of executing Q' is lower than of Q. Thus, checking equivalence of two queries is an essential step for any optimization algorithm. Checking if Q' is equivalent to Q can be done by checking two containment relations: $Q \subseteq Q'$ and $Q' \subseteq Q$. Vice versa, for many query languages, the equivalence problem turns out to be equivalent to the containment problem. Indeed, if $\mathcal{L}$ contains conjunction $\wedge$, then checking $Q \subseteq Q'$ amounts to checking $Q \equiv Q \wedge Q'$.

Query answering using views. Another problem involving containment is query answering using materialized views. This problem has received considerable attention in the literature because of its various applications in other areas such as speeding up query evaluation, mobile computing, semantic data caching, and data warehouse design, see (Levy et al., 1995) for references, and (Halevy, 2000) for an overview of the results in this area. Roughly, the problem can be stated as follows. Given a query Q and a set of view definitions $V_1, \dots, V_n$, can we answer Q using only the answers computed for $V_1, \dots, V_n$?

A common approach to solve this problem is *query rewriting*. That is, finding a query Q' which uses only the views $V_1, \dots, V_n$ such that $Q' \equiv Q$. This definition shows that checking equivalence is an essential step in query rewriting. As we have mentioned above, this problem is tightly connected with the containment problem.

Why-not explanations. A third application that we mention here has recently appeared in the context of explanations to why-not questions. A why-not question aims to explain why particular data is missing from the result of a query. A newly proposed framework for why-not explanations (ten Cate et al., 2015) makes use of ontologies. Roughly, an ontology is a hierarchy of concepts that models a domain of interest. As an example, assume that $\mathtt{Woman}$ and $\mathtt{Man}$ are two concepts that represent women and men, while the concept $\mathtt{Human}$ represents human beings. These concepts obey the following hierarchy: $\mathtt{Woman} \sqsubseteq \mathtt{Human}$ and $\mathtt{Man} \sqsubseteq \mathtt{Human}$, which is read as "the concept $\mathtt{Human}$ *subsumes* the concepts $\mathtt{Woman}$ and $\mathtt{Man}$", and simply formalizes the fact that every woman and every man is a human being.

Without going too much into the details of the framework in (ten Cate et al., 2015), we point out that an explanation is a tuple of concepts that "generalizes" the missing data and thus gives a high level explanation to why this data is missing. As an example, given a database of train connections, and the query asking for pairs of cities reachable from each other, one can see that the tuple ⟨New York, Amsterdam⟩ is missing from the query result. One possible high-level explanation is the tuple $\langle \mathtt{AmericanCity}, \mathtt{EuropeanCity} \rangle$, which denotes that fact that there is no train connection between *any* two cities of America and Europe, and thus it explains the fact why ⟨New York, Amsterdam⟩ is missing. Note that the tuple of concepts $\langle \mathtt{City}, \mathtt{City} \rangle$ would not be an explanation, because there are cities that are connected to each other.

In this framework, we are interested in *most general* explanations. That is, in explanations such that there is no other explanation which is strictly more general (or which does not strictly subsume). Therefore, checking subsumption between two concepts is essential in algorithms for producing a most general explanation. Concepts are in fact queries and thus checking subsumption between concepts is the same as checking containment between queries.

The above list of applications, by no means exhaustive, is meant to demonstrate the importance of the containment problem. Note that these applications can also be casted in any query language and data model. This emphasizes the fact that we need to study the containment problem for various query languages and data models. In this thesis we dwell on *fragments* and *expansions* of popular query languages such as XPath and Conjunctive Queries over XML and relational data, respectively. One notable structural property that all the languages considered in this thesis share is *acyclicity*. The acyclicity condition on either languages or their underlying models, is prominent in database theory. Indeed, acyclicity has appeared in various settings where a problem at hand is intractable while acyclicity allows for efficient solutions. A notable example is the query evaluation problem for conjunctive queries. It is known that the problem is NP-complete (Chandra and Merlin, 1977a), while acyclic conjunctive queries allow for fast parallelizable algorithms (Gottlob et al., 2001).

Next we elaborate more on the research questions that we address in the thesis.

1.1 Research outline and goals

This thesis is divided into two parts. The first part is devoted to the containment problem for expansions of fragments of XPath interpreted over trees and Conjunctive Queries (CQ) interpreted over both trees and relational structures. Concerning this part of the thesis, we list the classes of queries under consideration, with the corresponding expansions. Then we define the research questions addressed in this thesis.

XPath fragments

XPath (W3C, 1999a) is a standard language for selecting paths or patterns in XML documents. It is an essential component in query languages for XML, such as XQuery (W3C, 2010), the transformation language XSLT (W3C, 1999b), and constraint languages such as XML Schema (W3C). Because of its presence in practically all programming tools for manipulating XML documents, XPath and in particular its static analysis (query evaluation, satisfiability and containment) have received siginificant interest over the past few years (see an overview in (Benedikt and Koch, 2009), and Chapter 2 for an overview on the containment for fragments of XPath).

Although XPath has many features, we restrict our attention to its *navigational* part, also known as Core Xpath (Gottlob et al., 2005b). The complexity of static analysis crucially depends on the syntactic constructs, or *axes* such as `child` and `descendent`, being used in queries. In particular, for a large class of fragments of XPath that contain full negation and filter expressions, the problems of satisfiability and containment are equivalent. For the latter problem, i.e., satisfiability, a rather complete picture of the complexity landscape is understood (Benedikt et al., 2008; Hidders, 2003), thus settling the containment problem for many fragments. On the other hand, for fragments *without* negation, or *positive* fragments, the containment and satisfiability are different. A lot of work has been devoted to positive fragments of XPath, a notable one of which is Tree Patterns (Amer-Yahia et al., 2002; Miklau and Suciu, 2004). Tree Patterns is a fragment that uses only downward axes, i.e., `child` and/or `descendent`, and that may also use filter expressions and the wildcard.

In this thesis we continue the study of the containment problem for fragments of Positive XPath but with a few additional constructs that allow to express some useful properties of trees. In particular, we single out the following additional constucts, which are formally defined in the corresponding chapters.

- Label negation. As opposed to full negation, in XPath formulas with label negation we allow the negation sign only in front of a label. An additional restriction that is prominent in this thesis is *safety* that allows an occurrence of a negated label only in conjunction with an occurrence of a positive label. In Chapter 3 we study the impact of safe label negation on the containment problem.

- Attribute value comparisons. This type of construct allows us to reason over numeric values that sit in the attributes of tree nodes. As an example, the XPath query $//person/\star[@_{age} \geq 18]$ selects all adults present in a document about people. Likewise, in Chapter 3 we study the impact of expanding positive XPath with attribute value comparisons on the containment problem.

- Conditional descendent axis. We initiate the study on downward positive XPath expanded with a restricted form of the transitive closure of the child relation, called *conditional descendent* axis. This expansion can easily be explained pictorially: to descendent edges in usual Tree Patterns we add labels that are (conditional) Tree Patterns themselves. The meaning of a descendent edge labeled with Q and ending in P is a path of child steps ending in a P node such that all the intermediate nodes are Q-nodes. This very much resembles the until operator from temporal logic (Kamp, 1968).

  When boosting the expressive power, we can expect that the complexity of containment is higher than for usual Tree Patterns. In Chapter 5 we investigate what the exact complexity of containment is.

Conjunctive Queries

Conjunctive Queries (CQ) are one of the most popular query languages used in practice (Abiteboul et al., 1995). In SQL they correspond to select-from-where queries where the conditions can only be combined using “AND”. The containment problem for conjunctive queries interpreted over relational databases was one of the first fundamental problems in database theory, though under a different name. In particular, for this class of queries the containment problem is equivalent to the query evaluation problem that was studied in (Chandra and Merlin, 1977b). Furthermore, this problem has deep connections with other areas, and appears in a different disguise in AI as the Constraint Satisfaction Problem (Kolaitis and Vardi, 2000), and in graph theory as the H-coloring problem (Hell and Nesetril, 2004).

While being well studied over relational databases, the containment problem for conjunctive queries interpreted over unranked trees has been considered only recently (Björklund et al., 2011). In this contexts, conjunctive queries are built of unary relations, and the binary relations $Child$, $Descendent$ and $Following$. In (Björklund et al., 2011) it is shown that containment for CQ over trees is Π_2^P-complete. In Chapter 3, we investigate the impact of the following constructs on the containment problem: (safe) label negation, and attribute value comparisons.

All in all, for every language $\mathcal{L}$ of Positive Xpath, Tree Patterns and Conjunctive Queries expanded with the new constructs mentioned above, we study the following research question.

RQ 1 What is the complexity of the containment problem for $\mathcal{L}$ over unranked trees?

Tractable containment for expansions of Conjunctive Queries

From (Chandra and Merlin, 1977a) it is known that containment for CQ over relational databases is NP-complete. Because of the practical importance of this query language, a lot of research has been devoted to finding natural and large classes of restrictions on CQ allowing for tractable containment. One such restriction is *acyclicity* (Gottlob et al., 2001) that allows for a fast parallelizable algorithm.

Various expansions of Conjunctive Queries over relational databases have been considered in the past. Atomic negation and arithmetic comparisons are among them (Ullman, 2000; van der Meyden, 1997). A conjunctive query is said to have atomic negation if the negation sign appears only in front of an atom (or a subgoal). A conjunctive query has arithmetic comparisons if it contains comparisons of the form $X \mathtt{op}\ c$ as subgoals, where X is a variable appearing in some subgoal, $\mathtt{op}$ is a comparison operator and c a constant.

It turns out that adding these new features results in higher complexity for containment – Π_2^P. However, it is rather surprising that tractable restrictions of these expanded languages have received little attention before. In Chapter 4 we start addressing this issue by considering a natural candidate for a restriction – acyclicity. Thus, we try to answer the following

RQ 2 Does acyclicity make the complexity of containment for conjuctive queries expanded with atomic negation or arithmetic comparison tractable? If not, what additional restrictions can be imposed to make it tractable?

Why-not explanations

The second part of this thesis is devoted to a new framework for why-not explanations. Given a (relational) database possibly with integrity constraints, a computed answer to a query, and a tuple that does *not* belong to the answer, a *why-not question* asks why the tuple is missing in the answer. This problem appears quite often during the process of debugging either a database system, or a query posed against the system. Explanations to why-not questions might pinpoint a possible source of error which could be the constraints, views, the query or gaps in the data.

The proposed framework makes use of ontologies and provides *high-level* explanations. Ontologies contain *concepts* and *relationships* between them. In this way, ontologies are a powerful way to model the domain of interest and provide a high-level view of the data. Our framework can make use of either an existing ontology (e.g., DBPedia) or an ontology extracted from the database and/or a schema that includes integrity constraints. Since an ontology is a central object in our framework, the first question that we try to answer in Chapter 6 is

RQ 3 How to extract an ontology from a database instance or a schema?

It turns out that this problem can be cast as the containment problem. Indeed, an ontology can be represented as a set of concept subsumptions. In case we want to extract an ontology from a schema with integrity constraints, concepts have a form of a view (or a query) and thus subsumption is just a reformulation of the containment problem. The exact answer to the research question largely depends on the type of constraints available in the schema. In Chapter 6 we answer the questions for various constraints, including views, inclusion and functional dependencies.

Next, in the framework, among all why-not explanations we are interested in "good" ones only. Intuitively, good explanations are those that are as general as possible with respect to the ontology, and thus they are "high-level" and capture a general gap/error in the system. Consider the following example. Suppose we have a database of publications that was obtained as the result of an integration process (say, from DBLP, Google Scholar, and PubMed). Then suppose we are asking for all publications of Ronald Fagin that were published between the year 2000 and 2005. When we query the database, it turns out that his most cited paper R. Fagin et al. "Reasoning about knowledge." MIT Press, 2003, is not in the result. A possible "good" solution could be that "it is an MIT Press publication and all MIT Press publications are missing".

We are interested in the following research question.

RQ 4 How to produce "good" explanations?

In the following research chapters we seek answers to all of the above research questions.

1.2 Main contributions and overview

In this section we outline the structure of the thesis and summarize the contributions of each chapter except the introduction chapter.

Chapter 2. In this chapter we review related work and introduce key concepts that are useful for presenting the main material. In particular, we review previous work on the containment problem for various fragments of XPath, as well as Conjunctive Queries over trees and relational structures. Related to why-not explanations, we review known results on why and why-not explanations.

Part I: Containment problem for Acyclic queries

Chapter 3. In this chapter we study the containment problem for queries over trees expanded with attribute value comparisons. Björklund et al. (2011) showed that containment for conjunctive queries (CQ) over trees and positive XPath is respectively Π_2^P and coNP-complete. In this chapter we show that the same problem has the same complexity when we expand these languages with XPath's attribute value comparisons. We show that different restrictions on the domain of attribute values (finite, infinite, dense, discrete) have no impact on the complexity. Making attributes required does have an impact: the problem becomes harder. We also show that containment of tree patterns without the wildcard $*$, which is in PTIME, becomes coNP hard when adding equality and inequality comparisons, i.e., comparisons of the form $@_a = c$ and $@_a \neq c$.

Chapter 4. In this chapter we study the containment problem for conjunctive queries (CQ) with atomic negation or arithmetic comparisons over relational databases. It is known that the problem is Π_2^P-complete (Ullman, 2000; Wei and Lausen, 2003). The aim of this chapter is to find restrictions on CQ that allow for tractable containment.

In particular, we consider acyclic conjunctive queries. It turns out that even with the most restrictive form of acyclicity (Berge-acyclicity), containment is coNP-hard. We show that for a particular fragment of Berge-acyclic CQs with atomic negation, namely the class of child-only tree patterns with a restricted form of negation, containment is solvable in PTime.

Chapter 5. In this chapter we consider an expansion of traditional tree patterns. A Conditional Tree Pattern (CTP) expands an XML tree pattern with labels attached to the descendant edges. These labels can be XML element names or Boolean CTP's. The meaning of a descendant edge labelled by A and ending in a node labelled by B is a path of child steps ending in a B node such that all intermediate nodes are A nodes. In effect this expresses the *until B, A holds* construction from temporal logic.

This chapter studies the containment problem for CTP. For tree patterns (TP), this problem is known to be coNP-complete. We show that it is PSpace-complete for CTP. This increase in complexity is due to the fact that CTP is expressive enough to encode an unrestricted form of label negation: $* \setminus a$, meaning "any node except an a-node". Containment of TP expanded with this type of negation is already PSpace-hard.

CTP is a positive, forward, first order fragment of Regular XPath. Unlike TP, CTP expanded with disjunction is not equivalent to unions of CTP's. Like TP, CTP is a natural fragment to consider: CTP is closed under intersections and CTP with disjunction is equally expressive as positive existential first order logic expanded with the until operator.

Part II: Why-not explanations

Chapter 6. In this chapter we propose a foundational framework for *why-not explanations*, that is, explanations for why a tuple is missing from a query result. Our why-not explanations leverage concepts from an ontology to provide high-level and meaningful reasons for why a tuple is missing from the result of a query.

A key algorithmic problem in our framework is that of *computing a most-general explanation* for a why-not question, relative to an ontology, which can either be provided by the user, or it may be automatically derived from the data and/or schema. We study the complexity of this problem and associated problems, and present concrete algorithms for computing why-not explanations. In the case where an external ontology is provided, we first show that the problem of deciding the existence of an explanation to a why-not question is NP-complete in general. However, the problem is solvable in polynomial time for queries of bounded arity, provided that the ontology is specified in a suitable language, such as a member of the DL-Lite family of description logics, which allows for efficient concept subsumption checking. Furthermore, we show that a most-general explanation can be computed in polynomial time in this case. In addition, we propose a method for deriving a suitable (virtual) ontology from a database and/or a schema, and we present an algorithm for computing a most-general explanation to a why-not question, relative to such ontologies. This algorithm runs in polynomial-time in the case when concepts are defined in a selection-free language, or if the underlying schema is fixed. Finally, we also study the problem of computing *short* most-general explanations, and we briefly discuss alternative definitions of what it means to be an explanation, and to be most general.

Notably, in the above algorithms, it is important to decide the containment (or, subsumption) between two concepts. We provide a detailed complexity analysis for this problem when concepts are defined as projections of relations, or projections of relations with selection, or conjunctions thereof, with respect to different types of constraints.

1.3 Origins

We list publications and submissions on which the content chapters are based, and we discuss the role of co-authors.

Chapter 3 This chapter is based on M. Marx and E. Sherkhonov. "Containment for queries over trees with attribute value comparisons" (Marx and Sherkhonov, 2015), which is accepted for publication at *Information Systems* and an expanded version of E. Sherkhonov and M. Marx. "Containment for tree patterns with attribute value comparisons." *Proceedings of the 16th International Workshop on the Web and Databases 2013, WebDB 2013*, 2013 (Sherkhonov and Marx, 2013). Both authors contributed to the proofs of the results and to the text.

Chapter 4 This chapter is based on a journal submission E. Sherkhonov and M. Marx. "Containment of Acyclic conjunctive queries with atomic negation and arithmetic comparisons". (Sherkhonov and Marx, 2015), which is currently under review. Both authors contributed to the proofs of the results and to the text.

Chapter 5 This chapter is based on A. Facchini, Y. Hirai, M. Marx, and E. Sherkhonov. "Containment for conditional tree patterns". *Logical Methods in Computer Science*, 2015 (Facchini et al., 2015). All authors contributed to the proofs of the results and to the text.

Chapter 6 This chapter is based on B. ten Cate, C. Civili, E. Sherkhonov and W-C. Tan. "High-Level Why-Not Explanations using Ontologies". *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, PODS 2015 (ten Cate et al., 2015). We introduce a new framework for why-not explanations. All authors contributed to the development of the framework, algorithms, proofs of the results and to the text. This work also appears in the PhD thesis of Cristina Civili titled "Processing Tuple-Generating Dependencies for Ontological Query Answering and Query Explanation", Sapienza University of Rome. Technical contributions of Civili were Section 6.4.1 on the case of external ontology and Section 6.5 on algorithms for computing why-not explanations. Sherkhonov's contributions include Section 6.5.3 on deriving an ontology from a schema, and Section 6.6 on variations of the framework.

The chapters are almost exact copies of the conference and journal papers, except for the related work sections of the papers. Relevant related work is discussed in Chapter 2.

2 Background and Preliminaries

In this chapter we introduce the main concepts that will be useful for explaining our ideas as well as for providing the background on known results in the field.

We introduce some underlying concepts from database theory. First we recall the relational data model and conjunctive queries. After that, we review XML and its tree model, together with the XPath query language and its fragments. In particular, we give special attention to Tree Patterns.

At the end of this chapter we discuss existing approaches to why and why-not explanations.

2.1 Containment for queries over relational data

Relational data

Relational databases, proposed by E. Codd (Codd, 1970) is the standard model for storing data. In this model, data is stored in tables, where each row represents an entity or an object with information about their properties. For instance, a row can contain information about a person with their name, address and contact information. A table then represents a collection of objects of a similar type. An attribute of a table is a particular type that the objects share. By a *schema* we mean the set of all table names, with a set of integrity constraints.

Mathematically, each table name is a relational name R whose arity the same as the number of attributes in the table. Then each table in a given database defines the extension of the corresponding relation R, and each entry in the table is a tuple in the relation R. This way, a database defines a relational structure $I = (dom, \cdot^I)$ where dom is the set consisting of all data elements of the database, and $\cdot^I$ is the interpretation function assigning to each relational name R the set of tuples in the table corresponding to R. Alternatively, a database can be represented as a set of facts $R(\bar{a})$ such that $\bar{a}$ is an entry in the table of R.

Many systems are built upon the relational model, called Relational Database Management System (RDBMS), including MySQL, PostgreSQL, IBM DB2, Microsoft SQL Server and Oracle.

Conjunctive queries over relational databases

Querying is the basic operation of retrieving needed information from a database. Practically all database systems use SQL as the query language.

The most frequent class of SQL queries are select-from-where queries, or also known as conjunctive queries. Let $\mathbf{S}$ be a relational schema. Then a k-ary ($k \geq 0$) *conjunctive query* is a first-order logic formula of the form $\exists \bar{x}.(P_1(\bar{x}, \bar{y}) \wedge \ldots \wedge P_n(\bar{x}, \bar{y}))$, where $P_i \in \mathbf{S}$ for every $i \in [1, \ldots, n]$, such that the free variables are in $\bar{y}$ and $|\bar{y}| = k$. Each atom P_i is also called a *subgoal* of the query. Often conjunctive queries are written in the following datalog notation:

$$Q(\bar{y}) \text{ :- } P_1(\bar{x}, \bar{y}), \ldots, P_n(\bar{x}, \bar{y}),$$

for a new relational symbol Q. In this case the atom $Q(\bar{y})$ is called the *head*, and the expression on the right hand side is the *body* of the query. We will be using either of these notations.

As an example,

$$\begin{aligned} Q(e\_name, m\_name) \text{ :- } & Employee(id, p\_id, e\_name, b\_date), Project(p\_id, p\_name), \\ & Manager(p\_id, m\_name) \end{aligned}$$

is a conjunctive query which asks for employees and their supervisors. The semantics of conjunctive queries is as follows. We say that Q holds in a database instance I if there is an assignment θ of the variables of Q to the domain of I such that for every subgoal $P(\bar{x})$ of Q, the fact $P(\theta(\bar{x}))$ is in I. Then the *answer* of $Q(\bar{y})$ on I is the set of tuples $\theta(\bar{y})$ for every such assignment θ. If there is such an assignment θ, the instance I is called a *model* of Q.

Conjunctive queries are a major success in database theory. They are not only the most frequent class of queries used in practice, but also possesses some nice properties.

We start with the problem of query evaluation. Formally, given a database I and a conjunctive query Q and a tuple $\bar{a}$, the evaluation problem is to decide if $\bar{a} \in Q(I)$, i.e., if the tuple is present in the query result. When a database is considered part of the input, i.e., we talk about *combined complexity*, the evaluation problem for conjunctive queries is NP-complete (Chandra and Merlin, 1977b). However, when the query is fixed, i.e., we talk about *data complexity*, then the evaluation problem belongs to the low complexity class AC^0. Due to this low data complexity and their reasonable expressive power, conjunctive queries have been extremely successful in databases.

One of the characteristic properties of conjunctive queries is that the evaluation and containment problems are equivalent (Abiteboul et al., 1995). This is proved via the *canonical model* of a conjunctive query. That is, each conjunctive query can be considered as a database instance itself, where each variable is replaced with a distinct constant. This database is a model of the query. Thus, the containment of a query Q_1 in a query Q_2 amounts to evaluating Q_2 on the corresponding canonical model I_{Q_1}. Vice versa, each database instance can be considered as a query, where each fact is considered as a subgoal of the query. Then evaluating Q on a database instance I is the same as checking containment $Q_I \subseteq Q$, where Q_I is the query that corresponds to I.

This characterization gives some more nice properties. An immediate one is the small counter-example property. The latter means that if containment between two queries does not hold, then there is a witness for non-containment whose size is polynomial in the sizes of the input. Indeed, the canonical model of the left-hand side conjunctive query is already a counter-example and of linear size. Another nice property is that containment of *unions* of conjunctive queries can be reduced to containment of conjunctive queries. This is due to the following disjunctive property. For a database instance I and a union of conjunctive queries $\bigcup Q_i$ it holds that $I \models \bigcup Q_i$ if and only if $I \models Q_j$ for some j (Sagiv and Yannakakis, 1980).

Conjunctive queries with atomic negation and arithmetic comparisons

Conjunctive queries are a good language for specifying positive information. However often in practice we need to express negative information as well. Certain negative information can be expressed by expanding conjunctive queries with atomic negation, or in other words by negating subgoals of the query. For instance, the following is a conjunctive query with atomic negation

$$Q'(e\_name) \text{ :- } Employee(id, p\_id, e\_name, b\_date), \neg Manager(p\_id, e\_name),$$

which asks for the names of the employees for whom there is a project they are involved in but which they are not managing. The result of applying a conjunctive query with atomic negation to a database is essentially the same as for usual conjunctive queries. The only difference is that when we apply a substitution of constants for variables the atoms in the negated subgoals must be false in the database. We distinguish a restricted form of atomic negation, called *guarded*. In conjunctive queries with guarded negation, for an occurrence of a negated subgoal $\neg R(\bar{x})$ in the query there must be an occurrence of a positive subgoal $P(\bar{y})$ in the query such that the variables $\bar{x}$ are contained in the variables $\bar{y}$. Note that the negation in the query Q' above is guarded. In Chapter 4 we will see that this restricted form of negation helps to reduce the complexity of containment for acyclic conjunctive queries with atomic negation.

Another important expansion of conjunctive queries is arithmetic comparisons. In this case we assume that data values come from a total order $(\mathbf{Const}, <)$. In real databases, $(\mathbf{Const}, <)$ can be a numeric domain such as reals, rational or natural numbers, or the domain of strings with the lexicographic order. Syntactically, in addition to relational atoms we can have arithmetic comparisons of the form $X \texttt{op}\ c$ or $X \texttt{op}\ Y$, where X and Y are variables, $c \in \mathbf{Const}$ a constant and $\texttt{op}$ a comparison operator in $\{=, \neq, <, >, \leq, \geq\}$. The semantics is obvious: when the constants are substituted for constants, every arithmetic comparison must hold in $(\mathbf{Const}, <)$.

As an example, the simple query

$$Q''(e\_name) \text{ :- } Employee(id, p\_id, e\_name, b\_date), b\_date \geq \text{'1980-01-01'}$$

asks for names of the employees that were born in 1980 or after. Here the order $<$ is the natural order on the timeline.

Note that arithmetic comparisons also introduce some form of negative and disjunctive information.

Datalog

Datalog is another expansion which adds recursion capabilities to conjunctive queries. A *datalog program* Π consists of rules of the same form as conjunctive queries defined above. Now the relational schema is divided into two sets of *extensional* (EDB predicates) and *intentional* (IDB predicates) relations. Extensional relations can only appear in the body of the rules of the program. Intentional relation must appear in the head of some rule of the program, and can appear in the body of the rules. A distinguished intensional predicate G is called the *goal* predicate.

As an example, the following program computes the transitive closure of the extensional binary relation R, where T is the goal predicate:

$$T(x,y) \text{ :- } R(x,y)$$
$$T(x,y) \text{ :-} R(x,z), T(z,y).$$

Let Π be a datalog program, P an IDB predicate and I a database instance interpreting EDB predicates. We define the semantics of P in I w.r.t. Π as the fixpoint of relations $P_i(I)$ defined via the following process, starting with $P_0(I) = \emptyset$:

- Let I^i be the expansion of I with $P_i(I)$ for all IDB predicates P.
- If r is a rule with $P(x_1, \ldots, x_k)$ in the head, $\bar{y}$ the bound variables of r and $\varphi(\bar{x}, \bar{y})$ is the body of r, let $P^r_{i+1}(I)$ be defined as $\{\bar{a} \mid I^i \models \exists \bar{y} \varphi(\bar{a}, \bar{y})\}$.

We define $P_{i+1}(I)$ to be the union of $P^r_{i+1}(I)$ over all rules r with P in the head. Then the result of Π on I is $G_\Pi(I)$, the result of evaluating the goal predicate. By $\Pi(I)$ we denote the minimal model of Π on I, i.e., the union of I and $P_\Pi(I)$ over all intensional predicates P of Π.

Datalog can also be considered as a language for expressing *views*. If datalog is not allowed to use recursion, then we talk about UCQ-view definitions. We say that a relation R *depends* on a relation P w.r.t. a datalog program if there is a rule in the program such that R appears in the head and P in the body of the rule. The program is called *nonrecursive* if the depends relation is acyclic. Expressivity-wise nonrecursive datalog programs are exactly the unions of conjunctive queries. Indeed, this can be seen by performing *unfolding*. Namely, we keep replacing each intensional predicate P with the union of the bodies of the rules with P in the head, and bring the union in front. This process terminates due to the acyclicity condition of the depends relation. This way, non-recursive Datalog is a macro facility for conjunctive queries, which is very *succinct*. More precisely, Datalog is doubly exponential more succinct than conjunctive queries (Benedikt and Gottlob, 2010).

Query containment

Containment for queries is one of the well studied problems in database theory. A celebrated result of (Chandra and Merlin, 1977b) is NP-completeness of CQ containment. As mentioned in Chapter 1, the conjunctive query containment problem is underpinning for many other problems and thus it has been studied under various settings since the work of (Chandra and Merlin, 1977b). We review the following settings related to conjunctive query containment.

- Restrictions for tractable conjunctive query containment,
- Containment of expanded conjunctive queries,
- Conjunctive query containment under constraints.

Tractable restriction for containment of conjunctive queries

Because of its importance to practice, there has been a lot of work devoted to finding natural and large classes of conjunctive queries for which containment is tractable.

One of the most prominent restriction is acyclicity. From the lower bound proofs, it follows that NP hardness for containment is due to certain cycles in the hypergraph representation of conjunctive queries. A *hypergraph* is a structure with a non-empty set of vertices, and a collection of subsets of the set of vertices called hyperedges. For a given conjunctive query, the corresponding hypergraph is constructed as follows. The set of variables in the query constitutes the set of vertices. Furthermore, for every atom in the query there is a hyperedge consisting of the variables of the atom. Note that in case a conjunctive query contains atoms of arity at most 2, the corresponding hypergraph is an ordinary graph.

There is a number of non-equivalent definitions of acyclicity for hypergraphs. They are in decreasing order of restrictiveness: Berge-acyclicity, γ-acyclicity, β-acyclicty and α-acyclicity. We refer to (Fagin, 1983) for the definitions of acyclicity and examples that the order of restrictiveness is strict. All of them, however, collapse to ordinary acyclicity in case of graphs.

In (Yannakakis, 1981) it was shown that some problems that are NP-complete in general, have PTIME algorithms with the assumption of α-acyclicity. The evaluation problem, and thus containment, is one of them. The algorithm proposed in (Yannakakis, 1981) is based on the notion of *join forest*, where each node is labeled with an atom such that if a variable occurs in two atoms, then the corresponding nodes are connected and the variable occurs in each node on the unique path connecting these two nodes. Then the evaluation algorithm works in a bottom up manner on each tree of the join forest by performing a number of semi-join operations. This algorithm runs in PTIME. Gottlob et al. (2001) show that the evaluation and containment problems for α-acyclic conjunctive queries are in fact $LOGCFL$-complete. The latter is the class of decision problems that are logspace reducible to a context-free language, and which allow for highly parallelizable algorithms.

Since then there have been a number of publications on generalizing this tractability result. In this direction different notions of “width” have been introduced. Intuitively, these notions of width measure how “far” the query is from being acyclic.

One such notion of width is *query width* from (Chekuri and Rajaraman, 2000). In this paper the authors prove that if the query width of the left hand side of the containment problem is bounded by a fixed constant k, then containment is decidable in polynomial time. This result extends the result of (Yannakakis, 1981) since acyclic queries are exactly of query width 1. However, one drawback of the tractability result is that it is NP-complete to decide if the query width is bounded by a given constant (Gottlob et al., 2002). In (Gottlob et al., 2002) the authors further generalize the notion of query width, with the notion of *hypertree width* and show that containment is solvable in polynomial time. Furthermore, boundedness of a hypertree width by a constant is recognizable in polynomial time.

Containment for expansions of conjunctive queries

It is not surprising that when conjunctive queries are expanded with more expressivity capabilities, the complexity of the containment problem increases. In (Ullman, 2000) it is argued that conjunctive queries with atomic negation is Π_2^P-complete. For the upper bound, canonical databases are used. In particular, the algorithm constructs canonical models of the left hand side of the containment problem and checks if the right hand side query is true on each model. Note, there might be an exponential number of canonical models for a conjunctive query with atomic negation. In (Wei and Lausen, 2003) the authors propose an algorithm for containment of conjunctive queries with safe negation (this is when a variable occurring in a negative subgoal must occur in some positive subgoal) that can terminate earlier, but in the worst case takes the same time as the algorithm in (Ullman, 2000). As for the lower bound, Farré et al. (2007) propose a method for proving a Π_2^P lower bound where only safe (and thus guarded) unary atoms are negated.

Containment for conjunctive queries when expanded with arithmetic comparisons also turns out to be Π_2^P-complete (Klug, 1988; van der Meyden, 1997). Afrati et al. (2004) consider various restrictions on the type of comparison operations on either of the two input conjunctive queries with comparisons, as well as on interaction between the comparisons, so that the containment is in NP (cf. Table 1 in (Afrati et al., 2004)). However, it was left open what the exact complexity of containment of conjunctive queries with comparisons of the type $X \mathsf{op} c$ is, where c is a constant and $\mathsf{op} \in \{=, \neq, \leq, \geq, <, >\}$. Note that Π_2^P-lower bound proof in (van der Meyden, 1997) uses inequalities of variables, i.e., the construct $X \neq Y$ for variables X and Y. Nevertheless, restricting arithmetic comparisons to be of the form $X \mathsf{op} c$ does not lower the complexity. This is argued in (Farré et al., 2007), where Π_2^P-hardness of containment for CQs with comparisons was shown, using comparisons of the form $X \neq c$. This proof can also be adapted to use comparisons of the form both $X \leq c$ and $X > c$ (Nutt, 2013).

The containment problem for Datalog has also been a subject of large interest. Unfortunately, for full Datalog, containment is undecidable (Shmueli, 1993). The reason for undecidability of containment is recursion. Chaudhuri and Vardi (1997) show that when the use of recursion is forbidden in the right hand side program, containment becomes decidable. In particular, they show that containment of a datalog program in a union of conjunctive queries is 2EXPTIME-complete. Containment of a union of conjunctive queries in a Datalog program is EXPTIME-complete (Cosmadakis and Kanellakis, 1986), on the other hand. Furthermore, Benedikt et al. (2005) conduct a fine-grained complexity analysis of the containment problem for various fragments of nonrecursive Datalog programs. In particular, containment for nonrecursive datalog is CONEXPTIME-complete. In Chapter 6 we will make use of these results for nonrecursive datalog, to establish complexity bounds for the problem of extracting an ontology from a set of views.

Containment for conjunctive queries under constraints

Constraints are used to validate databases and thus discard unwanted ones. Typical integrity constraints for relational databases can be classified into the classes of *functional*

```
<company>
  <department code ="d1">
    <name>
       Supply
    </name>
    <manager>
      John
    </manager>
    <member id ="0005">
      Jack
    </member>
    <member id ="0003">
      Maria
    </member>
  </department>
  <department code ="d2">
    <name>
       HR
    </name>
    <manager>
      Jude
    </manager>
    <member id ="0023">
      Dan
    </member>
  </department>
</company>
```

Figure 2.1: Example XML document containing data about company employees.

dependencies, which include key dependencies, *join dependencies* which include multi-valued dependencies, and inclusion dependencies (Abiteboul et al., 1995). Query containment under constraints is defined in the same way as the usual containment with the difference that quantification is only over databases that conform to the constraints. Conjunctive query containment under constraints has been studied extensively as well. It is studied in (Johnson and Klug, 1984) for the case of inclusion and functional dependencies, in (Aho et al., 1979) for multi-valued and functional dependencies, in (Dong and Su, 1996) for constraints expressed as Datalog programs, and in (Calvanese et al., 2008) for constraints expressed in Description Logic.

2.2 Containment for XPath

In this section we recall the XML data model, the syntax and semantics of XPath, its popular fragments and results on the containment problem for these languages.

2.2.1 XML and tree models

The Extensible Markup Language (XML) is a standard markup language used to represent, store and transfer data across the web. It is used to describe *hierarchical* data with user defined nested tags. For instance, Figure 2.1 illustrates an XML document containing a piece of information about a company. As seen from this example, the structure of the document is hierarchical, and formed by nested tags. Each tag name is called an *element type*, and each occurrence of a tag is an *element*. Each element might contain *attributes*. In this example each `department` element contains an attribute `code` which denotes the unique code of the department.

Usually XML documents are abstracted as labeled sibling-ordered unranked *trees*. Here, each element type is a label. An element can have attributes. A tree structure explicitly illustrates the hierarchical structure of XML documents. For instance, the XML document from Figure 2.1 is represented as a tree in Figure 2.2.

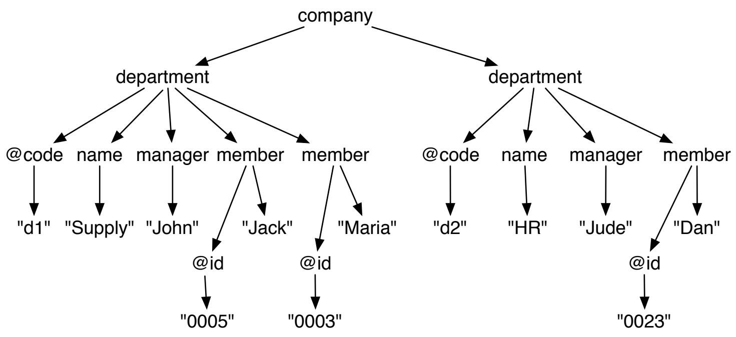


Figure 2.2: Ordered unranked tree corresponding to the document from Figure 2.1.

Let Σ be a countably infinite set of labels that correspond to the tag names, A a set of attribute names, $\mathbf{Const}$ a set of constants, all are pairwise disjoint. Formally, an *XML tree with attributes*, is a tuple $(N, E, <, r, \rho, att)$, where N, the set of nodes of the tree, is a prefix closed set of finite sequences of natural numbers, $E = \{(\langle n_1, \ldots, n_k\rangle, \langle n_1, \ldots, n_k, n_{k+1}\rangle) \mid \langle n_1, \ldots, n_{k+1}\rangle \in N\}$ is the child relation, the sibling order $<$ is defined as $\{(\langle n_1, \ldots, n_k\rangle, \langle n_1, \ldots, n_k + 1\rangle) \mid \langle n_1, \ldots, n_k\rangle, \langle n_1, \ldots, n_k + 1\rangle \in N\}$, $r = \langle\rangle$ is the root of the tree, ρ is the function assigning to each node in N an element of Σ, and $att : N \times A \rightarrow \mathbf{Const}$ a partial function. In this thesis we also consider *multi-labeled* trees which are defined in the same way as XML trees with the exception that the function ρ assigns to each node a *finite subset* of Σ.

2.2.2 XPath and its fragments

XPath is a popular query language for XML documents that lies at the heart of the more expressive language XQuery, the XML schema languages, and the transformation language XSLT. It is used to navigate and select nodes in an XML tree. In this thesis we consider fragments and expansions of the navigational part of XPath 1.0, called Core XPath in (Gottlob et al., 2005a). Its syntax allows for composing, and taking unions of path expressions, and all boolean operations in the filter expressions. We adopt the following syntax for XPath, where φ is a node expression, and α a path expression.

$$\begin{array}{lcl} \mathit{step} & ::= & \downarrow \mid \uparrow \mid \leftarrow \mid \rightarrow , \\ \varphi & ::= & p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\varphi \\ \alpha & ::= & \mathit{step} \mid ?\varphi \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \mathit{step}^+ \mid \rightarrow_f \mid \leftarrow_p . \end{array}$$

Given a tree $T = (N, E, <, r, \rho)$, the semantics of path and node expressions are defined by a mutual induction, where R^* and R^+ denote the reflexive transitive and transitive closure of the relation R respectively:

- $[\![\downarrow]\!]_T = E$,
- $[\![\uparrow]\!]_T = E^{-1}$,

- $[\![\rightarrow]\!]_T = <$,
- $[\![\leftarrow]\!]_T = <^{-1}$,
- $[\![\rightarrow_f]\!]_T = [\![\uparrow]\!]^* \circ [\![\rightarrow]\!]^+ \circ [\![\downarrow]\!]^*$,
- $[\![\leftarrow_p]\!]_T = [\![\rightarrow_f]\!]^{-1}$,
- $[\![?\varphi]\!]_T = \{(n, n) \in N \times N \mid T, n \models \varphi\}$,
- $[\![\alpha; \beta]\!]_T = [\![\alpha]\!]_T \circ [\![\beta]\!]_T$,
- $[\![\alpha \cup \beta]\!]_T = [\![\alpha]\!]_T \cup [\![\beta]\!]_T$,
- $[\![\alpha^+]\!]_T = ([\![\alpha]\!]_T)^+$ for $\alpha \in \{\downarrow, \uparrow, \rightarrow, \leftarrow\}$,

and

- $T, n \models \top$,
- $T, n \models p$ iff $p \in \rho(n)$,
- $T, n \models \neg\varphi$ iff $T, n \not\models \varphi$,
- $T, n \models \varphi \wedge \psi$ iff $T, n \models \varphi$ and $T, n \models \psi$,
- $T, n \models \varphi \vee \psi$ iff $T, n \models \varphi$ or $T, n \models \psi$,
- $T, n \models \langle\alpha\rangle\varphi$ iff there is a node m with $(n, m) \in [\![\alpha]\!]_T$ and $T, m \models \varphi$.

Thus, *step* selects a pair of nodes that are in the child, parent, next-sibling or previous-sibling relations in the tree. Furthermore, the $\rightarrow_f$ and $\leftarrow_p$ axes select nodes that are in the following and the preceding relations in the tree respectively. Note that $\top$ is the same as the wild card axis known in other variants of XPath syntax.

As an example, the following path expression $\downarrow^+; ?department; \downarrow; ?manager$ selects the managers of every department from the document in Figure 2.1.

Core XPath can be extended with attribute value comparisons of the form $(\alpha_1; @_a)$ $\mathsf{op}(\alpha_2; @_b)$ and $@_a\mathsf{op}c$ as node expressions, where a and b are attribute names, c a constant and $\mathsf{op} \in \{=, \neq, >, <, \geq, \leq\}$. Given a tree $T = (N, E, <, r, \rho, att)$ with attributes, the semantics for these expressions is as follows:

- $T, n \models (\alpha_1; @_a)\mathsf{op}(\alpha_2; @_b)$ iff there are nodes n' and n'' such that $(n, n') \in [\![\alpha_1]\!]_T$, $(n, n'') \in [\![\alpha_2]\!]_T$ and $att(n', a)\mathsf{op}att(n'', b)$,
- $T, n \models @_a\mathsf{op}c$ iff $att(n, a)\mathsf{op}c$.

Note that these constructs are available in XPath 1.0. This expansion allows us to reason about the attribute values of trees. A number of other expansions of XPath have been proposed and used in practices. XPath expanded with Kleene star for path expressions results in Regular XPath (ten Cate, 2006). XPath expanded with *conditional axes* results in Conditional XPath (Marx, 2005). Using Kleene star, the conditional axis can be expressed as $(?\varphi; step)^*$. Notably, Conditional XPath is expressively complete for FO

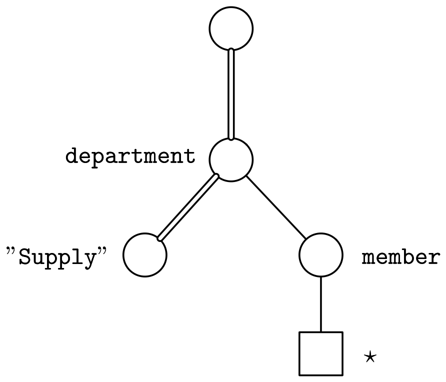


Figure 2.3: Tree representation of .//department[//"Supply"]/member/$\star$.

on XML trees. That is, every query defined in FO (over the signature for trees) can be expressed in Conditional XPath as well (Marx, 2005).

A popular fragment of XPath that has received considerable attention in the past is Tree Patterns. They can be considered to be the analog of conjunctive queries for XML. Formally, Tree Patterns is the positive downward fragment of XPath. That is, this is the fragment where in path expressions $step$ is restricted to $\downarrow$ only, and only the test, the transitive closure and compositions of paths are allowed. In node expressions, disjunction and negation are disallowed. For instance, the expression $\downarrow^+; ?department; \downarrow ; ?manager$ is also a Tree Pattern. A much more succinct syntax is very common:

$$q := \ l \mid \star \mid \ . \ \mid q/q \mid q//q \mid q[q]$$

In this grammar, l is a label, $\star$ is the wildcard, . denotes the current node. Then / and // denote the child and descendant constructs, and [ ] is a filter expression. In (Miklau and Suciu, 2004) this fragment is denoted as $\texttt{XP}^{\{[],\star,//\}}$. In this more succinct syntax, the above expression can be written as //department/manager. Tree patterns allow for a nice graphical representation, just like XML documents. That is, a tree pattern can be considered as a tree where each node is labeled by a label or by a wildcard, and with two types of edges: child and descendant edges. For instance, the expression .//department[//"Supply"]/member/$\star$ asks for all members of the department of supply. It can be represented as a tree on Figure 2.3. The square node is the output node. With this tree representation, tree patterns admit an equivalent semantic characterization via an *embedding* (Miklau and Suciu, 2004). Let t be a tree pattern $(N', E_/, E_{//}, r', l)$, where $E_/$ and $E_{//}$ are the child and descendant edges respectively, and $T = (N, E, r, \rho)$ an XML tree (the order $<$ is omitted since tree patterns are oblivious to the sibling order). Then a mapping $e : N' \to N$ is called an *embedding* if all of the following conditions are met

(root preservation) $e(r') = r$,

(label preservation) For every $x \in N'$, $\rho(x) = \star$ or $\rho(x) = l(e(x))$,

(child-edge preservation) If $xE_{/}y$, then $e(x)Ee(y)$,

(descendant-edge preservation) If $xE_{//}y$, then $e(x)E^{+}e(y)$.

For complete results on the evaluation problem for XPath, we refer the reader to the overview paper (Benedikt and Koch, 2009).

2.2.3 XPath query containment

The containment problem for various XPath fragments has been a topic of wide interest for the past several years. A polynomial time algorithm for tree patterns without the wildcard based on homomorphism between queries was given in (Amer-Yahia et al., 2002). The main result ofMiklau and Suciu (2004) is the coNP-completeness of containment of tree patterns *with* the wildcard. Almost a complete picture of the containment problem for the XPath fragments with disjunction, in the presence of DTDs and variables was given in (Neven and Schwentick, 2006). Notably, it was shown that with a finite alphabet the containment problem rises to PSPACE. Wood (2003) gives decidability results for various fragments with DTDs and a class of integrity constraints. XPath containment in the presence of dependency constraints was studied in (Deutsch and Tannen, 2001, 2005). A thorough investigation and overview of the results on the containment problem for tree patterns with or without constraints has recently been presented in (Czerwinski et al., 2015).

A closely related problem is XPath satisfiability (Benedikt et al., 2008; Hidders, 2003). Given an XPath expression, the satisfiability problem asks whether there exists a tree (conforming constraints if necessary) such that the result of evaluating the expression on the tree is not empty. Benedikt et al. (2008) contains an almost complete picture for the complexity of the satisfiability problem with or without the presence of constraints for various fragments of XPath. Query containment reduces to XPath satisfiability in fragments with enough expressive power (e.g. with negation and filter expressions).

Afrati et al. (2011) consider the containment problem for tree patterns with general arithmetic comparisons. They add the ability to compare the value of an attribute in two different nodes (note that this is not expressible in Core XPath) and show that containment for this fragment is Π_2^P-complete. As mentioned in the introduction, we extend their coNP result for tree patterns with attribute value comparisons.

We end with some results on tractable (PTIME) containment. As mentioned above, Amer-Yahia et al. (2002) provides a PTIME algorithm for containment of tree patterns without the wildcard. PTIME containment for acyclic conjunctive queries implies tractability for containment of tree patterns without descendant. Moreover, containment for tree patterns without filters is in PTIME as well (Miklau and Suciu, 2004). However, adding attribute value comparisons may raise the complexity. For instance, as shown below in Chapter 3 (Proposition 3.3.4), containment for tree patterns without the wildcard together with equality and inequality attribute comparisons is coNP-hard.

2.2.4 Conjunctive queries interpreted over trees and containment

Conjunctive queries can also be interpreted over unranked trees. In this case the relational schema consists of relation names corresponding to the axes relations $Child$,

$NextSibling$, $Descendant$, $NextSigbling^+$ and $Following$. Then the semantics of conjunctive queries on trees over such schema is defined in the same way as the semantics of conjunctive queries over relational databases.

A systematic study of conjunctive queries interpreted over trees started in (Gottlob et al., 2006), where the central problem was the evaluation problem. The authors established a PTIME and NP dichotomy of the problem. The containment problem for this language was considered in (Björklund et al., 2011), where it was shown to be Π_2^P-complete. The Π_2^P upper bound was shown via the small counter-example property, similar to the one in (Miklau and Suciu, 2004). On the other hand, the Π_2^P lower bound proof heavily relies on the DAG structure of conjunctive queries. Containment of conjunctive queries under schema constraints was studied in (Björklund et al., 2008), where 2EXPTIME-completeness of the problem was established.

2.3 Why and why-not explanations

An increasing number of databases are derived, extracted, or curated from disparate data sources. Consequently, it becomes more and more important to provide data consumers with mechanisms that will allow them to gain an understanding of the data that they are confronted with. An essential functionality towards this goal is the capability to provide meaningful explanations about why data is present or missing form the result of a query. Explanations help data consumers gauge how much trust one can place on the result. Perhaps more importantly, they provide useful information for debugging the query or data that led to incorrect results.

There have been a number of publications on why explanations (that is, for why particular data is present in the query result) and why-not explanations (that is, for why particular data is missing from the query result). Why explanations have been thoroughly studied in the past in the context of *provenance* and *lineage*, see (Cheney et al., 2009). In contrast, models for why-not explanations have only appeared fairly recently. We now review the main approaches to both why and why-not explanations.

2.3.1 Why explanations

The general formulation of the *why-problem* is as follows. Given a database I, a query Q, its computed result $Q(I)$ and a tuple $\bar{a} \in Q(I)$, explain why this tuple is in the query result, i.e., why $\bar{a} \in Q(I)$? One of the first models for why explanations in the context of databases is called *lineage* which has been proposed in (Cui et al., 2000). In this model, the lineage of $\bar{a}$ is a collection of "source" tuples from the database I which "helped" to produce $\bar{a}$. For instance, let I be the database containing two tables T_1 and T_2 as in Figure 2.4. For convenience, each row of the database has a tag. Now suppose we pose the conjunctive query

$$Q'(x, y) \text{ :- } T_1(x, y), T_2(y, z),$$

which has the answer set $Q'(I) = \{(1, \text{A}), (2, \text{B})\}$. Then the lineage of the tuple (2, B) are the tuples t_2 and t_6. The lineage of (1, A), on the other hand, consists of t_1 and $\{t_4, t_5\}$. The latter shows one of the shortcomings of the lineage model. That is, an

T_1

t_1 :	1	A
t_2 :	2	B
t_3 :	3	C

T_2

t_4 :	A	4
t_5 :	A	5
t_6 :	B	6

Figure 2.4: Example database.

explanation might not be as precise as one might like, as it does not specify that the tuples t_4 and t_5 need not be together to witness the tuple (1, A).

In fact, either of the "witnesses" $\{t_1, t_4\}$ or $\{t_1, t_5\}$ would be enough for explaining the tuple (1, A). This is formalized in the notion of why provenance which captures such different witnesses (Buneman et al., 2001). According to this model, an explanation (or a witness) to why $\bar{a} \in Q(I)$ is a part of the database $I' \subseteq I$ that is enough to ensure that $\bar{a}$ is in the output, i.e., $\bar{a} \in Q(I')$. Besides the two witnesses provided above for the query Q', the set $\{t_1, t_4, t_5\}$ is a witness for the tuple (1, A) as well. There might be multiple witnesses, in fact exponentially many of them. For that reason, *why provenance* of $\bar{a}$ in (Buneman et al., 2001) is defined as a particular set of small witnesses, called *witness basis* of $\bar{a}$. The witness basis can be seen as a compact representation of the space of all witnesses, as every element of a witness contains an element from the witness basis. In our example, $\{\{t_1, t_4\}, \{t_1, t_5\}\}$ is the witness basis for (1, A).

Another form for explanations that has been proposed in (Buneman et al., 2001) is "where-provenance". Where-provenance describes the way how the source and the output *locations* are related. A location for a data value is a cell defined by a table name, a tuple in the table and an attribute name. Then the where-provenance of an output value is the location where this valued was copied from. For instance, the where-provenance for the value "2" in the output of Q' is $(T_1, t_2, 1)$, i.e., it is the first attribute of tuple t_2 in table T_1.

The above models for why-provenance and where-provenance do not, however, explain *how* the source tuples contributed to the output tuple. This issue has been addressed in (Green et al., 2007b) using the notion of a *provenance semiring*. According to Green et al. (2007b), the provenance of an output tuple is represented by a polynomial using the semiring operations with integer coefficients where each variable is a source tuple. Such a polynomial hints to the structure of the proof (or the query plan) how the output tuple has been derived. Note that although query plans for equivalent queries are different, their provenance polynomials are always equivalent in the semiring. Moreover, from provenance polynomials we can easily read the why-provenance of an output tuple. As an example, let

$$Q''(x, y) \text{ :- } T_1(x, y), (T_1(x, z) \vee T_2(y, u))$$

be a query, for which the tuple (2, B) is in the output of $Q''(I)$. Then the provenance of this tuple is the polynomial $t_2 \cdot (t_2 + t_6)$. This polynomial hints that t_2 and t_6 contribute to the output of the inner query ($T_1(x, z) \vee T_2(y, u)$) and then t_2 contributes to the join. For comparison, the why-provenance of this tuple according to (Buneman et al., 2001) is the set $\{t_2, t_6\}$.

It is worth to note that in all the models above there could be multiple solutions (explanations), many of which are not useful or redundant to the user. Thus, certain preference criteria are imposed in order to single out "good" or "interesting" explanations. Note also that the above models (except for the lineage model of (Cui et al., 2000)) assume the queries to be monotone. The presence of negation in queries causes some nice properties to disappear. Overall, the principles of provenance in the presence of negation and aggregation have not been fully understood yet (Cheney et al., 2009).

We end with some applications and implementations of the above models of the why problem. Algorithms for computing lineage have been implemented in the WHIPS data warehouse system (Cui and Widom, 2000). Minimal witnesses from (Buneman et al., 2001) have been used in (Buneman et al., 2002) for the view deletion problem, i.e., the problem of finding source tuples to remove in order to remove the tuple from the view expressed as the union of conjunctive queries. Where-provenance has been applied to study annotation propagation in (Buneman et al., 2002), and implemented in an annotation management tool (Bhagwat et al., 2005). An application of how-provenance based on semirings appears in the context of ORCHESTRA (Green et al., 2007a), a collaborative data sharing system.

2.3.2 Why-not explanations

The input for the *why-not problem* is almost the same as for why problem. Given a database I, a query Q, its computed result $Q(I)$ and a tuple $\bar{a} \notin Q(I)$, explain why the tuple $\bar{a}$ is missing from the query result $Q(I)$. As opposed to why explanations in which the output tuple can be explained using the derivation or query plan, for why-not explanations the situation is different. Indeed, in this case we cannot talk about witnesses since there is no derivation to begin with. There have been a number of proposals for why-not explanations in the literature which can be classified into two groups: query-driven and data-driven approaches.

Query-driven approaches. In query-driven approaches, a reason for why a tuple is missing from the query result lies in the query itself. In (Chapman and Jagadish, 2009), an explanation is a query operator (e.g., selection) that prevented the given tuple from appearing in the query result. This approach was defined for the class of unions of conjunctive queries. In (Bidoit et al., 2014a) it was further extended to deal with aggregate queries. One of the shortcomings of the algorithms proposed in these articles is that the generated explanations are dependent on a particular query plan. As a consequence, these algorithms might produce different explanations for the same query and may miss some of them. To tackle this problem, Bidoit et al. (2014b) proposed explanations in the form of polynomials, similarly to the approach of (Green et al., 2007b).

In (Tran and Chan, 2010), an explanation is not an operator, but a relaxation (or *refinement*) of the original query such that it contains the missing tuple while retaining the previous results. As there are many possible refinements, a preference condition is imposed as well. A similar type of explanations are considered in (He and Lo, 2012) for reverse top-k queries and in (Islam et al., 2013) for reverse skyline queries.

Data-driven approach. In data-driven approaches, the assumption is that the query is correct and only the database lacks certain information allowing to infer the needed query results. In (Huang et al., 2008b), an explanation is a sequence of updates to the database such that the missing tuple is present in the query result. In this paper the attention is restricted to conjunctive queries. Herschel and Hernández (2010) extend this approach to the class of unions of conjunctive queries with aggregation and grouping.

Calvanese et al. (2013) consider the why-not question in the context of ontology-based data access (OBDA). In this setting, queries are restricted to be unions of conjunctive queries and defined over a possibly richer vocabulary than the database schema. An explanation is now a database (or an ABox, in description logic notation) such that the tuple is present in the result of the query asked against the combination of the new and the original databases. Calvanese et al. (2013) phrase this problem as an abductive reasoning problem. In such a setting, one imposes a preference (or minimality) condition on the space of possible explanations which allows to leave out trivial explanations. As opposed to the previous papers on why-not explanations whose focus is the problem of producing explanations, the main focus of (Calvanese et al., 2013) is to study the computational complexity of associated problems (e.g., the problem of existence of an explanation).

In Chapter 6 we introduce a new framework for why-not explanations which is neither query-driven nor data-driven. In our approach we do not propose a fix to a query or a database, but rather give a hint to a possible general pattern or a gap in the data. In particular, an explanation is a tuple of concepts that “generalizes” the missing data and thus gives a high level explanation to why this data is missing. As an example, given a database of train connections, and the query asking for pairs of cities reachable from each other, one can see that the tuple ⟨New York, Amsterdam⟩ is missing from the query result. One possible high-level explanation is the tuple $\langle \texttt{AmericanCity}, \texttt{EuropeanCity} \rangle$, which denotes that fact that there is no train connection between *any* two cities of America and Europe, and thus it explains the fact why ⟨New York, Amsterdam⟩ is missing. As in (Calvanese et al., 2013) we impose a preference condition on explanations. More precisely, we prefer most general explanations, i.e., explanations such that there is no strictly more general explanation. For instance, the tuple $\langle \texttt{AmericanCity}, \texttt{EuropeanCity} \rangle$ is strictly more general concept tuple than $\langle \texttt{NorthAmericanCity}, \texttt{DutchCity} \rangle$.

In order to reason about concepts like `AmericanCity` and `EuropeanCity`, we make use of ontologies that provide a formal specification of the relationships between concepts. The use of ontologies to facilitate access to databases is not new. A prominent example is OBDA mentioned earlier, where queries are either posed directly against an ontology, or an ontology is used to enrich a data schema against which queries are posed with additional relations, namely, the concepts from the ontology (Bienvenu et al., 2013; Poggi et al., 2008). Answers are computed based on an open-world assumption and using the mapping assertions and ontology provided by the OBDA specification. However, unlike in OBDA, in our framework we consider queries posed against a database instance under the traditional closed-world semantics, and the ontology is used only to derive why-not explanations.

Part I

Containment Problem for Acyclic Queries

3 Containment for Queries over Trees with Attribute Value Comparisons

In this chapter we address **RQ 1** for expanded conjunctive queries (CQ) and positive XPath (PosXPath) interpreted over trees. Björklund et al. (2011) showed that containment for CQ and PosXPath is respectively Π_2^P and coNP-complete. In this chapter we show that the same problem has the same complexity when we expand these languages with XPath's attribute value comparisons. We show that different restrictions on the domain of attribute values (finite, infinite, dense, discrete) have no impact on the complexity. Making attributes required does have an impact: the problem becomes harder. We also show that containment of tree patterns without the wildcard $*$, which is in PTIME, becomes coNP-hard when adding equality and inequality comparisons.

3.1 Introduction

In this chapter we study the containment problem for positive XPath (PosXPath) and conjunctive queries (CQ) interpreted over finite unranked ordered trees with respect to the axes $Child$, $NextSibling$, $Descendant$, $NextSibling^+$ and $Following$. PosXPath is a large fragment of Core XPath (Gottlob et al., 2005a) that contains all the axes and constructs except negation. Conjunctive queries over trees are an analog of relational conjunctive queries, which correspond to the select-from-where SQL queries in which the where-condition uses only conjunctions of equality comparisons, and are the most widely used query language in practice. A thorough study of the containment problem for CQ over trees has been carried out in (Björklund et al., 2011). Their main result is Π_2^P-completeness of the problem. In fact, conjunctive queries can be reformulated as the positive fragment of Core XPath with path intersection. Thus, the Π_2^P hardness result also holds for the containment problem for this fragment. Inspection of the proof in (Björklund et al., 2011) also indicates that the containment for just PosXPath remains in coNP. This extends the result of Miklau and Suciu (2004), who showed that containment for tree patterns is coNP-complete.

The query language considered in these previous results ignores attributes. However, in many practical scenarios we deal with data that come from numeric domains, such as real or natural numbers. Thus, it is natural to consider conjunctive queries expanded with attribute value comparisons and study basic static analysis problems such as satisfiability and containment. Such an expansion has been considered for Tree Patterns in (Afrati et al., 2011), where a Π_2^P-completeness result for the containment has been established. However, the hardness proof relies on the construct that allows comparisons of attributes of two different nodes, a feature that is not expressible in Core XPath. As a positive counterpart, a coNP upper bound for containment was shown in the case when comparisons are restricted to either so-called left semi-interval or right semi-interval attribute constraints. For an attribute a and constant c, an attribute constraint $(@_a \mathsf{op} c)$ is left semi-interval if $\mathsf{op} \in \{<, \leq, =\}$.

In this chapter we show that essentially the complexity does not increase if positive XPath and conjunctive queries over trees are expanded with *both* left and right semi-intervals constraints together with inequality constraint. Furthermore, the same upper bounds hold when we make certain assumptions on the underlying attribute domain D. That is, we show that all the complexity results still hold for the cases when D is a dense or discrete infinite linear order, with or without endpoints, or a finite linear order. As another result, we show that by requiring at least one attribute to be defined in every node of a tree, the complexity of containment over such trees rises to PSPACE. If, on the other hand, we require attributes to be defined only at nodes with a certain label (which can be expressed in DTDs) the complexity remains in coNP.

All the upper bound results for both PosXPath and CQ are obtained from a suitable polynomial reduction to the containment problem in $\mathsf{PosXPath}^{\neg^s}$ and $\mathsf{UCQ}^{\neg^s}$ (PosXPath and CQ expanded with safe label negation and union) over trees in which nodes may have multiple labels, respectively. Safe label negation is the construct $p \setminus \{q_1, \dots, q_n\}$ which denotes p-labelled nodes that are not labelled with any of the labels $q_1, \dots, q_n$. Table 3.1 summarizes our results.

Organization

The chapter is organized as follows. Section 3.2 contains all the necessary preliminary notions. Section 3.3 contains the main results. In particular, in Subsection 3.3.1 we show that containment for $\mathsf{UCQ}^{\neg^s}$ and $\mathsf{PosXPath}^{\neg^s}$ is in Π_2^P and coNP respectively. Next in Subsection 3.3.2 we consider containment for $\mathsf{CQ}^{@}$ and $\mathsf{PosXPath}^{@}$ and show the same upper bounds by reducing to the previous problem. Then in Subsection 3.3.3 we show that the upper bounds of containment do not change in case of some natural restrictions on the attribute domain. Section 3.3.4 contains lower bounds: containment of tree patterns without wildcard rises from PTIME to coNP when we add equality and inequality comparisons; containment of tree patterns rises from coNP to PSPACE when we add equality and inequality comparisons and interpret them on trees in which at least one attribute is defined at each node (a so-called *required* attribute). We finish the chapter with conclusions and future work.

3.2 Preliminaries

We work with node-labelled ordered unranked finite trees, where the nodes are labeled by finite subsets of the infinite set of labels Σ. Formally, a tree over Σ is a tuple $(N, E, <, r, \rho)$, where N, the set of nodes of the tree, is a prefix closed set of finite sequences of natural numbers, $E = \{(\langle n_1, \dots, n_k\rangle, \langle n_1, \dots, n_k, n_{k+1}\rangle) \mid \langle n_1, \dots, n_{k+1}\rangle \in N\}$ is

	$\mathsf{PosXPath}^{@}$	$\mathsf{CQ}^{@}$
no attributes	coNP (Björklund et al., 2011)	Π_2^P (Björklund et al., 2011)
optional attributes	coNP (Thm. 3.3.2)	Π_2^P (Thm. 3.3.2)
required attributes	PSPACE-hard (Thm. 3.3.3)	PSPACE-hard (Thm. 3.3.3)

Table 3.1: Complexity results for containment of Positive XPath and CQ with attribute value comparisons.

the child relation, the sibling relation $<$ is defined as $\{(\langle n_1, \ldots, n_k\rangle, \langle n_1, \ldots, n_k + 1\rangle) \mid \langle n_1, \ldots, n_k\rangle, \langle n_1, \ldots, n_k + 1\rangle \in N\}$, $r = \langle\rangle$ is the root of the tree, and ρ is the function assigning to each node in N a finite subset of Σ. If for every node n of a tree $\rho(n)$ is singleton, we call such a tree as a *single-labeled* tree. Otherwise, it is *multi-labeled*. A *pointed* tree is a pair T, n, where n is a node in T.

Let A be a set of attribute names and $(D, <)$ a dense linear order without endpoints. Then a tree with attributes from A over Σ is a tuple $(N, E, <, r, \rho, att)$ such that $(N, E, <, r, \rho)$ is a tree over Σ and $att : N \times A \to D$ is a partial function.

By E^+ and $<^+$ we denote the *descendant* and the *following sibling* relations which are transitive closures of the child and sibling relations respectively. We will also use $<_f$ for the *following* relation, i.e., the abbreviation for $(E^{-1})^* \circ <^+ \circ E^*$. For $x, y \in N$ and $R \in \{E, E^+, <, <^+, <_f\}$, by $T \models xRy$ we denote the fact that $(x, y) \in R$.

Positive XPath with attribute value comparisons. We define the syntax of Positive XPath (denoted as $\mathsf{PosXPath}^{@}$) node and path formulas with attribute value comparisons with the following grammar.

$$\begin{array}{lcl} step & ::= & \downarrow \mid \uparrow \mid \leftarrow \mid \rightarrow, \\ \varphi & ::= & p \mid \top \mid @_a \mathsf{op}\, c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\varphi \\ \alpha & ::= & step \mid ?\varphi \mid \alpha;\alpha \mid \alpha \cup \alpha \mid step^+ \mid \rightarrow_{\mathtt{f}} \mid \leftarrow_{\mathtt{p}}, \end{array}$$

where $p \in \Sigma$, $a \in A$, $\mathsf{op} \in \{\leq, \geq, <, >, =, \neq\}$, and $c \in D$.

The semantics of $\mathsf{PosXPath}^{@}$ path formulas α and node formulas φ is defined as follows. Let $T = (N, E, <, r, \rho, att)$ be a tree over Σ with attribute names from A. In a mutual induction we define the relation $[\![\alpha]\!]_T \subseteq N \times N$ and the satisfaction relation $T, n \models \varphi$.

- $[\![\downarrow]\!]_T = E$,
- $[\![\uparrow]\!]_T = E^{-1}$,
- $[\![\rightarrow]\!]_T =<$,
- $[\![\leftarrow]\!]_T =<^{-1}$,
- $[\![\rightarrow_{\mathtt{f}}]\!]_T =<_f$,
- $[\![\leftarrow_{\mathtt{p}}]\!]_T = (<_f)^{-1}$,
- $[\![?\varphi]\!]_T = \{(n, n) \in N \times N \mid T, n \models \varphi\}$,

- $[\![\alpha;\beta]\!]_T = [\![\alpha]\!]_T \circ [\![\beta]\!]_T$,
- $[\![\alpha \cup \beta]\!]_T = [\![\alpha]\!]_T \cup [\![\beta]\!]_T$,
- $[\![\alpha^+]\!]_T = ([\![\alpha]\!]_T)^+$ for $\alpha \in \{\downarrow, \uparrow, \rightarrow, \leftarrow\}$,

and

- $T, n \models \top$,
- $T, n \models p$ iff $p \in \rho(n)$,
- $T, n \models @_a \mathtt{op}\ c$ iff $(D, <) \models att(n, a)\ \mathtt{op}\ c$,
- $T, n \models \varphi \wedge \psi$ iff $T, n \models \varphi$ and $T, n \models \psi$,
- $T, n \models \varphi \vee \psi$ iff $T, n \models \varphi$ or $T, n \models \psi$,
- $T, n \models \langle\alpha\rangle\varphi$ iff there is a node m with $(n, m) \in [\![\alpha]\!]_T$ and $T, m \models \varphi$.

The *step* axes select a pair of nodes that are in the child, parent, next-sibling or previous-sibling relations in the tree. Furthermore, the $\rightarrow_{\mathtt{f}}$ and $\leftarrow_{\mathtt{p}}$ axes select nodes that are in the following and the preceding relations in the tree respectively. Note that $\top$ is the same as the wild card axis.

Sometimes we will write $T \models \varphi$ to denote $T, r \models \varphi$.

Conjunctive queries with attribute value comparisons. Let Var be a set of variables, A a set of attribute names and $(D, <)$ the attribute domain, which is a dense linear order without endpoints. A conjunctive query with attribute value comparisons ($\mathsf{CQ}^@$) over Σ, A and D is a positive existential first-order formula without disjunction in prenex normal form over a set of unary predicates $p(x)$ and $@_a(x)\mathtt{op}\ c$, where $p \in \Sigma$, $x \in Var, c \in D$ and $\mathtt{op} \in \{\leq, \geq, <, >, =, \neq\}$; and the binary predicates $Child$, $Descendant$, $NextSibling$, $NextSibling^+$ and $Following$. If Q is a $\mathsf{CQ}^@$, by $Var(Q)$ we denote the set of variables occurring in Q. By $FVar(Q)$ we denote the set of free variables in Q. If $|FVar(Q)| = k > 0$, we call Q a k-ary conjunctive query. If $|FVar(Q)| = 0$, we call Q a Boolean conjunctive query.

Let Q be a conjunctive query and $T = (N, E, <, r, \rho, att)$ a tree over Σ and attributes from A. A *valuation* of Q on T is a total function $\theta : Var(Q) \rightarrow N$. A valuation is a *satisfaction* if it satisfies the query, that is, every atom of Q is satisfied by the valuation. Satisfaction of an atom in T, given a valuation θ, is defined as follows.

- $T, \theta \models p(x)$ iff $p \in \rho(\theta(x))$,
- $T, \theta \models @_a(x)\mathtt{op}\ c$ iff $(D, <) \models att(\theta(x), a)\ \mathtt{op}\ c$,
- $T, \theta \models Child(x, y)$ iff $T \models \theta(x) E \theta(y)$,
- $T, \theta \models Descendant(x, y)$ iff $T \models \theta(x) E^+ \theta(y)$,
- $T, \theta \models NextSibling(x, y)$ iff $T \models \theta(x) < \theta(y)$,
- $T, \theta \models NextSibling^+(x, y)$ iff $T \models \theta(x) <^+ \theta(y)$,

- $T, \theta \models Following(x, y)$ iff $T \models \theta(x) <_f \theta(y)$

A tree T *models* Q, denoted as $T \models Q$, if there is a satisfaction of Q on T. If $(x_1, \ldots, x_k)$ is the tuple of free variables in Q, then the *answer* of Q over T is the set $answer(Q, T) = \{(\theta(x_1), \ldots, \theta(x_k)) \mid \theta$ is a satisfaction of Q on $T\}$. Note that tuples can be nullary as well. Thus, for a Boolean query Q, $answer(Q, T) = \{\langle\rangle\}$ (and we say Q is *true* on T) if there is a satisfaction of Q on T and $answer(Q, T) = \emptyset$ (and we say Q is *false* on T) otherwise.

We also consider unions of conjunctive queries with attribute value comparisons, denoted as $\mathsf{UCQ}^{@}$. These are formulas of the form $\bigvee_{i=1}^{n} Q_i$, where $Q_i \in \mathsf{CQ}^{@}$. The semantics of these formulas is defined in the obvious way.

$\mathsf{PosXPath}^{@}$ *formulas as* $\mathsf{CQ}^{@}$ *formulas with disjunction.* Every $\mathsf{PosXPath}^{@}$ formula can be translated into an equivalent $\mathsf{CQ}^{@}$ formula with disjunction in linear time. The translation is a standard translation of XPath into first-order logic language. It is defined by induction on the complexity of path and node formulas of $\mathsf{PosXPath}^{@}$ as follows. Note that the translation can be easily modified to yield a translation into the three variable fragment of first order logic.

$$\begin{array}{lcl}
TR_{xy}(\downarrow) & = & Child(x, y) \\
TR_{xy}(\uparrow) & = & Child(y, x) \\
TR_{xy}(\rightarrow) & = & NextSibling(x, y) \\
TR_{xy}(\leftarrow) & = & NextSibling(y, x) \\
TR_{xy}(?\varphi) & = & x = y \wedge TR_x(\varphi) \\
TR_{xy}(\alpha_1; \alpha_2) & = & \exists z.(TR_{xz}(\alpha_1) \wedge TR_{zy}(\alpha_2)) \\
 & & \text{where } z \text{ is a fresh variable.} \\
TR_{xy}(\alpha_1 \cup \alpha_2) & = & TR_{xy}(\alpha_1) \vee TR_{xy}(\alpha_2) \\
TR_{xy}(\downarrow^+) & = & Descendant(x, y) \\
TR_{xy}(\uparrow^+) & = & Descendant(y, x) \\
TR_{xy}(\rightarrow^+) & = & NextSibling^+(x, y) \\
TR_{xy}(\leftarrow^+) & = & NextSibling^+(y, x) \\
TR_{xy}(\rightarrow_{\mathtt{f}}) & = & Following(x, y) \\
TR_{xy}(\leftarrow_{\mathtt{p}}) & = & Following(y, x)
\end{array}$$

$$\begin{array}{lcl}
TR_x(p) & = & p(x) \\
TR_x(@_a \mathtt{op}\ c) & = & @_a(x) \mathtt{op}\ c \\
TR_x(\top) & = & \top \\
TR_x(\varphi_1 \wedge \varphi_2) & = & TR_x(\varphi_1) \wedge TR_x(\varphi_2) \\
TR_x(\varphi_1 \vee \varphi_2) & = & TR_x(\varphi_1) \vee TR_x(\varphi_2) \\
TR_x(\langle\alpha\rangle\varphi) & = & \exists y.(TR_{xy}(\alpha) \wedge TR_y(\varphi)), \\
 & & \text{where } y \text{ is a fresh variable.}
\end{array}$$

Query graphs and embeddings. It is convenient to consider $\mathsf{CQ}^{@}$ and $\mathsf{PosXPath}^{@}$ without path union and disjunction in the node formulas as graphs (Gottlob et al., 2006).

By Σ_A we denote the attribute labels of the form $@_a \mathtt{op}\ c$, where $a \in A, c \in D$ and $\mathtt{op} \in \{\leq, \geq, <, >, =, \neq\}$.

Definition 3.2.1 (Graph query). *Let Q be a* $\mathsf{CQ}^{@}$. *Then* $G_Q = (V, E, E^+, <, <^+, <_f, \rho, \rho_{att})$, *where V is the set of nodes, $R \subseteq V \times V$ for $R \in \{E, E^+, <, <^+, <_f\}$, $\rho : V \to 2^{\Sigma}$, $\rho_{att} : V \to 2^{\Sigma_A}$, is a graph query of Q if the following holds.*

- $V = Var(Q)$,
- *$p \in \rho(x)$ iff $p(x)$ occurs as a conjunct in Q,*
- *$@_a \mathsf{op}\ c \in \rho_{att}(x)$ iff $@_a(x) \mathsf{op}\ c$ occurs as a conjunct in Q,*
- *$(x, y) \in E$ iff $Child(x, y)$ occurs as a conjunct in Q,*
- *$(x, y) \in E^+$ iff $Descendant(x, y)$ occurs as a conjunct in Q,*
- *$(x, y) \in <$ iff $NextSibling(x, y)$ occurs as a conjunct in Q,*
- *$(x, y) \in <^+$ iff $NextSibling^+(x, y)$ occurs as a conjunct in Q,*
- *$(x, y) \in <_f$ iff $Following(x, y)$ occurs as a conjunct in Q.*

By $Nodes(G)$ we denote the set of nodes V of G. We write $G_Q \models u_1 R u_2$, to specify that $(u_1, u_2) \in R$ for $R \in \{E, E^+, <, <^+, <_f\}$. Note that for fragments without attribute value comparisons, the value of the labeling function ρ_{att} is always the empty set. In these cases we omit ρ_{att} in query graphs. The semantics of query graphs is given in terms of embeddings, which are essentially valuations for conjunctive queries.

Definition 3.2.2 (Embedding). *Let $T = (N, E, <, r, \rho, att)$ be a tree over Σ with attributes from A and $G = (V, E, E^+, <, <^+, <_f, \rho, \rho_{att})$ a graph query. A function $g : V \to N$ is called an* embedding *of G into T if the following conditions are satisfied.*

- *Edge preserving. For every $u_1, u_2 \in V$, if $G \models u_1 R u_2$ then $T \models g(u_1) R g(u_2)$, for any of the edge relations $R \in \{E, E^+, <, <^+, <_f\}$,*
- *Label preserving. For every $u \in V$, $\rho(u) \subseteq \rho(g(u))$.*
- *Attribute comparison preserving. For every $u \in V$, if $@_a \mathsf{op}\ c \in \rho_{att}(u)$, then $(D, <) \models att(e(u), a)\ \mathsf{op}\ c$.*

Proposition 3.2.1. *Let T be a tree, Q a* $\mathsf{CQ}^{@}$ *query, G_Q its graph query, and θ a function from $Nodes(G_Q)$ to T. Then*

$$T, \theta \models Q \text{ iff } \theta \text{ is an embedding of } G_Q \text{ into } T.$$

Containment. Let Q and P be two k-ary conjunctive queries. We say that P is *contained* in Q, denoted as $P \subseteq Q$, if for every single-labeled tree T, it holds that $answer(P, T) \subseteq answer(Q, T)$. We also say that P is contained in Q over multi-labeled trees and denote it by $P \subseteq_{\mathsf{ML}} Q$ if $answer(P, T) \subseteq answer(Q, T)$ for every multi-labeled tree T.

In this chapter, the central problem is the following decision problem.

- Given two conjunctive queries P and Q,
- Decide: is $P \subseteq Q$?

As pointed out in (Björklund et al., 2011), the containment of k-ary queries can be PTIME reduced to the containment of Boolean conjunctive queries, i.e., queries without free variables. The same reduction works for positive XPath and for containment over multi-labeled trees. Thus, in the remainder of this chapter we concentrate on Boolean query containment only.

Removing the attribute value comparisons. In our upper bound proofs we will treat the attribute value comparisons as ordinary labels, whose interpretation will be restricted by adding constraints. We make that precise using the translation $\widetilde{(\cdot)}$ which maps each $@_a \mathsf{op}\ c$ to a new label $p_{@_a \mathsf{op} c}$. This tranlation can then be homomorphically extended to the translation $\widetilde{(\cdot)}$ from formulas in $\mathsf{PosXPath}^{@}$ and $\mathsf{CQ}^{@}$ over Σ, A and D to formulas without attribute value comparisons in respectively $\mathsf{PosXPath}$ and CQ over the alphabet $\Sigma \cup \{p_{@_a \mathsf{op} c} \mid \mathsf{op} \in \{=, \neq, <, >, \leq, \geq\}, a \in A, c \in D\}$.

PosXPath and CQ with safe negation

We define an expansion of the languages $\mathsf{PosXPath}^{@}$ and $\mathsf{CQ}^{@}$ ($\mathsf{UCQ}^{@}$) with a restricted form of negation. That is, we define formulas of $\mathsf{PosXPath}^{@,\neg^s}$ as formulas of $\mathsf{PosXPath}^{@}$ with the additional node formulas $p \wedge \neg q_1 \wedge \ldots \wedge \neg q_k$, whenever $p, q_1, \ldots, q_k$ are labels from Σ. We define $T, n \models p \wedge \neg q_1 \wedge \ldots \wedge \neg q_k$ iff $p \in \rho(n)$ and $q_i \notin \rho(n), 1 \leq i \leq k$.

Similarly, formulas of $\mathsf{CQ}^{@,\neg^s}$ ($\mathsf{UCQ}^{@,\neg^s}$) are formulas of $\mathsf{CQ}^{@}$ ($\mathsf{UCQ}^{@}$) expanded with the construct $p(x) \wedge \neg q_1(x) \wedge \ldots \wedge \neg q_k(x)$, where $x \in Var$ and $p, q_1, \ldots, q_k \in \Sigma$ with semantics: $T, \theta \models p(x) \wedge \neg q_1(x) \wedge \neg q_k(x)$ iff $p \in \rho(\theta(x))$ and $q_i \notin \rho(\theta(x))$, for every $1 \leq i \leq k$.

For a formula from $\mathsf{CQ}^{@,\neg^s}$ its corresponding graph query is defined in the same way as in Definition 3.2.1 with the addition that nodes can have negative labels. The notion of an embedding can also be extended for $\mathsf{CQ}^{@,\neg^s}$. The additional clause that has to be added to Definition 3.2.2 requires preservation of negated labels:

- For every $u \in V$, if $\neg p \in \rho(u)$ then $p \notin \rho(g(u))$.

By $\mathsf{PosXPath}^{\neg^s}$, $\mathsf{CQ}^{\neg^s}$ and $\mathsf{UCQ}^{\neg^s}$ we denote the fragments of $\mathsf{PosXPath}^{@,\neg^s}$, $\mathsf{CQ}^{@,\neg^s}$ and $\mathsf{UCQ}^{@,\neg^s}$ without attribute value comparisons respectively.

3.3 Containment of PosXPath$^{@}$ and CQ$^{@}$

This section contains the main result of this chapter. First, in Section 3.3.1 we show that containment for $\mathsf{PosXPath}$ and CQ expanded with safe negation are in coNP and Π_2^P respectively. Next we show that containment for these fragments expanded with attribute value comparisons remains the same by a polynomial reduction to the corresponding fragments without attribute value comparisons. This result holds under the assumption that attribute values come from a dense linear order without endpoints. In Section 3.3.3 we show that imposing different constraints on the linear domain of attribute values does not impact the complexity. However, making attributes required everywhere in a tree increases the complexity of containment, as shown in Section 3.3.4.

3.3.1 Containment of Positive XPath and CQs with safe negation

In Subsection 3.3.2 we will reduce the complexity of the containment problem for $\mathsf{PosXPath}^{@}$ and $\mathsf{CQ}^{@}$ to that of $\mathsf{PosXPath}^{\neg^s}$ and $\mathsf{UCQ}^{\neg^s}$. The next theorem shows that adding safe negation to PosXPath and UCQ does not make the containment problem harder. The argument is similar to the one in (Björklund et al., 2011), but additional care needs to be taken when we deal with negation.

Theorem 3.3.1. *The containment problem over multi-labeled trees for* $\mathsf{PosXPath}^{\neg^s}$ *and* $\mathsf{UCQ}^{\neg^s}$ *is in* coNP *and* Π_2^P *respectively.*

Proof. In both cases the proof strategy is the same. Throughout the proof we assume that we deal with multi-labeled trees without attributes. Our goal is to show that whenever $\varphi \not\subseteq \psi$, there is a small (polynomial in φ and ψ) counterexample witnessing this fact. In the proof, we start with an arbitrary counterexample T, and shrink it in two steps: in the first step (creating T^*), we roughly restrict T to the image of φ and intermediate nodes. This can still be too large. In the second step we shrink long paths between image nodes.

Let $\varphi = \bigvee_i \varphi_i$ and $\psi = \bigvee_j \psi_j$ be $\mathsf{UCQ}^{\neg^s}$ formulas. Let $T = (N, E, <, r, \rho)$ be a tree such that $T \models \varphi$ and $T \not\models \psi$. Then there exist i and an embedding $e : \mathit{Nodes}(G_{\varphi_i}) \to T$, where G_{φ_i} is the query graph of φ_i. By $e(G_{\varphi_i})$ we denote the image of $\mathit{Nodes}(G_{\varphi_i})$. If $G_{\varphi_i} \models u_1 <_f u_2$, then there must exist nodes x_1 and x_2 such that $e(u_1)(E^{-1})^* x_1 <^+ x_2 E^* e(u_2)$ in T. We call such x_1 and x_2 *knee-nodes* for G.

Our aim is to create a small tree out of T which is still a counterexample. For the first "shrinking step", we color nodes that we must keep. We use three colors: $\{I, V, H\}$.

- Mark the root r with I,
- If $x \in e(G_{\varphi_i})$, mark x with I ("image" nodes),
- If $G_{\varphi_i} \models u_1 <_f u_2$, then there must exist knee-nodes x_1 and x_2. Mark x_1 and x_2 by I too,
- If there exist two nodes x and y marked by I such that $T \models xE^+y$ and there is no node z marked by I with $T \models xE^+z \wedge zE^+y$, then mark all the nodes on the path from x to y by V ("vertical" nodes),
- If there exist two nodes x and y marked by I or V such that $T \models x <^+ y$ and there is no node z marked by I or V with $T \models x <^+ z \wedge z <^+ y$, then mark all the sibling nodes between x and y by H ("horizontal" nodes),
- Let $T^* = (N^*, E^*, <^*, r^*)$ be the substructure of T restricted to the nodes marked by I, H or V.
- Let $\rho^*(n) = \rho(n)$ for $n \in I$ and $\rho^*(n) = \emptyset$ otherwise.

Example 3.3.1. Let T be a tree as in Figure 3.1 and Q the Boolean conjunctive query $\exists xyzw(a(x) \wedge \mathit{Descendant}(x,y) \wedge b(y) \wedge \mathit{Child}(x,z) \wedge c(z) \wedge \mathit{NextSibling}^+(z,w) \wedge d(w))$. The corresponding query graph G_Q is depicted in Figure 3.1, where the downward and horizontal double line arrows denote E^+ and $<^+$ respectively, and the single line arrow denotes E. The embedding e is defined by the dashed arrows. The marking of nodes of T with the colors $\{I, V, H\}$ is depicted in Figure 3.1.

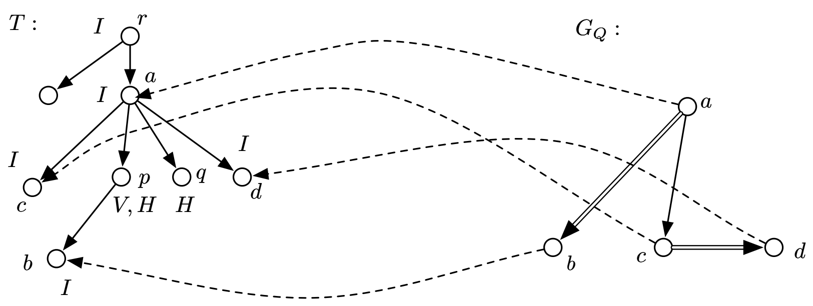


Figure 3.1: The tree T and the query graph corresponding to Q from Example 3.3.1.

Claim 3.3.1. *$T^* = (N^*, E^*, <^*, r^*, \rho^*)$ is a tree and $T^* \models \varphi$ and $T^* \not\models \psi$.*

Proof of Claim 3.3.1. It is easy to check that adding the V and H nodes to I is the minimum needed to ensure that T^* is a tree. First, we argue that $T^* \models \varphi_i$. Since we maintained the image of G_{φ_i} (nodes labeled by I), we have that e is a mapping from $Nodes(G_{\varphi_i})$ to N^*. The node labels are preserved under e since we did not change the labeling of I nodes. Let $\langle x, y\rangle$ be an edge in G_{φ_i}. If $G_{\varphi_i} \models xEy$, or $G_{\varphi_i} \models x < y$ then $e(x)$ and $e(y)$ are in child or next sibling relation in T^* respectively since both nodes are labeled with I and they were in that relation in T. If $G_{\varphi_i} \models xE^+y$ or $G_{\varphi_i} \models x <^+ y$, then $e(x)$ and $e(y)$ are in the corresponding relations in T^* since the intermediate vertical (V) and horizontal (H) nodes were kept. In case $G_{\varphi_i} \models x <_f y$ we have $T^* \models e(x) <_f e(y)$ since we kept the knee-nodes which witness the following relation in T. Thus, we obtain $T^* \models \varphi$.

Now we show $T^* \not\models \psi$. Suppose to the contrary that $T^* \models \psi$. Then there exists an embedding g of G_{ψ_j} into T^* for some j. Because T^* is a substructure of T, g is also a mapping of $Nodes(G_{\psi_j})$ into $Nodes(T)$ that preserves the edge relation. We show that g also preserves the labels. Note that by definition of T^*, the label $\rho^*(n)$ is either equal to $\rho(n)$, when $n \in I$, or empty otherwise. Positive labels are always preserved: $\rho_j(u) \subseteq \rho^*(g(u)) \subseteq \rho(g(u))$ for every node $u \in Nodes(G_{\psi_j})$, where ρ_j is the labeling function of G_{ψ_j}. We show that negative labels are preserved too. Let $u \in Nodes(G_{\psi_j})$. If $\neg p \in \rho_j(u)$, we have that $p \notin \rho^*(g(u))$, since g preserves negative labels. Since negation is safe, there must exist a label $q \in \rho_j(u)$, which implies $q \in \rho^*(g(u))$, and, thus, $\rho^*(g(u))$ is not empty. In this case $\rho^*(g(u)) = \rho(g(u))$, and thus $p \notin \rho(g(u))$ as required. Thus, $T \models \psi$, which is a contradiction since T is a counterexample for $\varphi \not\subseteq \psi$. □

Next, we prove two crucial lemmas. In particular, the following lemma claims that if we have a tree T with a long enough non-branching vertical path, where each node has the empty label, and a query Q with $T \models Q$, then the path can be extended even more while preserving the fact that Q is true in the tree. We use the contrapositive of the lemma to *shrink* such long paths while keeping the query ψ *false* in the smaller tree. The same reasoning applies for horizontal paths.

Lemma 3.3.1 (V-path)**.** *Let G be a query graph with labels from Σ and $T = (N, E, <, \rho, r)$ a tree such that there is an embedding of G into T. Suppose $u_1 E u_2 E \dots E u_n$ is a path in T, such that*

- *$\rho(u_i) = \emptyset$, for every $i \in \{1, \dots, n\}$,*
- *If $T \models u_i E x$, then $x = u_{i+1}$ for $i < n$,*
- *$n > |Nodes(G)|$.*

Let $\hat{T}$ be the tree obtained from T by inserting a node with the empty label in the middle of the path, i.e., by making u_m the parent and u_{m+1} a child of the new node, where $n = 2m$ (when n is even) or $n = 2m - 1$ (when n is odd). Then there exists an embedding from G into $\hat{T}$.

Proof. Let $G = (V, E, E^+, <, <^+, <_f, \rho)$ be the given query graph and g an embedding of G into T. Since the length n of the path is strictly greater than the number of nodes in G, there must exist an index $k \leq n$ such that $u_k \notin g(V), k \in \{1, \dots, n\}$. Let $T' = (N', E', <', \rho', r')$ be the tree defined as follows:

- $r' = r$,
- $N' = N \cup \{u'_k\}, u'_k \notin N$,
- $E' = (E \setminus \{(u_k, x) \in E \mid x \in N\}) \cup \{(u_k, u'_k)\} \cup \{(u'_k, x) \mid T \models u_k E x\}$,
- $<' = <$,
- For every node $v \in N$, $\rho'(v) = \rho(v)$, and $\rho'(u'_k) = \emptyset$.

We prove that in fact the same g is an embedding of G into T'.[1] First, from the definition of T' we obtain the following properties.

Claim 3.3.2. *Let T be from the statement of Lemma 3.3.1 and T' as defined above. Then*

(i) If $T \models x E^+ y$, then $T' \models x E'^+ y$,

(ii) If $x \neq u_k$ and $T \models x E y$, then $T' \models x E' y$,

(iii) If $T \models x < y$ (resp. $T \models x <^+ y$ and $T \models x <_f y$), then $T' \models x <' y$ (resp. $T' \models x <'^+ y$ and $T' \models x <'_f y$).

Proof of Claim 3.3.2. All items except (i) are immediate by the definition of E'. For (i), let $T \models x E^+ y$. We then consider two cases: First suppose $T \models u_k E y$. Then $T \models x E^* u_k$ and thus $T' \models x E'^* u_k$. Since $T' \models u_k E' u'_k$ and $T' \models u'_k E' y$, we have $T' \models x E'^+ y$. In the case that $T \not\models u_k E y$, we obtain $T' \models x E'^+ y$ by the definition of E'. □

[1]Note that the defined T' is isomorphic to the "real" intended tree T' where the nodes are sequences of natural numbers. Here, we treat g as the old g composed with the isomorphism.

We prove that $g : V \to N'$ is an embedding.

Preservation of labels follows from the fact that the image of g is in N and g is an embedding of G into T and thus preserves labels. We show that g still preserves the edge relations. Let xEy hold in G. Then it holds that $g(x) \neq u_k$ as u_k is not in the image of g. Since g is an embedding of G into T, it holds that $T \models g(x)Eg(y)$. Then by Claim 3.3.2(ii), it holds that $T' \models g(x)E'g(y)$.

Let $G \models xE^+y$. Since g is an embedding of G into T, we have $T \models g(x)E^+g(y)$. By Claim 3.3.2 (i), it follows that $T' \models g(x)E'^+g(y)$. Preservation of the relations $<$, $<^+$ and $<_f$ under g follow from Claim 3.3.2 (iii).

Now let $\hat{T}$ be the tree defined in the statement of the Lemma. Formally, $\hat{T} = (\hat{N}, \hat{E}, \hat{<}, \hat{\rho}, \hat{r})$ is defined as follows.

- $\hat{r} = r$,
- $\hat{N} = N \cup \{u'_m\}, u'_m \notin N$,
- $\hat{E} = (E \setminus \{(u_m, x) \in E \mid x \in N\}) \cup \{(u_m, u'_m)\} \cup \{(u'_m, x) \mid T \models u_m E x\}$,
- $\hat{<} = <$,
- For every node $v \in N$, $\hat{\rho}(v) = \rho(v)$, and $\hat{\rho}(u'_m) = \emptyset$.

The trees $\hat{T}$ and T' are isomorphic. Recall the indexes m and k from the definitions of $\hat{T}$ and T'. We define a mapping $f : N' \to \hat{N}$ as follows.

- If $m \leq k$, then $f(v) = \begin{cases} v & \text{if } v \in N \setminus \{u_{m+1}, \dots, u_k\}, \\ u'_m & \text{if } v = u_{m+1}, \\ u_{i-1} & \text{if } v = u_i, m+1 < i \leq k, \\ u_k & \text{if } v = u'_k. \end{cases}$

- If $m > k$, then $f(v) = \begin{cases} v & \text{if } v \in N \setminus \{u_{k+1}, \dots, u_m\}, \\ u_{k+1} & \text{if } v = u'_k, \\ u_{i+1} & \text{if } v = u_i, k+1 \leq i < m, \\ u'_m & \text{if } v = u_{m+1}. \end{cases}$

The function f is onto and 1-1. We show that the V-paths in T' and $\hat{T}$ are isomorphic. We consider the case $m \leq k$, the other case is similar. Let $T' \models uE'v$ for $u, v \in \{u_1, \dots, u_k, u'_k, u_{k+1}, u_{k+2}, \dots, u_n\}$. We need to show that $T' \models uE'v$ iff $\hat{T} \models f(u)\hat{E}f(v)$. There are the following possible cases.

- $u = u_i$ and $v = u_{i+1}$ with $1 \leq i < m$ or $k < i < n$. In this case $f(u_j) = u_j, j \in \{i, i+1\}$. By definition of T' and $\hat{T}$ it holds that $T' \models u_i E' u_{i+1}$ and $\hat{T} \models u_i \hat{E} u_{i+1}$. Thus, $T' \models uE'v$ iff $\hat{T} \models f(u)\hat{E}f(v)$.

- $u = u_m$ and $v = u_{m+1}$. In this case $f(u_m) = u_m$ and $f(u_{m+1}) = u'_m$. By definition, it holds that $T' \models u_m E' u_{m+1}$ and $\hat{T} \models u_m \hat{E} u'_m$. Thus, $T' \models uE'v$ iff $\hat{T} \models f(u)\hat{E}f(v)$.

- $u = u_{m+1}$ and $v = u_{m+2}$. In this case $f(u_{m+1}) = u'_m$ and $f(u_{m+2}) = u_{m+1}$. By definition, it holds that $T' \models u_{m+1}E'u_{m+2}$ and $\hat{T} \models u'_m\hat{E}u_{m+1}$. Thus, $T' \models uE'v$ iff $\hat{T} \models f(u)\hat{E}f(v)$.
- $u = u_i$ and $v = u_{i+1}$ with $m + 1 < i < k$. In this case, $f(u_i) = u_{i-1}$ and $f(u_{i+1}) = u_i$. By definition, it holds that $T' \models u_iE'u_{i+1}$ and $\hat{T} \models u_{i-1}\hat{E}u_i$. Thus, $T' \models uE'v$ iff $\hat{T} \models f(u)\hat{E}f(v)$.
- $u = u_k$ and $v = u'_k$. In this case, $f(u_k) = u_{k-1}$ and $f(u'_k) = u_k$. By definition, it holds that $T' \models u_kE'u'_k$ and $\hat{T} \models u_{k-1}\hat{E}u_k$. Thus, $T' \models uE'v$ iff $\hat{T} \models f(u)\hat{E}f(v)$.
- $u = u'_k$ and $v = u_{k+1}$. In this case, $f(u'_k) = u_k$ and $f(u_{k+1}) = u_{k+1}$. By definition, it holds that $T' \models u'_kE'u_{k+1}$ and $\hat{T} \models u_k\hat{E}u_{k+1}$. Thus, $T' \models uE'v$ iff $\hat{T} \models f(u)\hat{E}f(v)$.

Since $f(v) = v$ for every $v \in N \setminus \{u_1, \ldots, u_n, u'_k, u'_m\}$ and $\rho(v) = \emptyset$ for every $v \in \{u_1, \ldots, u_n, u'_k, u'_m\}$, the labels are preserved as well.

Thus, the mapping $f \circ g$ is an embedding of G into $\hat{T}$. □

Analogous to the above lemma for V-paths, we formalize one for H-paths. The crucial properties of H-paths are that their labels are empty and that all nodes in the path are leafs. We omit the proof.

Lemma 3.3.2 (H-path). *Let G be a query graph with labels from Σ and $T = (N, E, <, \rho, r)$ an ordered tree such that there is an embedding of G into T. Suppose T has a horizontal path $v_1 < v_2 < \ldots < v_n$ and v is their parent in T, where*

- *$\rho(v_i) = \emptyset$ for every $i \in \{1, \ldots, n\}$,*
- *$V_i = \{u \mid T \models v_iEu\} = \emptyset$ for every $i \in \{1, \ldots, n\}$,*
- *$n > |Nodes(G)|$.*

Let $\hat{T}$ be the tree obtained from T by inserting a node with the empty label in the middle of the horizontal path, i.e., by making v_m the predecessor and v_{m+1} a the successor of the new node, where $n = 2m$ (when n is even) or $n = 2m - 1$ (when n is odd). Then there exists an embedding from G into $\hat{T}$.

The proof of Theorem 3.3.1 relies on the small tree property which follows from the two lemmas above. We first show how, using Lemma 3.3.1, we can reduce the number of V-nodes. Let G_{ψ_j} be the query graph of maximal number of nodes among all G_{ψ_i}. Let $u_1Eu_2 \ldots Eu_n$ be a V-path in T^* of length greater than $|Nodes(G_{\psi_j})| + 1$. Then we remove the node u_m, where $n = 2m$ (i.e., if n is even) or $n = 2m + 1$ (i.e., if n is odd), from T^*, and make u_{m+1} to be the child of u_{m-1}. Let T^{**} be the resulting tree. We claim that $T^{**} \models \varphi$ and $T^{**} \not\models \psi$. The former follows from the fact that we did not change I-nodes in T^{**}. For the latter, suppose $T^{**} \models \psi$. Then there exists an embedding $g : G_{\psi_i} \to T^{**}$ for some i. Since $n - 1 > |Nodes(G_{\psi_j})| \geq |Nodes(G_{\psi_i})|$, we can apply Lemma 3.3.1 to show that there is an embedding of G_{ψ_i} to T^*, which contradicts to the fact $T^* \not\models \psi$.

Thus, we can iteratively apply the same argument to make long V-paths shorter and while preserving the fact that $T^* \not\models \psi$ and $T^* \models \varphi$. Similar for H-paths, if they are longer than $|Nodes(G_{\psi_j})| + 1$, we can apply Lemma 3.3.2 to shorten them.

Let us find out how the size of the small tree is bounded. The number of I nodes in T^* is bounded by $|Nodes(G_{\varphi_i})|$. Each I node has at most one V path above it, one H path to its right, and one H path through its children. The number of nodes in all these paths is, by the argument above, maximally $|Nodes(G_{\psi_j})| + 1$. Thus after repeated application of the Lemmas to T^* the resulting size is bounded by $O(|\varphi| \cdot |\psi|)$.

A Π_2^P algorithm for deciding the UCQ$^{\neg^s}$ containment then works as follows. It first guesses a tree T of size $O(|\varphi| \cdot |\psi|)$ and then checks in NP if $T \models \varphi$ and in CONP if $T \not\models \psi$. The CONP algorithm for PosXPath$^{\neg^s}$ works similarly. It also guesses a tree T of polynomial size and checks if $T \models \varphi$ and $T \not\models \psi$ which can be done in PTIME (Gottlob et al., 2005a). $\square$

Note that the safeness condition for negation turns out to be crucial. Indeed, in Chapter 5 we will see that containment for tree patterns with unrestricted label negation is already PSPACE-complete.

3.3.2 Adding attributes

Now we are ready to provide upper bounds for our fragments with attribute value comparisons.

Theorem 3.3.2. *The containment problem over trees with attributes is*

- *in* CONP *for* PosXPath$^{@,\neg^s}$,
- *in* Π_2^P *for* UCQ$^{@,\neg^s}$.

Given the containment problem $\varphi \subseteq \psi$ for $\varphi, \psi \in$ PosXPath$^{@,\neg^s}$ (UCQ$^{@,\neg^s}$), we reduce it to the containment problem $\varphi' \subseteq_{\mathsf{ML}} \psi'$ in PosXPath$^{\neg^s}$ (UCQ$^{\neg^s}$), which is known to be in CONP (Π_2^P) by Theorem 3.3.1. Thus Theorem 3.3.2 is a consequence of the following lemma.

Lemma 3.3.3. *Let φ and ψ be* PosXPath$^{@,\neg^s}$ *(*UCQ$^{@,\neg^s}$*) formulas. Then there exist* PTIME *computable* PosXPath$^{\neg^s}$ *(*UCQ$^{\neg^s}$*) formulas φ' and ψ' such that*

$$\varphi \subseteq \psi \textit{ iff } \varphi' \subseteq_{\mathsf{ML}} \psi'.$$

This holds for both single-labeled and multi-labeled trees.

Proof. The idea behind the proof is as follows. We abstract away from arithmetic comparisons by replacing each of them with a new label. These labels have to obey certain constraints, like comparisons do. To this purpose, we define a list of axioms (Figure 3.2) that faithfully encode these constraints.

In case of PosXPath$^{@,\neg^s}$ we define $\varphi' := \widetilde{\varphi}$ and $\psi' := \widetilde{\psi} \vee Ax$, where $\widetilde{(\cdot)}$ replaces comparisons with labels (definition is in Section 3.2) and Ax is the disjunction of the formulas in Figure 3.2. In the definition of Ax, Σ_p, Σ_a and Σ_c are respectively the sets of labels, attributes and constants appearing in φ or ψ. We use the abbreviation $\langle\downarrow^*\rangle\theta = \theta \vee \langle\downarrow^+\rangle\theta$. Note that the formula Ax is in $\mathsf{PosXPath}^{\neg^s}$. In case of $\mathsf{UCQ}^{@,\neg^s}$ the translation $(\cdot)'$ is defined essentially the same. The only difference is that we take $\exists x.TR_x(Ax)$ instead of Ax. Notice that the resulting formulas φ' and ψ' are in $\mathsf{UCQ}^{\neg^s}$.

We first argue that the size of Ax is $O((|\varphi|+|\psi|)^3)$. Since the sets Σ_p, Σ_a and Σ_c are respectively the sets of labels, attributes and constants appearing in φ or ψ, their sizes are bounded by the combined size $|\varphi| + |\psi|$. The axioms in Ax are in fact axiom schemas. Each schema has at most 3 parameters from Σ_p, Σ_a and Σ_c and 1 parameter from the set of possible operators from $\{=, \neq, <, >, \leq, \geq\}$. Thus each schema stands for at most $6 \cdot (|\varphi| + |\psi|)^3$ disjuncts. The number of axioms and their size does not depend on φ and ψ. Whence the size of Ax is bounded by $O((|\varphi| + |\psi|)^3)$.

We give some intuition behind Ax. In order to prove this lemma, we need to show that there is a counterexample for $\varphi \subseteq \psi$ iff there is a counterexample for $\varphi' \subseteq_{\mathsf{ML}} \psi'$. Note that every counterexample tree T for $\varphi' \subseteq_{\mathsf{ML}} \psi'$ must refute every disjunct (axiom) in Ax. Intuitively, the axioms enforce the following properties of T:

- (*Label*): each node has at most one label from Σ_p,
- (*SName*): each attribute of a node can take only at most one value,
- (*MExcl*),(*Eq*): there is no inconsistent comparison,
- (*DNeg*): if a node contains a comparison with a constant, then it must contain a comparison with all other constants from Σ_c,
- (*LEQ1*) – (*LEQGEQ*): the natural interaction between the comparisons with a constant,
- (*Order1*) – (*Order4*): the order is preserved.

The following claim is crucial for constructing a counterexample tree for $\varphi \subseteq \psi$ from a counterexample for $\varphi' \subseteq_{\mathsf{ML}} \psi'$.

Claim 3.3.3. *Let $T = (N, E, <, r, \rho)$ be a multi-labeled tree over Σ' such that $T, r \not\models Ax$. Then for every $a \in \Sigma_a, c \in \Sigma_c$, node $n \in N$, exactly one of the following holds.*

(i) there is no $p_{@_a \mathsf{op} c} \in \rho(n)$ for every $\mathsf{op} \in \{=, \neq, \geq, \leq, <, >\}$,

(ii) there is exactly one $p_{@_a = c} \in \rho(n)$ and for every $c_1 \in \Sigma_c$ it holds that $p_{@_a \mathsf{op} c_1} \in \rho(n)$ iff $D \models c \;\mathsf{op}\; c_1$,

(iii) there is no $p_{@_a = c} \in \rho(n)$ and there exists $c' \in D \setminus \Sigma_c$ such that for every $c_1 \in \Sigma_c$ it holds that $p_{@_a \mathsf{op} c_1} \in \rho(n)$ iff $D \models c' \;\mathsf{op}\; c_1$.

Proof of Claim. Let T be as stated in the Claim, $a \in \Sigma_a$ an attribute name, $c \in \Sigma_c$ a constant, $n \in N$ a node in T. Assume that there exists $p_{@_a \mathsf{op} c}$ in $\rho(n)$. Otherwise, item (i) holds. Assume that op is in fact "=". Because $T, r \not\models Ax$, the formula (*SName*) is false and thus there cannot be another $p_{@_a = c_1}$ in $\rho(n)$.

We show, for every $c_1 \in \Sigma_c$ and $\mathsf{op} \in \{=, \neq, >, <, \geq, \leq\}$, that $p_{@_a \mathsf{op} c_1} \in \rho(n)$ iff $D \models c \;\mathsf{op}\; c_1$.

(i) op is $=$. Assume $p_{@_a = c_1} \in \rho(n)$, then we have $c = c_1$ by (*SName*). The converse implication holds since $p_{@_a = c} \in \rho(n)$ by the assumption.

For every $p_i, p_j \in \Sigma_p$, $p_i \neq p_j$:

$$\langle\downarrow^*\rangle(p_i \wedge p_j), \qquad (\textit{Label})$$

For every $a \in \Sigma_a, c, c_1, c_2 \in \Sigma_c$, $c_1 \neq c_2$

$$\begin{aligned}
&\langle\downarrow^*\rangle(p_{@_a=c_1} \wedge p_{@_a=c_2}), && (\textit{SName})\\
&\langle\downarrow^*\rangle(p_{@_a=c} \wedge p_{@_a\neq c}), && (\textit{Eq})
\end{aligned}$$

For every $a \in \Sigma_a, c \in \Sigma_c$ and R, S in $\{<, =, >\}$ with $R \neq S$,

$$\langle\downarrow^*\rangle(p_{@_a Rc} \wedge p_{@_a Sc}), \qquad (\textit{MExcl})$$

For every $a \in \Sigma_a, c, c_1 \in \Sigma_c$ and $R \in \{\neq, \leq, \geq, <, >\}$,

$$\begin{aligned}
&\langle\downarrow^*\rangle(p_{@_a Rc_1} \wedge \neg p_{@_a=c} \wedge \neg p_{@_a>c} \wedge \neg p_{@_a<c}), && (\textit{DNeg})\\
&\langle\downarrow^*\rangle(p_{@_a\leq c} \wedge \neg p_{@_a=c} \wedge \neg p_{@_a<c}), && (\textit{LEQ1})\\
&\langle\downarrow^*\rangle(p_{@_a\geq c} \wedge \neg p_{@_a=c} \wedge \neg p_{@_a>c}), && (\textit{GEQ1})\\
&\langle\downarrow^*\rangle(p_{@_a=c} \wedge \neg p_{@_a\leq c}), && (\textit{LEQ2})\\
&\langle\downarrow^*\rangle(p_{@_a=c} \wedge \neg p_{@_a\geq c}), && (\textit{GEQ2})\\
&\langle\downarrow^*\rangle(p_{@_a<c} \wedge \neg p_{@_a\leq c}), && (\textit{LEQ3})\\
&\langle\downarrow^*\rangle(p_{@_a>c} \wedge \neg p_{@_a\geq c}), && (\textit{GEQ3})\\
&\langle\downarrow^*\rangle(p_{@_a<c} \wedge \neg p_{@_a\neq c}), && (\textit{LNEQ})\\
&\langle\downarrow^*\rangle(p_{@_a>c} \wedge \neg p_{@_a\neq c}), && (\textit{GNEQ})\\
&\langle\downarrow^*\rangle(p_{@_a\neq c} \wedge \neg p_{@_a<c} \wedge \neg p_{@_a>c}), && (\textit{TRI})\\
&\langle\downarrow^*\rangle(p_{@_a\geq c} \wedge p_{@_a\leq c} \wedge \neg p_{@_a=c}), && (\textit{LEQGEQ})
\end{aligned}$$

For every $c_1 < c_2, c_1, c_2 \in \Sigma_c$, add the disjuncts,

$$\begin{aligned}
&\langle\downarrow^*\rangle(p_{@_a<c_1} \wedge \neg p_{@_a<c_2}), && (\textit{Order1})\\
&\langle\downarrow^*\rangle(p_{@_a>c_2} \wedge \neg p_{@_a>c_1}), && (\textit{Order2})\\
&\langle\downarrow^*\rangle(p_{@_a=c_1} \wedge \neg p_{@_a<c_2}), && (\textit{Order3})\\
&\langle\downarrow^*\rangle(p_{@_a=c_2} \wedge \neg p_{@_a>c_1}). && (\textit{Order4})
\end{aligned}$$

Figure 3.2: The disjuncts of the formula Ax from the proof of Lemma 3.3.3.

(ii) op is $\neq$. Assume $p_{@_a\neq c_1} \in \rho(n)$. By *Eq*, it follows that $c_1 \neq c$. Thus, $D \models c \neq c_1$. Conversely, assume $D \models c \neq c_1$. It means that either $c > c_1$ or $c < c_1$. First assume that $c > c_1$. Since $p_{@_a=c} \in \rho(n)$ and $c > c_1$, by (*Order4*) we obtain $p_{@_a>c_1} \in \rho(n)$. Then, by (*GNEQ*), it follows that $p_{@_a\neq c_1} \in \rho(n)$, as desired. Similarly for the case $c < c_1$, using (*Order3*) and (*LNEQ*), we can show $p_{@_a\neq c_1} \in \rho(n)$.

(iii) op is $>$. Assume $p_{@_a > c_1} \in \rho(n)$. We show that $D \models c > c_1$. Suppose the opposite, i.e., either $c = c_1$ or $c_1 > c$. In the first case, it would mean that both $p_{@_a = c}$ and $p_{@_a > c}$ occur in $\rho(n)$, which is a contradiction with (*MExcl*). In the second case, by (*Order3*), both $p_{@_a < c_1}$ and $p_{@_a > c_1}$ are in $\rho(n)$, which is again a contradiction with (*MExcl*).

Now suppose $D \models c > c_1$. Then by (*Order4*), it follows that $p_{@_a > c_1} \in \rho(n)$, as needed.

(iv) op is $<$. Similar to the previous case.

(v) op is $\geq$. Assume $p_{@_a \geq c_1} \in \rho(n)$. If $c_1 = c$, then we immediately obtain $D \models c \geq c_1$. Now suppose $c \neq c_1$ and we show that $D \models c \geq c_1$. Suppose the opposite, i.e., $D \models c < c_1$. Then by (*Order3*), we have $p_{@_a < c_1} \in \rho(n)$. By (*LEQ3*), it implies that $p_{@_a \leq c_1} \in \rho(n)$, which in turn by (*LEQGEQ*) implies that $p_{@_a = c_1} \in \rho(n)$. The latter is a contradiction with (*SName*).

Now assume $D \models c \geq c_1$. This means that either $c = c_1$ or $c > c_1$ in D. In the first case, we have $p_{@_a = c} \in \rho(n)$ by the assumption. Thus by (*GEQ2*), we have that $p_{@_a \geq c} \in \rho(n)$. In the second case, similarly to the case when op is $>$, we can show that $p_{@_a > c_1} \in \rho(n)$. Then by (*GEQ3*), we have $p_{@_a \geq c_1} \in \rho(n)$, as needed.

(vi) op is $\leq$. Similar to the previous case.

Thus we have proved item (ii).

Let us consider (iii). Now there is no $p_{@_a = c} \in \rho(n)$ for any $c \in D$. We define $c_1 = max\{c \mid p_{@_a > c} \in \rho(n)\}$ and $c_2 = min\{c \mid p_{@_a < c} \in \rho(n)\}$. If the former set is empty, we let $c_1 = -\infty$, and if the latter set is empty, we let $c_2 = +\infty$. We claim that at least one of c_1 and c_2 is finite. Indeed, there must be $p_{@_a \mathrm{R} c}$ in $\rho(n)$ for some $R \in \{\neq, <, >, \leq, \geq\}$ since otherwise item (i) would hold. By *DNeg* it follows that either $p_{@_a < c}$ or $p_{@_a > c}$ is in $\rho(n)$, which means that c_1 or c_2 is finite.

We also claim that $c_1 < c_2$. It is trivially true if one of c_1 and c_2 is infinity, thus assume both are finite. If $c_1 = c_2$, then both $p_{@_a > c_1}$ and $p_{@_a < c_1}$ appear in $\rho(n)$ at the same time, which is forbidden due to (*MExcl*). If $c_1 > c_2$, then by (*Order1*), both $p_{@_a > c_1}$ and $p_{@_a < c_1}$ are in $\rho(n)$, which is again forbidden by (*MExcl*).

Now, since $c_1 < c_2$ and the assumption that D is a dense order, there exists $c' \in D$ such that $c_1 < c' < c_2$.

We claim that $c' \notin \Sigma_c$. Suppose the opposite. Then since $p_{@_a = c'} \notin \rho(n)$ by the assumption and the fact that $p_{@_a \mathrm{R} c''} \in \rho(n)$ for some R and c'' (otherwise we would be in case (i)), we obtain either $p_{@_a < c'} \in \rho(n)$ or $p_{@_a > c'} \in \rho(n)$, by (*DNeg*). If $p_{@_a < c'} \in \rho(n)$, then $D \models c' \geq c_2$ which contradicts the fact that $D \models c' < c_2$. Similarly, if $p_{@_a > c'} \in \rho(n)$, then $D \models c' \leq c_1$, which contradicts with $D \models c' > c_1$.

We now show that for every $c \in \Sigma_c$ and $\mathsf{op} \in \{=, \neq, >, <, \geq, \leq\}$, $p_{@_a \mathsf{op} c} \in \rho(n)$ iff $D \models c' \;\mathsf{op}\; c$.

(i) op is $=$. The equivalence holds since for every $c \in \Sigma_c$, there is no $p_{@_a = c}$ in $\rho(n)$ and $D \not\models c' = c$ since $c' \notin \Sigma_c$.

(ii) op is $\neq$. Assume $p_{@_a \neq c} \in \rho(n)$. Then we have that $D \models c' \neq c$ because c' is not in Σ_c. Conversely, assume $D \models c' \neq c$. By (*DNeg*), since there is no $p_{@_a = c}$

in $\rho(n)$ for every $c \in \Sigma_c$ and there is $p_{@_a \mathrm{R} c_1}$ for some c_1, it follows that either $p_{@_a < c} \in \rho(n)$ or $p_{@_a > c} \in \rho(n)$. Applying (*LNEQ*) or (*GNEQ*), respectively, we obtain $p_{@_a \neq c} \in \rho(n)$, as desired.

(iii) op is $>$. Assume $p_{@_a > c} \in \rho(n)$. Then $D \models c' > c$ by definition of c'. Conversely, assume $D \models c' > c$. We show that $p_{@_a > c} \in \rho(n)$. Since there is no $p_{@_a = c} \in \rho(n)$ and there exists $p_{@_a \mathrm{R} c_1} \in \rho(n)$, we obtain either $p_{@_a < c}$ or $p_{@_a > c}$ in $\rho(n)$, by (*DNeg*). The first case is impossible since it would imply $c' < c$ which contradicts the assumption. Thus we have $p_{@_a > c} \in \rho(n)$, as desired.

(iv) op is $<$. Similar to the previous case.

(v) op is $\geq$. If $p_{@_a \geq c} \in \rho(n)$, then by (*GEQ1*) either $p_{@_a = c} \in \rho(n)$ or $p_{@_a > c} \in \rho(n)$. The first case is impossible by the assumption. In the second case, we obtain that $D \models c' > c$ by definition of c'. Thus, $D \models c' \geq c$.

Now assume $D \models c' \geq c$. This means that either $c' = c$ or $c' > c$ in D. The first case is impossible, since $c' \notin \Sigma_c$. In the second case, as with the case op = ">", we can show that $p_{@_a > c} \in \rho(n)$. Then by (*GEQ3*), it holds that $p_{@_a \geq c} \in \rho(n)$.

(vi) op is $\leq$. Similar to the previous case.

This concludes the proof of the claim. □

We now prove that $\varphi \subseteq \psi$ iff $\varphi' \subseteq_{\mathsf{ML}} \psi'$,

($\Rightarrow$) Let $T = (N, E, <, r, \rho)$ be a multi-labeled tree such that $T, r \models \varphi'$ and $T, r \not\models \psi'$. Note that then $T, r \not\models Ax$. Then we define a single-labeled tree $T' := (N, E, <, r, \rho', att)$, where att is a partial function assigning a value in D to a given node and an attribute name, as follows:

- For $p \in \Sigma_p$, $\rho'(n) = p$ iff $p \in \rho(n)$. If there is no $p \in \Sigma_p$ such that $p \in \rho(n)$, we set $\rho'(n) = z$ for a fresh symbol z.
- $att(n,a) = \begin{cases} \text{undefined} & \text{if there is no } p_{@_a \mathrm{op} c_1} \text{ in } \rho(n), \\ c & \text{if } p_{@_a = c} \in \rho(n), \\ c' & \text{from Claim 3.3.3, (iii), otherwise.} \end{cases}$

We claim that T' is well defined. Indeed, (*Label*) ensures that every node is labeled by exactly one label from Σ_p or by z. Morever, the function att is well defined since exactly one of the conditions in the definition of att is fulfilled, according to Claim 3.3.3. By induction, using Claim 3.3.3, we can show that for every $\theta, T, n \models \widetilde{\theta}$ iff $T', n \models \theta$. Thus, it follows $T', r \models \varphi$ and $T', r \not\models \psi$, which was desired.

($\Leftarrow$) Let $T = (N, E, <, r, \rho, att)$ be a single-labeled tree such that $T \models \varphi$ and $T \not\models \psi$. We define the tree $T' := (N, E, <, r, \rho')$, where ρ' is defined as follows:

- For $p \in \Sigma_p$, $p \in \rho'(n)$ iff $\rho(n) = p$,
- $p_{@_a = c} \in \rho'(n)$ iff $att(n,a) = c$,
- $p_{@_a \mathrm{op} c} \in \rho'(n)$ iff $D \models att(n,a)$ op c for op $\in \{\neq, \leq, \geq, <, >\}$, $c \in \Sigma_c$.

It is straightforward to check that T' does not satisfy any of the disjuncts in Ax. Thus, we obtain $T' \models \varphi'$ and $T' \not\models \psi'$.

The same argument goes through for multi-labeled trees, except that we must not include formulas (*Label*) as disjuncts of Ax. □

3.3.3 Restricting the attribute domain

We now show that the same complexity results for the containment problem hold in presence of various restrictions on the domain of attribute values. A linear order D has a smallest (largest) element if there exists $c' \in D$ such that $c' \leq c$ ($c \geq c'$) for every $c \in D$. We say D has an endpoint if there exists a smallest or a largest element in D. D is discrete if any point which has a successor also has an immediate successor. D is dense linear order if for every $x < y$ in D there exists $z \in D$ such that $x < z < y$.

Proposition 3.3.1. *Let D be one of the following linear orders:*

(i) finite,

(ii) discrete,

(iii) dense or discrete with one or two endpoints.

Then the containment problem for $\mathsf{PosXPath}^{@,\neg^s}$ *and* $\mathsf{UCQ}^{@,\neg^s}$ *over single-labeled trees with the domain of attribute values D is in* coNP *and* Π_2^P *respectively.*

Proof. Let φ and ψ be $\mathsf{PosXPath}^{@,\neg^s}$ ($\mathsf{UCQ}^{@,\neg^s}$) formulas over D and $\Sigma_c \subseteq D$, Σ_a and Σ_p the sets of constants, attribute names and labels in Σ appearing in φ or ψ. We then construct in PTime formulas φ' and ψ' over $\Sigma' = \Sigma_p \cup \{p_{@_a \mathsf{op} c} \mid a \in \Sigma_a, c \in \Sigma_c, \mathsf{op} \in \{=, \neq, <, >, \leq, \geq\}\}$ such that

$$\varphi \subseteq \psi \text{ if and only if } \varphi' \subseteq_{\mathsf{ML}} \psi'. \qquad (1)$$

Namely, we take $\varphi' := \widetilde{\varphi}$ and $\psi' := \widetilde{\psi} \vee Ax \vee Ax_k$, where $\widetilde{(\cdot)}$ is defined in Section 3.2, Ax is from Figure 3.2 and $Ax_k, k \in \{(Fin), (Discr), (End)\}$ is constructed according to the cases (i), (ii) and (iii) of the Proposition. Note that the formulas φ' and ψ' in all the cases are in fact $\mathsf{PosXPath}^{\neg^s}$ formulas. In case of $\mathsf{UCQ}^{@,\neg^s}$, the translation $(\cdot)'$ is defined essentially the same. The difference is that we use $\exists x.TR_x(Ax)$ and $\exists y.TR_y(Ax_k)$ instead. Note that the result of $(\cdot)'$ is a union of $\mathsf{CQ}^{\neg^s}$ formulas. The upper bounds then follow from Theorem 3.3.1.

Now we construct the formulas $Ax_k, k \in \{(Fin), (Discr), (End)\}$.

(i). Assume $D = \{c_1 < c_2 < \ldots < c_k\}$ is a finite linear order. We then write down the formulas of $Ax_{(Fin)}$. It is the disjunction of the following formulas. For every $a \in \Sigma_a$, $c \in \Sigma_c$ and $\mathsf{op} \in \{=, \neq, <, >, \leq, \geq\}$:

$$\langle\downarrow^*\rangle(p_{@_a \mathsf{op} c} \wedge \neg p_{@_a = c_1} \wedge \ldots \wedge \neg p_{@_a = c_k}). \qquad (Fin)$$

This axiom enforces that whenever an attribute is defined, its value equals one of $c_i, 1 \leq i \leq k$. The following claim, which is easy to verify using (*Fin*), is crucial.

Claim 3.3.4. *Let $T = (N, E, <, r, \rho)$ be a multi-labeled tree over Σ' such that $T, r \not\models Ax \vee Ax_{(Fin)}$. Then for every $a \in \Sigma_a, c \in \Sigma_c$, node $n \in N$ and* $\mathsf{op} \in \{=, \neq, \geq, \leq, <, >\}$*, exactly one of the following holds:*

(i) there is no $p_{@_a = c} \in \rho(n)$,

(ii) there is exactly one $p_{@_a=c} \in \rho(n)$ *and for every* $c_1 \in \Sigma_c$ *it holds that* $p_{@_a \mathsf{op} c_1} \in \rho(n)$ *iff* $D \models c \ \mathsf{op} \ c_1$.

Now we prove the equivalence (*1*). For the direction from left to right, given a multi-labeled tree T over Σ' such that $T \models \varphi'$ and $T \not\models \psi'$, we construct a single-labeled tree with attributes T' as it was done in Lemma 3.3.3. The only difference is in the definition of the attribute function att. In our case we take

$$att(n,a) = \begin{cases} \text{undefined} & \text{if there is no } @_a \mathsf{op}\ c \text{ in } \rho(n), \mathsf{op} \in \{=, \neq, <, >, \geq, \leq\}, c \in \Sigma_c \\ c & \text{if } @_a = c \in \rho(n). \end{cases}$$

Using Claim 3.3.4 we can show that for every $n \in T$, θ over Σ, A and D, $T, n \models \widetilde{\theta}$ iff $T', n \models \theta$.

The direction from right to left of (*1*) can be proved exactly as in Lemma 3.3.3.

(ii). D is a discrete linear order. We assume that D is infinite, as the finite case is covered by the case (i). We take $Ax_{(Discr)}$ as the disjunction of the following formulas. For every $a \in \Sigma, c_1, c_2 \in \Sigma_c$ such that $c_1 < c_2$ in D and there is no c' in D with $c_1 < c' < c_2$,

$$\langle \downarrow^* \rangle (p_{@_a > c_1} \wedge p_{@_a < c_2}), \qquad (\textit{Discr})$$

This axiom enforces the requirement that a value for a-attribute cannot be between an element in D and its immediate successor.

Similarly to Lemma 3.3.3 we can show that the reduction is correct. To this purpose we need the claim which is a reformulation of Claim 3.3.3 where instead of Ax we take $Ax \vee Ax_{(Discr)}$ and D is the discrete linear order.

We highlight the difference with the proof of Claim 3.3.3. The only nontrivial difference is item (iii). Assume the conditions (i) and (ii) of Claim 3.3.3 do not hold for $a \in A$ and $n \in T$. We define $c_1 = max\{c \mid p_{@_a > c} \in \rho(n)\}$ and $c_2 = min\{c \mid p_{@_a < c} \in \rho(n)\}$. As in Claim 3.3.3 we can show that $D \models c_1 < c_2$. Having that, there exists c' such that $c_1 < c' < c_2$. Indeed, suppose the opposite. Then both $p_{@_a > c_1}$ and $p_{@_a < c_2}$ are in $\rho(n)$ and c_2 is the immediate successor of c_1 in D, which is a contradiction with (*Discr*). It follows from (*DNeg*) that $c' \notin \Sigma_c$. Moreover, for every $c'' \in \Sigma_c$, $p_{@_a \mathsf{op} c''} \in \rho(n)$ iff $D \models c' \ \mathsf{op} \ c''$. This can be verified as it was done in Claim 3.3.3. Thus we have proved the claim. Having this claim at hand, we can prove the equivalence (*1*) in the same way as in Lemma 3.3.3.

(iii). D is dense or discrete linear order with one or two endpoints. If D is dense, take $Ax_{(End)}$ as the disjunction of the following formulas:

If D has the least endpoint c_l, for every $a \in \Sigma_a$:

$$\langle \downarrow^* \rangle p_{@_a < c_l}. \qquad (\textit{LEnd})$$

If D has the greatest endpoint c_g, for every $a \in \Sigma_a$:

$$\langle \downarrow^* \rangle p_{@_a > c_g}. \qquad (\textit{REnd})$$

In case D is discrete linear order, $Ax_{(End)}$ additionally has (*Discr*) as a disjunct.

The axioms (*LEnd*) and (*REnd*) enforce the requirement that attributes cannot take their values outside of the bounds in D. As in the previous case, we can prove the variant of Claim 3.3.3, where we consider $Ax_{(End)}$ and D a dense or discrete linear order with one or two endpoints. We do not spell out the proof, but the crucial difference is that in item (iii) axioms (*LEnd*) and (*REnd*) ensure the fact that c' is chosen within the interval $[c_l, c_g]$.

Having this claim, we can prove the equivalence (*1*) in the same way as in Lemma 3.3.3. Clearly, the constructed φ' and ψ' are PTIME computable from φ and ψ. □

3.3.4 Lower bounds

In this section we show a number of lower bounds on containment for $\mathsf{CQ}^{@}$ and $\mathsf{PosXPath}^{@}$. The following lower bound was shown in (Björklund et al., 2011).

Proposition 3.3.2 (Björklund et al. (2011))**.** *Containment is Π_2^P-hard for $\mathsf{CQ}(Child, Descendant)$, i.e., conjunctive queries that use only the predicates $Child$ and $Descendant$.*

For $\mathsf{PosXPath}^{@}$, the CONP lower bound for containment follows from hardness of containment for tree patterns (Miklau and Suciu, 2004), which is a fragment of $\mathsf{PosXPath}^{@}$. In order to compare our results to those in (Miklau and Suciu, 2004), we follow their notation. Let $\mathsf{XP}^{\{[\,],*,//\}}$ denote the fragment of $\mathsf{PosXPath}$ without union and disjunction, only the $\downarrow$ step, and no occurence of the following and preceding axes. These are called *tree patterns* in the literature. Let $\mathsf{XP}^{\{[\,],//\}}$ denote $\mathsf{XP}^{\{[\,],*,//\}}$ in which no wildcard (denoted by $\top$ in $\mathsf{PosXPath}$) occurs.

Containment of $\mathsf{XP}^{\{[\,],//\}}$ and $\mathsf{XP}^{\{[\,],*,//\}}$ patterns is in PTIME and CONP-complete, respectively. Let $\mathsf{XP}^{\{[\,],//\}}_{=,\neq}$ and $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$ denote the expansions of $\mathsf{XP}^{\{[\,],//\}}$ and $\mathsf{XP}^{\{[\,],*,//\}}$ with equality and inequality attribute value comparisons, respectively. We show that containment of $\mathsf{XP}^{\{[\,],//\}}_{=,\neq}$ patterns becomes CONP hard. Containment of $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$ patterns becomes PSPACE hard when interpreted over trees with at least one required attribute.

The following property is used in our lower bound arguments. The proof can be found in (Miklau and Suciu, 2004, Lemma 3).

Proposition 3.3.3. *Let L be $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$ or $\mathsf{XP}^{\{[\,],//\}}_{=,\neq}$. Let φ be an L formula and Δ a finite set of L formulas. Then there are* PTIME *computable L formulas φ' and ψ' such that*

$$\varphi \subseteq \bigvee \Delta \text{ iff } \varphi' \subseteq \psi'.$$

The same holds for the case of multi-labeled trees.

Proposition 3.3.4. *The containment problem for $\mathsf{XP}^{\{[\,],//\}}_{=,\neq}$ is* CONP*-hard.*

Proof. We reduce the 3SAT problem to the non-containment problem in $\mathsf{XP}^{\{[\,],//\}}_{=,\neq}$.

Firstly, we can use a union of tree patterns on the right side of the containment problem, due to Proposition 3.3.3.

Let Q be the conjunction of clauses $C_i = (X_1^i \vee X_2^i \vee X_3^i), 1 \leq i \leq k$ over the variables $\{x_1, \ldots, x_n\}$, where X_j^i are literals. From Q, we construct in PTIME two formulas over the signature $\Sigma = \{r, b\}$, attribute names $A = \{a_1, \ldots, a_n\}$ and an attribute domain D containing values $\{0, 1, 2\}$ as follows.

We define

$$\varphi := r \wedge \langle\downarrow\rangle(b \wedge @_{a_1} \neq 2 \wedge \ldots \wedge @_{a_n} \neq 2)^2$$

and

$$\psi := \bigvee_{i=1}^{k} \langle\downarrow\rangle(b \wedge B_1^i \wedge B_2^i \wedge B_3^i),$$

where $B_j^i = (@_{a_l} = 0)$ iff $X_j^i = x_l$ in C_i and $B_j^i = (@_{a_l} \neq 0)$ iff $X_j^i = \neg x_l$ in C_i.

We claim that Q is satisfiable if and only if $\varphi \not\subseteq \psi$. First assume that Q is satisfiable, i.e., there is a variable assignment $V : \{x_1, \ldots, x_n\} \to \{0, 1\}$ such that $V \models Q$. We then define the following tree $T = (\{v_1, v_2\}, \{(v_1, v_2)\}, v_1, \rho, att)$, where the labeling ρ is defined as $\rho(v_1) = \{r\}, \rho(v_2) = \{b\}$ and $att(v_2, a_l) = 1$ iff $V(x_l) = 1$ and $att(v_2, a_l) = 0$ iff $V(x_l) = 0$ for every $l, 1 \leq l \leq n$. Clearly, T satisfies φ. Suppose $T, v_1 \models \psi$. This means there exists an index i such that $T, v_1 \models \langle\downarrow\rangle(b \wedge B_1^i \wedge B_2^i \wedge B_3^i)$, which implies $T, v_2 \models B_j^i, j = 1, 2, 3$. Hence, by the definition of the attribute function we obtain that if $B_j^i = (@_{a_l} = 0)$, then $V(X_j^i) = V(x_l) = 0$ and, similarly, if $B_j^i = (@_{a_l} \neq 0)$, then $V(X_j^i) = V(\neg x_l) = 0$. Thus, we obtain $V \not\models C_i$, which is a contradiction with the fact that V is a satisfying variable assignment. Thus, $T \not\models \psi$.

We now prove the converse. Assume there is a tree T with $T \models \varphi$ and $T \not\models \psi$. The former implies that there exists a child of the root of T, v such that $T, v \models b$ and the attributes $a_1, \ldots, a_n$ are defined at v. Moreover since $T \not\models \psi$, for every $i, 1 \leq i \leq n$ it holds that $T, v \not\models b \wedge B_1^i \wedge B_2^i \wedge B_3^i$. We define the variable assignment $V : \{x_1, \ldots, x_n\} \to \{0, 1\}$ as follows: $V(x_l) := 0$ iff $att(v, a_l) = 0$ and $V(x_l) := 1$ iff $att(v, a_l) \neq 0$. We claim that $V \models Q$. Assume the opposite, i.e., there exists a clause C_j which is mapped to 0 under V. By definition of V, it follows that $T, v \models B_1^j \wedge B_2^j \wedge B_3^j$ and therefore, $T, v \models b \wedge B_1^j \wedge B_2^j \wedge B_3^j$, which is a contradiction. $\square$

Required attributes

In Section 3.3.2 we dealt with the case when attributes are optional. We now consider the case when some attributes are required. We say that an attribute $a \in A$ is required in a tree T with domain N if the function $att : N \times \{a\} \to D$ is total. We show that when at least one attribute is required, containment of tree patterns with equality and inequality comparisons rises to PSPACE.

Theorem 3.3.3. *The containment problem for* $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$ *interpreted over trees with at least one required attribute is* PSPACE*-complete.*

Proof. We show the upper bound for $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$ expanded with the other equality operators (i.e., $<, >, \leq$ and $\geq$). For that, we reduce the containment problem in this fragment to containment for unions of $\mathsf{XP}^{\{[\,],*,//,\neg\}}$ (tree pattern formulas with unrestricted

^{2}The purpose of the inequalities $@_{a_i} \neq 2$ is to guarantee that the attribute a_i is defined in the b-node of a model of φ. We could express the same with the comparison $@_{a_i} \leq 1$ or $@_{a_i} \geq 0$.

label negation) similar to Lemma 3.3.3. The additional axiom in Ax (Figure 3.2) is $\langle\downarrow^*\rangle(\neg p_{@_a=c} \wedge \neg p_{@_a \neq c})$ for every required $a \in A$, where c is a constant (note that this axiom contains *unsafe* negation). This axiom enforces that the attribute a is defined everywhere in the tree. In (Facchini et al., 2015) and Chapter 5 we show that containment for unions of $\mathsf{XP}^{\{[\,],*,//,\neg\}}$ is solvable in PSPACE.

For proving the lower bound we encode the corridor tiling problem, which is known to be hard for PSPACE (Chlebus, 1986). Our lower bound proof uses the construction from the PSPACE-hardness proof for the containment problem in tree patterns with disjunction over a finite alphabet in (Neven and Schwentick, 2006).

The corridor tiling problem is formalized as follows. Let $\mathsf{Til} = (D, H, V, \bar{b}, \bar{t}, n)$ be a tiling system, where $D = \{d_1, \ldots, d_m\}$ is a finite set of tiles, $H, V \subseteq D^2$ are horizontal and vertical constraints, n is a natural number in unary notation, $\bar{b}$ and $\bar{t}$ are tuples over D of length n. Given such a tiling system, the goal is to construct a tiling of the corridor of width n using the tiles from D so that the constraints H and V are satisfied. Moreover, the bottom and the top row must be tiled by $\bar{b}$ and $\bar{t}$ respectively.

Let $a \in A$ be a required attribute. Now we construct two $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$ expressions φ and ψ such that $\varphi \not\subseteq \psi$ over trees with a required attribute a iff there exists a tiling for Til. To this purpose, we use a string representation of a tiling. Each row of the considered tiling is represented by the tiles it consists of. If the tiling of a corridor of width n has k rows, it is represented by its rows separated by the special symbol $\sharp$. Thus, a tiling is a word of the form $u_1\sharp u_2\sharp\cdots\sharp u_k\$$, where each u_i is the word of length n corresponding to the i-th row in the tiling, and \$ denotes the end of tiling. Note $u_1 = \bar{b}$ and $u_k = \bar{t}$.

For the sake of readability, for expression r, we use the abbreviation r^i to denote the path formula $?r; \downarrow; ?r; \ldots; \downarrow; ?r$ with i occurrences of r.

We then define the formulas over attributes $\{a\}$ and attribute domain containing $D \cup \{\sharp\}$.

Define φ' as

$$\langle ?(@_a = b_1); \downarrow; ?(@_a = b_2); \ldots; \\ \downarrow; ?(@_a = b_n); \downarrow; ?(@_a = \sharp);\ \downarrow^+\ ; ?(@_a = t_1), \downarrow; \ldots \downarrow; ?(@_a = t_n); \downarrow\rangle\$.$$

Intuitively, this expression enforces a tiling to start with a path starting with $\bar{b}$ and finishing with $\bar{t}$. Now the formula ψ' defines all incorrect tilings and additional constraints. It is the disjunction of the following $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$ formulas.

(1) Incorrect length of a row.

(1a) $\bigvee_{i=0}^{n-1} \langle\downarrow^+\ ; ?(@_a = \sharp); \downarrow; \top^i;\ \downarrow\rangle(@_a = \sharp)$, a row is too short,

(1b) $\langle\downarrow^+\ ; (@_a \neq \sharp)^{n+1}\rangle\top$, a row is too long.

(2) $\langle\downarrow^+;\ ?(@_a \neq d_1 \wedge \ldots \wedge @_a \neq d_m \wedge @_a \neq \sharp);\ \downarrow^+\rangle\$$, neither the delimiter or a tile on a position,

(3) Horizontal or vertical constraints are violated.

(3a) $\bigvee_{(d_1,d_2)\notin H} \langle\downarrow^+;\ ?(@_a = d_1);\ \downarrow;\ ?(@_a = d_2)\rangle\top$, a horizontal constraint is violated,

(3b) $\bigvee_{(d_1,d_2)\notin V} \langle\downarrow^+;\ ?(@_a = d_1); \downarrow;\ \top^n;\ \downarrow;\ ?(@_a = d_2)\rangle\top$, a vertical constraint is violated.

We show that there exists a tree with a required attribute a such that $T \models \varphi'$ and $T \not\models \psi'$ iff there exists a tiling for Til.

$(\Leftarrow)$. Assume that there exists a tiling of the corridor. Let s be the string representation of it. Then, $s = u_1 \sharp u_2 \sharp \dots \sharp u_k \$$, where $|u_i| = n$, $u_i \in D^n$, $u_1 = \bar{b}$ and $u_k = \bar{t}$. Moreover, on the one hand if $x \cdot y$, is an infix of some u_i, then $(x, y) \in H$, and on the other hand for every infix $x \cdot u' \cdot y$ of length $n + 1$ of $u_i \sharp \cdot u_{i+1}$, it holds that $(x, y) \in V$. Let T_s be the corresponding tree, i.e., a single path of $|s|$ nodes $\{v_1, \dots, v_{|s|}\}$ where the label of each node $v_i, i < |s|$ is z, the label of $v_{|s|}$ is $\$$ and attribute function is defined according to s, i.e., $att(v_i, a) = s_i$. Clearly, T_s is a model of φ' and not of ψ'.

$(\Rightarrow)$. Let T be a tree such that $T, r \models \varphi'$, $T, r \not\models \psi'$ and $att(n, a)$ is defined for every $n \in Nodes(T)$. Since $T, r \models \varphi'$, there must exist a path $r = v_1, \dots, v_m$ in T such that $att(v_i, a) = b_i, 1 \leq i \leq n$ and $att(v_{m-n+i}, a) = t_j, 1 \leq j \leq n$. Moreover, either $\sharp$ or a symbol from D is in the attribute of every node $v_i, 1 \leq i < m$, according to (2).

We define a tiling function $g : \{0, \dots, n-1\} \times \mathbb{N} \to D$ assigning a tile to every position in the corridor as follows: $g(i, j) = att(v_{(n+1) \times j + i + 1}, a), 1 \leq i \leq n$, where l is the labeling function of T. Indeed, this function is well defined, as (1) ensures the correct counting.

By formulas (3a) and (3b) the tiling defined by g satisfies the horizontal and vertical constraints.

We then apply Proposition 3.3.3 to remove the outermost disjunction in ψ' to obtain the equivalent containment problem $\varphi \subseteq \psi$ in $\mathsf{XP}^{\{[\,],*,//\}}_{=,\neq}$. □

Theorem 3.3.3 provides a lower bound for the containment problem for $\mathsf{PosXPath}^{@,\neg^s}$ and $\mathsf{UCQ}^{@,\neg^s}$ over trees with required attributes. Only for tree patterns we know that the problem is PSPACE-complete. Using the same reduction as in the proof of the upper bound in Theorem 3.3.3, and the results on containment for XPath (Marx, 2005) and XPath with path intersection (ten Cate and Lutz, 2009), we obtain EXPTIME and 2EXPTIME upper bounds for containment for $\mathsf{PosXPath}^{@,\neg^s}$ and $\mathsf{UCQ}^{@,\neg^s}$ over trees with required attributes, respectively.

However, if we restrict attributes to be required at nodes labeled with a certain symbol, then the containment is still in coNP and Π_2^P. Let $p \in \Sigma$ be a label and $a \in A$ an attribute name. We say that a is *required at label element* p if $att(n, a)$ is defined whenever $p \in \rho(n)$ for every tree T and node $n \in Nodes(T)$.

Proposition 3.3.5. *The containment problem for* $\mathsf{PosXPath}^{@,\neg^s}$ *and* $\mathsf{UCQ}^{@,\neg^s}$ *with required attributes at certain labeled nodes is in* coNP *and* Π_2^P *respectively.*

Proof. As before, we can prove a variant of Lemma 3.3.3. Let c be a constant name. Whenever attribute a is required at nodes labelled by p we add the axiom $\langle\downarrow^*\rangle(p \wedge \neg p_{@_a = c} \wedge \neg p_{@_a \neq c})$ to the set Ax. Note that the negation is safe. This axiom is obviously sound, and it enforces that whenever p holds, at least one $p_{@_a \mathsf{op} c}$ label holds as well. This ensures that in the construction of the tree with attribues a is defined at each p node. □

3.3.5 Tractable cases

In this section we consider fragments of $\mathsf{PosXPath}^{@}$ where the containment problem remains in PTIME. It is known that containment in $\mathsf{XP}^{\{[\,],//\}}$ and $\mathsf{XP}^{\{[\,],*\}}$ is decidable in PTIME (Amer-Yahia et al., 2002; Miklau and Suciu, 2004).

Proposition 3.3.6. *Let* XP^{X} *be any fragment whose containment problem over multiple-labeled trees is in* PTIME. *Then the containment problem in* $\mathsf{XP}^{@,X}_{=}$ *over multi-labeled trees with attributes is also in* PTIME.

Proof. Let φ and ψ be formulas in $\mathsf{XP}^{@,X}_{=}$.

Our algorithm first checks (in PTIME) if φ is consistent, i.e., if it contains both $@_a = c$ and $@_a = d$ in the label of a node in φ for some $a \in A, c, d \in D$. If φ is inconsistent, we output $\varphi \subseteq \psi$. Otherwise, we proceed as in the proof of Lemma 3.3.3 by reduction to a containment of attribute-free formulas using the translation $\widetilde{(\cdot)}$ and the formula (*Label*) only. □

3.4 Conclusion

We have considered the containment problem for positive XPath and conjunctive queries over trees expanded with attribute value comparisons. We have shown that in general attribute value comparisons do not increase the complexity of containment. The main idea behind the upper bound was to extend the small counterexample technique to positive XPath and conjunctive queries expanded with a restricted form of negation. Then by axiomatizing the needed constraints in the corresponding expanded fragment, we could abstract away the attribute value comparisons.

The complexity, however, does increase from PTIME to CONP for the fragment $\mathsf{XP}^{\{//,[\,]\}}$ of XPath that uses child, descendant and filter expressions when we add equality and inequality comparisons. Another parameter that affects the complexity is optionality of attributes. If we restrict our trees to have at least one required attribute in every node, then the complexity rises to PSPACE. If, however, attributes are required at elements with specific labels only, the complexity of containment remains the same: CONP for positive XPath and Π^P_2 for conjunctive queries.

We end by listing some open problems. Proposition 3.3.6 shows that adding equality comparison only does not affect the PTIME complexity of containment for fragments $\mathsf{XP}^{\{//,[\,]\}}$ and $\mathsf{XP}^{\{[\,],*\}}$. We do not know what happens when only inequality comparison is added.

For conjunctive queries over trees, it is known that the fragments $\mathsf{CQ}(Child)$ and $\mathsf{CQ}(NextSibling)$ have PTIME containment. It is open whether the complexity increases if we add attribute value comparisons.

In this chapter we have introduced safe label negation and showed that if we add this construct to CQ or $\mathsf{PosXPath}$, complexity of containment remains the same. In Chapter 5 we show that when label negation is unrestricted, containment becomes PSPACE-complete already for tree patterns. In Chapter 4 we show that containment for child-only tree patterns with label negation is solvable in PTIME.

4 Containment for Acyclic CQs with Atomic Negation and Arithmetic Comparisons

In the previous chapter (Chapter 3) we have considered containment for conjunctive queries that are expanded with (safe) negation and attribute value comparisons and that are interpreted over trees. In this chapter we consider conjunctive queries (CQ) expanded with atomic negation or arithmetic comparisons and that are interpreted over relational instances. It is known that the containment problem for this language is Π_2^P-complete (Ullman, 2000; Wei and Lausen, 2003). The aim of this chapter is to find restrictions on CQ that allow for tractable containment. In particular, we consider acyclic conjunctive queries and we seek an answer for **RQ 2**. We show that even with the most restrictive form of acyclicity (Berge-acyclicity), containment is coNP-hard. But for a particular fragment of Berge-acyclic CQ with atomic negation or arithmetic comparisons—child-only tree patterns—containment is solvable in PTime.

4.1 Introduction

We revisit the containment problem for conjunctive queries, one of the classic fundamental problems in database theory. As explained in Chpater 2, conjunctive queries (CQ) correspond to select-from-where SQL queries, a class of most frequent queries used in practice. The containment problem is to decide, given two conjunctive queries Q_1 and Q_2, whether the answers of Q_1 are contained in the answers of Q_2 over every database. A well-known result of Chandra and Merlin (1977a) is NP-completeness of the containment problem for CQ. Because of the topic's relevance to practice, there have been a number of papers dedicated to finding syntactic restrictions of conjunctive queries allowing polynomial-time algorithms for containment. *Acyclic* conjunctive queries have been studied as one of the restrictions (Gottlob et al., 2001; Yannakakis, 1981).

Conjunctive queries expanded with atomic negation or arithmetic comparisons are used in practice as well. The containment problem is harder for these classes than for CQ – Π_2^P-complete (Klug, 1988; Ullman, 2000; van der Meyden, 1997). There has been little work on finding fragments of CQ with negation or comparisons that have tractable query containment. Even the restriction of acyclicity for CQ has not been considered in presence of atomic negation or arithmetic comparisons. Indeed, it is a restriction of CQ allowing polynomial-time containment and, furthermore, the known Π_2^P-lower bounds (both in presence of atomic negation and comparisons) involve cyclic queries.

We show that in some cases the acyclicity condition does reduce the complexity of containment. We show a coNP upper bound for acyclic conjunctive queries with negated atoms of bounded arity, and where, furthermore, the negation is guarded. We call this fragment ACQ($\neg^{\mathrm{g}}$). Atomic negation in a conjunctive query $\mathcal{Q}$ is *guarded* if the variables of each negated atom occur in a positive atom of $\mathcal{Q}$ (Bárány et al., 2011). Moreover, we show that containment for acyclic conjunctive queries with arithmetic comparisons of the form $x\mathtt{op}\ c$, where x is a variable, c a constant and op a comparison operator from $\{=, \neq, <, >, \leq, \geq\}$, is also solvable in coNP. We obtain several coNP-hardness results for containment of acyclic queries with atomic negation or comparisons. These lower bounds indicate that the usual notions of acyclicity are not enough to obtain tractability, even with guarded negation and the most restrictive form of acyclicity – Berge acyclicity (Fagin, 1983). On the positive side we show that containment for a particular fragment of Berge-acyclic conjunctive queries with guarded negation, namely child-only tree patterns, is decidable in PTime. We extend this PTime result to the case with arithmetic comparisons. The PTime results are based on a characterization of containment in terms of the existence of a homomorphism. The latter can be checked by reducing to known efficient algorithms for positive acyclic queries (Gottlob et al., 2001).

The contributions of this chapter are summarized in Table 4.1. In particular,

- We identify a fragment of CQ with atomic negation for which containment is coNP-complete: acyclic conjunctive queries with guarded negation with bounded arity of negated atoms. We derive the same bound for acyclic CQ with arithmetic comparisons.

- Consider the following three conditions on a conjunctive query $\mathcal{Q}$ with guarded atomic negation (resp. with arithmetic comparisons). Let `root` be a fixed constant.

  (i) $\mathcal{Q}$ contains an atom with `root` as an argument,

  (ii) $\mathcal{Q}$ is connected,

  (iii) $\mathcal{Q}$ is Berge-acyclic.

  We call queries satisfying conditions (i)–(iii) *pointed Berge-acyclic with guarded atomic negation* (with arithmetic comparisons). We show that for a class of conjunctive queries with guarded atomic negation or comparisons (containing a binary relational name) satisfying at most two of the conditions (i)–(iii), containment is coNP-hard.

- We show that containment for a particular fragment of pointed Berge acyclic conjunctive queries with guarded atomic negation is in PTime using the homomorphism characterisation of containment. It is the class of conjunctive queries with guarded atomic negation that express rooted child-only tree patterns with label negation, or more precisely, queries Q that are built over unary and binary relational names, contain a constant `root` such that it can only appear in the first

Class	Complexity
CQ with atomic $\neg$	Π_2^P-c (Ullman, 2000), (Wei and Lausen, 2003)
CQ with comparisons	Π_2^P-c (Klug, 1988), (van der Meyden, 1997)
ACQ($\neg^{\mathrm{g}}$) , ACQ with comparisons	coNP-c (Thm. 4.3.2, Thm. 4.3.3)
pointed Berge-ACQs with $\neg^g$ or comparisons	in coNP
child-only Tree patterns with $\neg^g$	PTime (Cor. 4.4.1)
desc.-only Tree patterns with $\neg^g$	PTime (Cor. 4.4.2)
child-only Tree patterns with comparisons	PTime (Cor. 4.4.2)
desc.-only Tree patterns with comparisons	PTime (Cor. 4.4.2)

Table 4.1: Known results and the results of this chapter. Here $\neg^g$ denotes guarded atomic negation.

position of an atom in Q, and are Berge-acyclic and satisfiy the following property: if $E_1(x_1, y) \wedge E_2(x_2, y) \in Q$, then $E_1 = E_2$ and $x_1 = x_2$. We use the same technique to show that containment for descendant-only tree patterns, which are defined exactly like child-only tree patterns with the exception that each binary relation E is interpreted as the transitive closure of E in the model, is in PTime as well.

Organization

In Section 4.2 we introduce the needed concepts and notation. In Section 4.3 we show coNP completeness for containment of acyclic conjunctive queries with guarded negated atoms of bounded arity, or comparisons. In Section 4.4 we provide PTime results for tree patterns with label negation or arithmetic comparisons. We end with conclusions, open problems and future work.

4.2 Preliminaries

A relational schema $\mathbf{S}$ is a set of relational names with associated arities. We assume countably infinite disjoint sets of variables and constants $\mathbf{Var}$ and $\mathbf{Const}$. A *term* is an element from $\mathbf{Var} \cup \mathbf{Const}$. We also assume a dense linear order $<$ on $\mathbf{Const}$. For tuples of terms $\bar{x}$ and $\bar{y}$, by $\bar{x} \subseteq \bar{y}$ we denote the fact that every element of $\bar{x}$ is an element of $\bar{y}$. An *instance* I over $\mathbf{S}$ is a set of *facts* of the form $R(a_1, \ldots, a_n)$, where $R \in \mathbf{S}$ is a relational name of arity n and each $a_i \in \mathbf{Const}$. By $dom(I)$ we denote the domain of I, i.e. all the constants appearing in I. A *positive atom* (or just an *atom*) and a *negative atom* are expressions of the form $R(x_1, \ldots, x_n)$ and $\neg R(x_1, \ldots, x_n)$, respectively, where $R \in \mathbf{S}$ is a relational name of arity n and each x_i is a term.

A k-ary *conjunctive query* ($k \geq 0$) over $\mathbf{S}$ is an expression $\mathcal{Q}(\bar{x}) = \exists \bar{y}.\varphi(\bar{x}, \bar{y})$, where $\bar{x}$ is a k-tuple of variables and $\varphi(\bar{x}, \bar{y})$ is a conjunction of positive atoms over $\mathbf{S}$ containing all the terms in $\bar{x}, \bar{y}$. A 0-ary query is called *Boolean*. We say that $\mathcal{Q}(\bar{x}) = \exists \bar{y}.\varphi(\bar{x}, \bar{y})$ is a *conjunctive query with atomic negation* if $\varphi(\bar{x}, \bar{y})$ is a conjunction of positive or negative atoms. We say that an atom $R(\bar{x})$ is *negated* in $\mathcal{Q}$ if $\neg R(\bar{x})$ is a conjunct of $\mathcal{Q}$. Negation is *guarded* in $\mathcal{Q}$ if for every negative atom $\neg R(\bar{x})$ in $\mathcal{Q}$ there is a positive atom $P(\bar{y})$ in $\mathcal{Q}$ such that $\bar{x}' \subseteq \bar{y}$, where $\bar{x}'$ are the variables in $\bar{x}$. In this case, we call $P(\bar{y})$ a *guard* of $\neg R(\bar{x})$ in $\mathcal{Q}$, and $\neg R(\bar{x})$ a *guarded negated atom* in $\mathcal{Q}$. Query $\mathcal{Q}$ is *inconsistent* if an atom and its negation appear in $\mathcal{Q}$ at the same time. Otherwise, it is *consistent*.

We say that $\mathcal{Q}(\bar{x}) = \exists \bar{y} \varphi(\bar{x}, \bar{y})$ is a *conjunctive query with arithmetic comparisons* if $\varphi(\bar{x}, \bar{y})$ is a conjunction of atoms and expressions of the form $x \mathtt{op}\ c$, where $\mathtt{op} \in \{=, \neq, <, >, \leq, \geq\}$, $x \in \mathbf{Var}$ is a variable from $\bar{x}, \bar{y}$, and $c \in \mathbf{Const}$. Note that we do *not* allow comparisons of the form $x \mathtt{op}\ y$, where x and y are variables. We say that $\mathcal{Q}$ is *consistent* if the comparisons of $\mathcal{Q}$ are consistent.

For a positive or a negative atom P and conjunctive query $\mathcal{Q}$, $P \in \mathcal{Q}$ denotes the fact that P is a conjunct of $\mathcal{Q}$. We denote by $Var(\mathcal{Q})$, $Const(\mathcal{Q})$ and $Term(\mathcal{Q})$ the sets of variables, constants and terms occurring in $\mathcal{Q}$. We say that $\mathcal{Q}$ is *connected* if for every pair t and t' of terms in $\mathcal{Q}$, there is a sequence of atoms $P_1, \ldots, P_n$ in $\mathcal{Q}$ such that $t \in Term(P_1)$, $t' \in Term(P_n)$ and $Term(P_i) \cap Term(P_{i+1}) \neq \emptyset$, for every $i, 1 \leq i < n$.

The *answer* of a k-ary conjunctive query with atomic negation $\mathcal{Q}(\bar{x})$ on an instance I is a k-ary relation $Ans(\mathcal{Q}, I) \subseteq \mathbf{Const}^k$ that consists of all tuples $\theta(\bar{x})$ such that $\theta : Var(\mathcal{Q}) \to dom(I)$ is a substitution such that for every positive atom $R(\bar{u}) \in \mathcal{Q}$ it holds that $R(\theta(\bar{u})) \in I$, and for every negative atom $\neg P(\bar{v}) \in \mathcal{Q}$ it holds that $P(\theta(\bar{v})) \notin I$ (here we assume that θ is identity on $\mathbf{Const}$.). A Boolean conjunctive query $\mathcal{Q}$ evaluates to *true* in I (denoted as $I \models \mathcal{Q}$) if the empty tuple is the answer of the query on I, and *false* otherwise. If $I \models \mathcal{Q}$, we refer to θ that witnesses that fact as a *satisfying assignment* for $\mathcal{Q}$ in I. The semantics for conjunctive queries with comparisons is defined similarly. Now instead of preserving negation, a substitution θ must preserve the comparisons.

Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be conjunctive queries of the same arity (with atomic negation or comparisons). We say that $\mathcal{Q}_1$ is *contained* in $\mathcal{Q}_2$, denoted as $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$, if for every instance I it holds $Ans(\mathcal{Q}_1, I) \subseteq Ans(\mathcal{Q}_2, I)$. In case $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are Boolean, $\mathcal{Q}_1$ is *contained* in $\mathcal{Q}_2$ if $I \models \mathcal{Q}_1$ implies $I \models \mathcal{Q}_2$, for every instance I.

Let $\mathcal{C}$ be a class of conjunctive queries. *The containment problem for $\mathcal{C}$* is the following decision problem:

- Given: $\mathcal{Q}_1$ and $\mathcal{Q}_2$ from $\mathcal{C}$,
- Decide: $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$?

We follow Gottlob et al. (2001) in the definition of acyclic conjunctive queries. Acyclicity is defined using the notion of a hypergraph. A *hypergraph* is a pair $H = (V, E)$, where V is a set of vertices and $E \subseteq 2^V$ a set of hyperedges. Given a hypergraph $H = (V, E)$, the *GYO-reduct*, denoted as $GYO(H)$, is the hypergraph obtained from H by repeatedly applying the following rules in exhaustive manner:

- Remove hyperedges that are empty or contained in other hyperedges,
- Remove vertices that appear in at most one hyperedge.

A hypergraph H is (α-)*acyclic* if $GYO(H)$ is the empty hypergraph. The hypergraph $H(\mathcal{Q}) = (V, E)$ of a conjunctive query $\mathcal{Q}$ is defined as follows. The set of vertices

$V = Var(\mathcal{Q})$, and for each atom $R(\bar{x})$ in $\mathcal{Q}$, the set E contains a hyperedge consisting of all variables occurring in $\bar{x}$. If $H(\mathcal{Q})$ is acyclic, then $\mathcal{Q}$ is an *acyclic* conjunctive query. By ACQ($\neg^{g}$) we denote the class of *acyclic conjunctive queries with guarded atomic negation*, that is, the class of conjunctive queries with guarded atomic negation whose positive part (the conjunctive query obtained by removing all negative atoms) is acyclic. *Acyclic conjunctive queries with arithmetic comparisons* is a class of conjunctive queries whose "relational part" (that is, the query obtained by removing the comparisons in the original query) is acyclic. Here we assume that if the comparisons of a query entail that $x = y$, then every occurrence of y in the query is replaced by x.

There is a number of notions of acyclicity for hypergraphs (Fagin, 1983). α-acyclicity is the least restrictive among those. The most restrictive one is Berge-acyclicity (Berge, 1985) defined next.

Definition 4.2.1 (Berge cycle). *Let H be a hypergraph. A* Berge cycle *in H is a sequence $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ such that*

(i) $x_1, \ldots, x_m$ *are distinct vertices in H,*

(ii) $S_1, \ldots, S_m$ *are distinct hyperedges in H, and $S_{m+1} = S_1$,*

(iii) $m \geq 2$*, that is, there are at least 2 hyperedges involved, and*

(iv) x_i *is in S_i and S_{i+1}, $1 \leq i \leq m$.*

We say that H is *Berge-cyclic* if it has a Berge cycle, otherwise it is *Berge-acyclic*. We say that a conjunctive query $\mathcal{Q}$ is Berge-cyclic (Berge-acyclic) if its hypergraph is Berge-cyclic (respectively Berge-acyclic). Similarly to α-acyclicity, we can define Berge-acyclic conjunctive queries with guarded atomic negation or arithmetic comparisons. Note that if a hypergraph contains two hyperedges that share two or more vertices then the hypergraph is Berge-cyclic. By *Berge tree queries* we mean the class of conjunctive queries $\mathcal{Q}$ with *unary* guarded negation (that is, negation occurs only in front of a unary atom) or arithmetic comparisons such that

- $\mathcal{Q}$ uses only unary and binary relational names,
- $\mathcal{Q}$ contains no self-loops, i.e., atoms of the form $E(x, x)$,
- $\mathcal{Q}$ contains a constant `root` which may appear only in the first position of atoms in $\mathcal{Q}$, and $\mathcal{Q}$ contains no other constant,
- $\mathcal{Q}$ is Berge-acyclic,
- $\mathcal{Q}$ is connected,
- If $E_1(x_1, y) \wedge E_2(x_2, y) \in \mathcal{Q}$, then $E_1 = E_2$ and $x_1 = x_2$. In other words, the (hyper)graph of $\mathcal{Q}$ is a tree.

It is easy to see that this class of queries defined over a schema with a single binary relation and (possibly many) unary relations, is precisely the class of child-only *tree patterns* (with negated labels or comparisons) used in an XML context (Miklau and Suciu, 2004).

We give the definition of Tree Patterns that can contain labeled child and descendent edges, and that capture Berge tree queries.

Let Σ and Σ_e be disjoint sets of node and edge labels. A *tree pattern with label negation* is a tree $(N, E, E_{//}, r, l_e, l^+, l^-)$, where N is the set of nodes, $E \cup E_{//} \subseteq N^2$ is the edge relation consisting of disjoint child and descendant relations respectively, $r \in N$ the root node, $l_e : E \cup E_{//} \to \Sigma_e$ the edge labeling function and $l^+, l^- : N \to 2^{\Sigma}$ are positive and negative node labeling functions. For a tree pattern P, by $Nodes(P)$ we denote the set of nodes of P. Let additionally A be a set of attribute names. By Σ_A we denote the set $\{@_a \mathsf{op}\ c \mid a \in A, \mathsf{op} \in \{=, \neq, <, >, \leq, \geq\}, c \in \mathbf{Const}\}$. A *tree pattern with attribute comparisons* is a tree $(N, E, E_{//}, r, l_e, l)$, such that N, $E, E_{//}, r, l_e$ are as above, and $l : N \to 2^{\Sigma \cup \Sigma_A}$ is a node labelling function.

We define what it means that a tree pattern is true in a graph as follows. Let $G = (dom(G), E', \rho_e, \rho_n, r')$ be a graph, where $dom(G)$ is the set of nodes, $E' \subseteq dom(G)^2$ the edge relation, $\rho_e : E' \to 2^{\Sigma_e}$ and $\rho_n : dom(G) \to 2^{\Sigma}$ are the edge and node labelling functions, and $r' \in dom(G)$ is a fixed designated node. We say that a tree pattern with label negation P is *true* in G, or G *satisfies* P, denoted as $G \models P$, if there is a function $e : N \to dom(G)$, called *embedding* of P in G, such that all of the following hold:

(1) If $(x, y) \in E$ and $l_e(\langle x, y\rangle) = \alpha$, then $(e(x), e(y)) \in E'$ and $\alpha \in \rho_e(\langle e(x), e(y)\rangle)$;

(2) If $(x, y) \in E_{//}$ and $l_e(\langle x, y\rangle) = \alpha$, then there exist nodes $n_1, \ldots, n_k$ in $dom(G)$ such that $n_1 = e(x)$, $n_k = e(y)$, $(n_i, n_{i+1}) \in E'$ and $\alpha \in \rho_e(\langle n_i, n_{i+1}\rangle)$, for every $i, 1 \leq i \leq k-1$;

(3) For every $x \in N$, $l^+(x) \subseteq \rho_n(e(x))$;

(4) For every $x \in N$, $l^-(x) \cap \rho_n(e(x)) = \emptyset$.

We say that G *strongly* satisfies P, denoted as $G \models_s P$, if there is an embedding e of P in G such that it additionally satisfies the following condition:

(0) $e(r) = r'$.

The semantics of tree patterns with attribute comparisons is defined over graphs that are additionally equipped with a partial function $att : dom(G) \times A \to \mathbf{Const}$. The definition of $G \models P$ and $G \models_s P$, where P is a tree pattern with attribute comparisons, is defined similarly to the above definition, where (3) and (4) are replaced with the following conditions:

(3') For every $x \in N$, $l(x) \cap \Sigma \subseteq \rho_n(e(x))$;

(4') For every $x \in N$, if $@_a \mathsf{op}\ c \in l(x)$, then $att(e(x), a) \mathsf{op}\ c$.

We say that a tree pattern $P = (N, E, E_{//}, r, l_e, l^+, l^-)$ is *consistent* if for every $x \in N$ it holds $l^+(x) \cap l^-(x) = \emptyset$. Similarly, a tree pattern with attribute comparisons is consistent if the comparisons of every attribute in every node are consistent. Note that a tree pattern P is consistent if and only if there is a structure G such that $G \models P$. Furthermore, consistency of a tree pattern can be checked in PTIME.

For tree patterns with label negation or comparisons P and Q, we say that P is *contained* in Q (resp. *strongly contained*), denoted as $P \subseteq Q$ ($P \subseteq_s Q$), if for every G it holds that $G \models P$ ($G \models_s P$) implies $G \models Q$ ($G \models_s Q$).

We say that a tree pattern is *child-only* if the set $E_{//}$ is empty, and *descendant-only* if E is empty. In these cases we omit the relations $E_{//}$ and E respectively. It is easy to see that every child-only tree pattern with label negation is equivalent (in a strong sense) to a Berge tree query and vice versa. Likewise, every child-only tree pattern with comparisons (with one attribute name) is equivalent to a Berge tree query with comparisons, and vice versa. We omit details of the translations. Thus, child-only tree patterns can be seen as particular fragments of acyclic conjunctive queries. We show in Section 4.4 that containment for this fragment is in PTIME.

4.3 Containment for acyclic conjunctive queries with guarded atomic negation or comparisons

We first state the known result on the containment for CQ with atomic negation or comparisons.

Theorem 4.3.1 (Klug (1988); Nutt (2013); van der Meyden (1997))**.** *The containment problem for conjunctive queries with atomic negation or arithmetic comparisons is* Π_2^P*-complete.*

As noted in the introduction of this chapter, the known proofs for the Π_2^P lower bound involve conjunctive queries that are *cyclic*.

In this section we show that containment for *acyclic* conjunctive queries with guarded negated atoms of bounded arity or comparisons is CONP-complete. We also provide several CONP lower bounds that help to identify the sources of intractability. W.l.o.g. we can consider containment for acyclic *Boolean* conjunctive queries. Indeed, the containment problem for non-Boolean CQ can be reduced in PTIME to containment of Boolean CQ while preserving the acyclicity restriction.

Proposition 4.3.1. *Let $\mathcal{P}$ and $\mathcal{Q}$ be acyclic conjunctive queries with guarded atomic negation (with comparisons). Then there exist* PTIME *computable acyclic Boolean conjunctive queries with guarded atomic negation (with comparisons) $\mathcal{P}'$ and $\mathcal{Q}'$ such that*

$$\mathcal{P} \subseteq \mathcal{Q} \textit{ iff } \mathcal{P}' \subseteq \mathcal{Q}'.$$

This also holds for Berge-acyclic queries.

Proof. Let $\mathcal{P}(\bar{x})$ and $\mathcal{Q}(\bar{y})$ be acyclic conjunctive queries with guarded atomic negation (comparisons). We check if $\mathcal{P}$ and $\mathcal{Q}$ are inconsistent which can be done in PTIME (van der Meyden, 1997; Wei and Lausen, 2003). If $\mathcal{P}$ is inconsistent, let $\mathcal{P}' = \mathcal{Q}' = \exists x.P(x)$. If $\mathcal{Q}$ is inconsistent or the length of $\bar{x}$ and the length $\bar{y}$ are different, let $\mathcal{P}' = \exists x.P_1(x)$ and $\mathcal{Q}' = \exists x.P_2(x)$. In case $\mathcal{P}$ and $\mathcal{Q}$ are consistent, let $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_n)$, and $P_1, \ldots, P_n$ be unary relational names that do not appear in $\mathcal{P}$ or $\mathcal{Q}$. Then we define $\mathcal{P}' = \exists \bar{x}.P_1(x_1) \wedge \ldots \wedge P_n(x_n) \wedge \mathcal{P}(\bar{x})$ and $\mathcal{Q}' = \exists \bar{y}.P_1(y_1) \wedge \ldots \wedge P_n(y_n) \wedge \mathcal{Q}(\bar{y})$. Clearly, $\mathcal{P}'$ and $\mathcal{Q}'$ are PTIME computable. Moreover, if $\mathcal{P}$ and $\mathcal{Q}$ are p-acyclic ($p \in \{\alpha, \text{Berge}\}$), then $\mathcal{P}'$ and $\mathcal{Q}'$ are p-acyclic as well. It is easy to show that $\mathcal{P} \subseteq \mathcal{Q}$ iff $\mathcal{P}' \subseteq \mathcal{Q}'$. $\square$

Thus, in the rest of the chapter we only consider containment for *Boolean* acyclic CQ with guarded atomic negation or comparisons.

Now we show that restricting conjunctive queries with atomic negation to be acyclic and with guarded negated atoms of bounded arity makes the containment problem coNP-complete.

Theorem 4.3.2. *The containment problem for acyclic conjunctive queries with guarded negated atoms of bounded arity (or with arithmetic comparisons), is in* coNP.

Proof. Let $\mathcal{P}$ and $\mathcal{Q}$ be input queries. A coNP algorithm then works as follows. We first guess a potential counterexample I, and, second, check whether $I \models \mathcal{P}$ and $I \not\models \mathcal{Q}$. Lemma 4.3.1 below guarantees that it is enough to guess a counterexample with size polynomial in the sizes of $\mathcal{P}$ and $\mathcal{Q}$. By Lemma 4.3.2 below, the second step can be done in PTime. The guardness condition is not used in the proof of Lemma 4.3.1, but it is crucial in the proof of Lemma 4.3.2. $\square$

Lemma 4.3.1. *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be Boolean* ACQ($\neg^{g}$) *queries such that the arity of negative atoms is bounded by a fixed constant k (resp. with arithmetic comparisons). If $\mathcal{Q}_1 \not\subseteq \mathcal{Q}_2$, then there is an instance I such that $I \models \mathcal{Q}_1$, $I \not\models \mathcal{Q}_2$, and the size of I is polynomial in the sizes of $\mathcal{Q}_1$, $\mathcal{Q}_2$.*

Proof. We first consider the case of ACQ($\neg^{g}$). By assumption, for every negative atom $\neg R$ in $\mathcal{Q}_2$, the arity of R is bounded by a constant k. Let I' be a counterexample for $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$. Since $I' \models \mathcal{Q}_1$, there is a satisfying assignment $\theta : Var(\mathcal{Q}_1) \to dom(I')$. By $\theta(Var(\mathcal{Q}_1))$ we denote the range of θ. Furthermore, by $\theta(\mathcal{Q}_1)$ we denote the image of positive atoms in $\mathcal{Q}_1$ wrt θ, i.e., the set $\{R(\theta(\bar{x})) \mid R(\bar{x}) \in \mathcal{Q}_1\}$. We then define the instance I as the set

$$\begin{aligned}\theta(\mathcal{Q}_1) \cup \{P(\bar{a}) \in I' \mid P \text{ occurs negatively in } \mathcal{Q}_2,\\ \bar{a} \subseteq \theta(Var(\mathcal{Q}_1)) \cup Const(\mathcal{Q}_1) \cup Const(\mathcal{Q}_2)\}.\end{aligned}$$

Note that I is a subinstance of I' and its size is bounded by $|\mathcal{Q}_1| + |\mathcal{Q}_2| \cdot (|\mathcal{Q}_1| + |\mathcal{Q}_2|)^k$. Firstly, θ is a satisfying assignment for $\mathcal{Q}_1$ in I, and thus $I \models \mathcal{Q}_1$. Indeed, the positive atoms are preserved since $\theta(\mathcal{Q}_1) \subseteq I$. Furthermore, no negative atom in $\mathcal{Q}_1$ becomes true under θ since I is a subinstance of I' and θ is a satisfying assignment for $\mathcal{Q}_1$ in I'. Secondly, we show $I \not\models \mathcal{Q}_2$. Suppose the opposite. This means there is a satysfying assignment $h : Var(\mathcal{Q}_2) \to dom(I)$. We show that h is also a satisfying assignment for $\mathcal{Q}_2$ in I', which contradicts the assumption.

- Let $R(\bar{x}) \in \mathcal{Q}_2$. Then $R(h(\bar{x})) \in I \subseteq I'$.
- Let $\neg R(\bar{x}) \in \mathcal{Q}_2$. Then $R(h(\bar{x})) \notin I$. Note that $h(\bar{x}) \subseteq \theta(Var(\mathcal{Q}_1)) \cup Const(\mathcal{Q}_1) \cup Const(\mathcal{Q}_2)$. Thus, because of that and the fact that R occurs negatively in $\mathcal{Q}_2$, it follows that $R(\bar{a}) \notin I'$ by the definition of I.

We now prove the lemma for the case of arithmetic comparisons. Let θ be a satisfying assignment for $\mathcal{Q}_1$ in I'. We take I as $\theta(\mathcal{Q}_1) = \{R(\theta(\bar{x})) \mid R(\bar{x}) \in \mathcal{Q}_1\}$. The size of I is obviously polynomial. $I \models \mathcal{Q}_1$ holds because θ is a satisfying assignment for $\mathcal{Q}_1$ in I. Furthermore, $I \not\models \mathcal{Q}_2$ holds since any satisfying assignment for $\mathcal{Q}_2$ in I is a satisfying assignment for $\mathcal{Q}_2$ in I'. $\square$

The *evaluation problem* for a class of Boolean queries $\mathcal{C}$ is the following decision problem. Given an instance I, a Boolean query $\mathcal{Q} \in \mathcal{C}$, decide whether $\mathcal{Q}$ evaluates to true in I, i.e., $I \models \mathcal{Q}$.

Lemma 4.3.2. *The evaluation problem is in* PTIME *for each of the following classes of Boolean queries:*

(i) Boolean acyclic conjunctive queries with guarded negated atoms of bounded arity, and

(ii) Boolean acyclic conjunctive queries with arithmetic comparisons.

Proof. We prove item (i). Let I be an instance and $\mathcal{Q}$ a Boolean acyclic query with guarded atomic negation such that each negated atom is bounded by a constant k. We make a polynomial reduction to the evaluation problem for (positive) acyclic Boolean conjunctive queries which is known to be in PTIME (Gottlob et al., 2001; Yannakakis, 1981).

For every relational name R that occurs negatively in $\mathcal{Q}$, we introduce a new relational name $\widetilde{R}$ of the same arity as R. By $\widetilde{\mathcal{Q}}$ we denote the result of replacement of each negated atom $\neg R(\bar{x})$ in $\mathcal{Q}$ by $\widetilde{R}(\bar{x})$. Note that $\widetilde{\mathcal{Q}}$ now is an ordinary CQ. Moreover, $\widetilde{\mathcal{Q}}$ is acyclic because (1) $\mathcal{Q}$ is acyclic and (2) every hyperedge in $H(\widetilde{\mathcal{Q}})$ corresponding to an atom $\widetilde{R}(\bar{x})$ is contained in a hyperedge corresponding to an (positive) atom $P(\bar{y})$ that appears in $\mathcal{Q}$ and $\widetilde{\mathcal{Q}}$. We then define the instance

$$\widetilde{I} = I \cup \{\widetilde{R}(\bar{a}) \mid \bar{a} \subseteq dom(I), \neg R(\bar{x}) \in \mathcal{Q}, \text{ and } R(\bar{a}) \notin I\}.$$

Note that the size of $\widetilde{I}$ is bounded by $|I| + |\mathcal{Q}| \cdot |dom(I)|^k$ which is polynomial in the sizes of I and $\mathcal{Q}$. We claim that $I \models \mathcal{Q}$ if and only if $\widetilde{I} \models \widetilde{\mathcal{Q}}$.

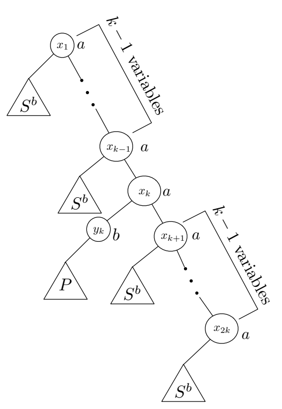
($\Rightarrow$). Suppose $I \models \mathcal{Q}$, i.e., there is a satisfying variable assignment $\theta : Var(\mathcal{Q}) \to dom(I)$. Note that $Var(\mathcal{Q}) = Var(\widetilde{\mathcal{Q}})$ and $dom(I) = dom(\widetilde{I})$. We show that θ is a satisfying assignment for $\widetilde{\mathcal{Q}}$ in $\widetilde{I}$. The positive atoms from $\mathcal{Q}$ are still preserved since we did not remove any facts from I. Let $\widetilde{R}(\bar{x}) \in \widetilde{\mathcal{Q}}$. This means that $\neg R(\bar{x}) \in \mathcal{Q}$. Hence, $R(\theta(\bar{x})) \notin I$. Since also $\theta(\bar{x}) \subseteq dom(I)$, we have that $R(\theta(\bar{x})) \in \widetilde{I}$, as needed.

($\Leftarrow$). Suppose $\widetilde{I} \models \widetilde{\mathcal{Q}}$, i.e., there is a satisfying assignment $\theta : Var(\widetilde{\mathcal{Q}}) \to dom(\widetilde{I})$. We show that θ is a satisfying assignment for $\mathcal{Q}$ in I. Positive atoms in $\mathcal{Q}$ are preserved since they are positive atoms in $\widetilde{\mathcal{Q}}$ as well and θ preserves them in $\widetilde{I}$ and thus in I. Let $\neg R(\bar{x}) \in \mathcal{Q}$. This means that $\widetilde{R}(\bar{x}) \in \widetilde{\mathcal{Q}}$. Hence $\widetilde{R}(\theta(\bar{x})) \in \widetilde{I}$ and, in particular, $\widetilde{R}(\theta(\bar{x})) \in \{\widetilde{R}(\bar{a}) \mid \bar{a} \subseteq dom(I), \neg R(\bar{x}) \in \mathcal{Q}, \text{ and } R(\bar{a}) \notin I\}$. It follows that $R(\theta(\bar{x})) \notin I$.

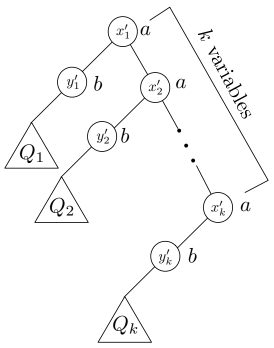
Item (ii) is shown similarly. Now each arithmetic comparison $x\mathsf{op}\ c$ that occurs in $\mathcal{Q}$ is replaced with a new unary atom $P_{\mathsf{op}\ c}(x)$. Let $\widetilde{\mathcal{Q}}$ be the result of this replacement. Let Σ_c be the constants occurring in the comparisons of $\mathcal{Q}$. Note that $|\Sigma_c| \leq |\mathcal{Q}|$. We define the instance

$$\widetilde{I} = I \cup \{P_{\mathsf{op}\ c}(a) \mid a \in dom(I), \mathsf{op} \in \{=, \neq, <, >, \leq, \geq\} \text{ and } a\mathsf{op}\ c\}.$$

Note that the size of $\widetilde{I}$ is bounded by $|I| + 6 \cdot |\mathcal{Q}| \cdot |dom(I)|$, which is polynomial in the sizes of I and $\mathcal{Q}$. It is straightforward to show that $I \models \mathcal{Q}$ if and only if $\widetilde{I} \models \widetilde{\mathcal{Q}}$. □



4.1(a) Query P'.



4.1(b) Query Q'.

Figure 4.1: Queries P' and Q' from Lemma 5.4.2.

We show that the corresponding coNP lower bound for containment already holds for child-only tree patterns with negated labels (or comparisons). For this, we first show that we can allow disjunction on the right hand side query of the containment problem.

We extend the definition of containment for *unions* of tree patterns with label negation, i.e., expressions of the form $\bigvee_{i=1}^{k} Q_i$ where each Q_i is a tree pattern with label negation. Let P be a tree pattern and $Q = \bigvee_{i=1}^{k} Q_i$ a union of tree patterns with label negation. We say that P is *contained* (resp. *strongly contained*) in Q if for every G it holds that $G \models P$ ($G \models_s P$) implies that there is a $j \in \{1, \dots, k\}$ such that $G \models Q_j$ ($G \models_s Q_j$).

Lemma 4.3.3. *Let P be a child-only tree pattern with label negation (resp. attribute comparisons) and $Q = \bigvee_{i=1}^{k} Q_i$ a union of child-only tree patterns with label negation (attribute comparisons). There exist* PTIME *computable child-only tree patterns with label negation (attribute comparisons) P' and Q' such that*

$$P \subseteq_s Q \textit{ if and only if } P' \subseteq Q'.$$

Proof. The proof is similar to the one of Lemma 3 in (Miklau and Suciu, 2004). Let a, b be node labels not occurring in P or Q. Let S^b be the child-only tree patterns corresponding to (written as conjunctive queries) $b(x) \wedge \bigwedge_{i=1}^{k}(e(x, y_i) \wedge Q_i)$, where y_i is the root of Q_i, x is not among the variables of every Q_i and e is the child relation. We define P' and Q' as in Figure 4.1.

Clearly P' and Q' are child-only tree patterns with label negation and PTIME computable. We verify that $P \subseteq_s Q$ if and only if $P' \subseteq Q'$.

($\Rightarrow$). Assume $P \subseteq_s Q$. Let I be an instance (graph) such that $I \models P'$. That means there is an embedding $\theta : Nodes(P') \to dom(I)$. Note that θ is also an embedding for P in I as well, since P is a subquery of P'. Thus, $I \models Q$, i.e., there exists an index

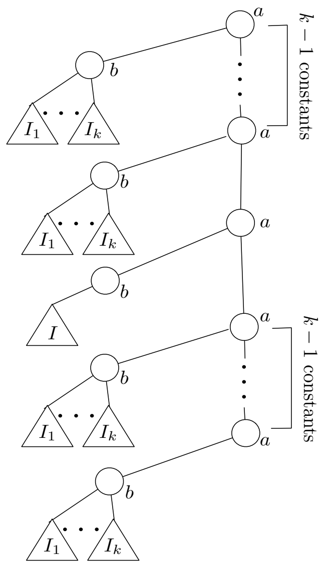


Figure 4.2: The instance I' from Lemma 5.4.2.

$j, 1 \leq j \leq k$, such that $I \models Q_j$. The latter implies there is an embedding h' of Q_j in I. Moreover, since the containment of P in Q is strong, we must have that h' maps the root of Q_j to the θ-image of the root of P. Then we define a mapping $\theta' : Nodes(Q') \to I$ as the composition of h with θ, where h is a mapping from $Nodes(Q')$ to $Nodes(P')$ that extends h' and is defined for the other nodes as follows. For every $i, 1 \leq i \leq k$, we define $h(x'_i) := x_{k-j+i}$, $h(y'_i) = y_{k-j+i}$ and the nodes of Q_i ($i \neq j$) in Q' are mapped "canonically" to the corresponding nodes in S^b. It is easy to see that θ' is indeed an embedding of Q' in I.

($\Leftarrow$). Assume $P' \subseteq Q'$. Let I be an instance such that $I \models P$ and h an embedding of P in I. Let also I_i be the canonical tree (instance) of Q_i (i.e., the instance containing only the positive atoms of Q_i and replacing each variable by a fresh constant). Then we construct the instance I' depicted in Figure 4.2. Note that I is connected to the b-node via the h-image of the root of P. By the assumption, it holds that $I' \models Q'$, i.e., there is an embedding $\theta : Nodes(Q') \to dom(I')$. In particular, since a only appears in the vertical span of $2k-1$ nodes in Figure 4.2, the span of k a-nodes of Q' can only be mapped on that vertical span in I'. Because of this and the fact that a b-node in Q' must be mapped to a b-node in I', there must exist an index j for which θ is an embedding of Q_j in I. Moreover, this embedding maps the root of Q_j to the h-image of the root of P. Thus, $I \models_s Q_j$ and, therefore, $I \models_s Q$.

The case of tree patterns with comparisons is proved similarly. $\square$

Lemma 4.3.4. *The containment problem $\varphi \subseteq_s \bigvee_{i=1}^{m} \psi_i$ is* coNP*-hard for each of the following cases:*

(i) φ, ψ_i are propositional conjunctive formulas with guarded atomic negation,

(ii) φ, ψ_i are child-only tree patterns with attribute comparisons.

Proof. (i). We reduce 3SAT to the complement of the containment problem for the stated fragment. Let φ' be a conjunction of clauses $C_i = (b_1^i \vee b_2^i \vee b_3^i), 1 \leq i \leq k$, over the variables $\{x_1, \ldots, x_n\}$, where b_j^i are literals, i.e., variables or their negations. For a literal l, by $\dot{\neg}\, l$ we denote $\neg p$ if $l = p$ and p if $l = \neg p$.

For every clause C_i we introduce a propositional variable c_i. Then we define propositional conjunctive formulas with atomic negation φ and ψ_i over the propositional variables $c_i, 1 \leq i \leq k$, and $x_j, 1 \leq j \leq n$, as follows. We define $\varphi = c_1 \wedge \ldots \wedge c_k$ and $\psi_i = c_i \wedge \dot{\neg} b_1^i \wedge \dot{\neg} b_2^i \wedge \dot{\neg} b_3^i$.

We claim that φ' is satisfiable iff $\varphi \not\subseteq_s \bigvee_{i=1}^k \psi_i$. This is easy to see using de Morgan's laws. In particular, $\varphi \not\subseteq_s \bigvee_{i=1}^k \psi_i$ iff $\varphi \wedge \bigwedge_{i=1}^k \neg\psi_i$ is satisfiable iff $c_1 \wedge \ldots \wedge c_k \wedge \bigwedge_{i=1}^k (\neg c_i \vee b_1^i \vee b_2^i \vee b_3^i)$ is satisfiable iff $\bigwedge_{i=1}^k (b_1^i \vee b_2^i \vee b_3^i)$ is satisfiable, i.e., φ' is satisfiable.

Item (ii) is proved as follows. Let φ' be as above. The tree pattern φ is defined as $(\{r, m_1, \ldots, m_n\}, \{\langle r, m_1\rangle, \ldots, \langle r, m_n\rangle\}, r, l_e, l)$, where for every $i \in \{1, \ldots, n\}$, $l_e(\langle r, m_i\rangle) = \alpha$, for some α, and $l(m_i) = \{p_i, @_a \neq 2\}$. Each ψ_i is defined as $(\{r_i, n_i^1, n_i^2, n_i^3\}, \{\langle r_i, n_i^1\rangle, \langle r_i, n_i^2\rangle, \langle r_i, n_i^3\rangle\}, r_i, l_e^i, l_i)$, where $l_e^i(\langle r_i, n_i^j\rangle) = \alpha$ and $l(n_i^j) = \{p_l, B_j^i\}$, where B_j^i is $(@_a = 0)$ iff $b_j^i = x_l$ and B_j^i is $(@_a \neq 0)$ iff $b_j^i = \neg x_l$ in C_i. It is straightforward to show that φ' is satisfiable iff $\varphi \not\subseteq_s \bigvee_{i=1}^k \psi_i$. □

Theorem 4.3.3. *Containment is* coNP-*hard for each of the following classes of queries:*

a) for connected Berge-acyclic conjunctive queries with guarded unary negated atoms (or with comparisons),

b) for Berge-acyclic conjunctive queries with guarded unary negated atoms (with comparisons) and with the requirement that they contain an atom with `root` *as an argument, for a fixed constant* `root`.

c) and for connected α-acyclic conjunctive queries with guarded unary negation (with comparisons) and with the requirement that they contain an atom with `root` *as an argument, for a fixed constant* `root`.

Proof. Item a) follows from Lemmas 5.4.2 and 4.3.4, since tree patterns with negated labels (resp. with comparisons) are connected Berge-acyclic conjunctive queries with guarded unary negated atoms (with comparisons).

Item b) follows from a). Indeed, let $\mathcal{P} \subseteq \mathcal{Q}$ be the encoding from a). Then for $\mathcal{P}' = R(\texttt{root}) \wedge \mathcal{P}$ and $\mathcal{Q}' = R(\texttt{root}) \wedge \mathcal{Q}$, where `root` is a constant and R a unary relational name, it holds $\mathcal{P} \subseteq \mathcal{Q}$ if and only if $\mathcal{P}' \subseteq \mathcal{Q}'$.

For item c) we use the encoding from a). Let $\mathcal{P} \subseteq \mathcal{Q}$ be the encoding from a). We construct Boolean α-acyclic conjunctive queries $\mathcal{P}'$ and $\mathcal{Q}'$ as follows. Let $\{x_1, \ldots, x_n\}$ be the variables of $\mathcal{P}$, P a fresh $(n+1)$-ary relational name, `root` a constant, G a fresh binary relational name, and r the root (variable) of $\mathcal{Q}$. Then $\mathcal{P}' = P(\texttt{root}, x_1, \ldots, x_n) \wedge \mathcal{P} \wedge \bigwedge_{i=1}^n G(\texttt{root}, x_i)$ and $\mathcal{Q}' = G(\texttt{root}, r) \wedge \mathcal{Q}$. Note that both $\mathcal{P}'$ and $\mathcal{Q}'$ are (α-)acyclic queries with guarded negation. In particular, $\mathcal{P}'$ is acyclic because all its hyperedges are contained in the hyperedge P. We show that $\mathcal{P} \subseteq \mathcal{Q}$ iff $\mathcal{P}' \subseteq \mathcal{Q}'$.

($\Rightarrow$). Let I be an instance such that $I \models \mathcal{P}'$. Thus there exists a satisfying variable assignment $\theta : Var(\mathcal{P}') \to dom(I)$. In particular, θ is satisfying for $\mathcal{P}$ in I. Let $I' = I \upharpoonright \theta(\{x_1, \ldots, x_n\})$. Since $I' \models \mathcal{P}$, it holds that $I' \models \mathcal{Q}$. Let θ' be a satisfying assignment for Q in I'. Then r is mapped to one of $\theta(x_j)$. Since `root` must be mapped to `root` and $G(\texttt{root}, \theta(x_i))$ holds in I for every i, we have that $\theta' \cup \{\texttt{root} \to \texttt{root}\}$ is a satisfying assignment for $\mathcal{Q}'$ in I.

($\Leftarrow$). Let I be an instance such that $I \models \mathcal{P}$. Let θ be a satisfying assignment for $\mathcal{P}$ in I. W.l.o.g. we can assume that $I = I \upharpoonright \theta(Var(\mathcal{P}))$. We then construct the instance I' as

$$\begin{aligned} I' = I \cup \{&P(\texttt{root}, \theta(x_1), \ldots, \theta(x_n)), \\ &G(\texttt{root}, \theta(x_1)), \ldots, G(\texttt{root}, \theta(x_n)). \end{aligned}$$

Clearly, $\theta \cup \{\texttt{root} \to \texttt{root}\}$ is a satisfying assignment for $\mathcal{P}'$ in I'. Then it holds that $I' \models \mathcal{Q}'$. Let θ' be a satisfying assignment for $\mathcal{Q}'$ in I'. In particular, $Var(\mathcal{Q})$ must be mapped to $\theta(\{x_1, \ldots, x_n\})$ since G is fresh and `root` is mapped to `root`. Thus, θ' is a satisfying assignment for $\mathcal{Q}$ in I. Thus $I \models \mathcal{Q}$. $\square$

We say that a conjuntive query $\mathcal{Q}$ with guarded negation (resp. with arithmetic comparisons) is *pointed Berge acyclic with guarded negation* (resp. with arithmetic comparisons) if $\mathcal{Q}$ contains at least one binary atom and

(i) is connected,

(ii) contains an atom having `root` as an argument, where `root` is a fixed constant, and

(iii) is Berge-acyclic.

Theorem 4.3.3 shows that containment for the class of conjunctive queries with guarded atomic negation or arithmetic comparisons (with at least one binary atom) that satisfies at most two of the conditions (i)–(iii) is coNP-hard.

We leave it as an open question whether containment for pointed Berge acyclic queries is in PTime. However, we are able to obtain PTime results for containment of child-only tree patterns with label negation or comparisons, which is a fragment of pointed Berge acyclic queries.

4.4 Polynomial-time algorithms for containment

In this section we consider a fragment of pointed Berge-acyclic conjunctive queries with guarded negation – child-only tree patterns with label negation. The additional requirement for tree patterns is that every query must contain a constant `root`, which always maps to the designated node in a graph, i.e., we consider *strong* satisfaction. We refer to such tree patterns as *rooted* child-only tree patterns. This root requirement, besides Berge-acyclicity and connectedness, turns out to be crucial for PTime containment, since without this requirement containment becomes coNP-hard, by Theorem 4.3.3, (b). Note that containment in this case is the same as strong containment.

We characterize containment via the notion of *homomorphism*. Let $P = (N, E, r, l_e, l^+, l^-)$ and $Q = (N', E', r', l'_e, l'^+, l'^-)$ be child-only tree patterns with label negation. A mapping $h : N' \to N$ is called a *homomorphism* from Q to P, if the following are satisfied:

(i) $h(r') = r$;

(ii) If $(x, y) \in E'$ and $l'_e(\langle x, y\rangle) = \alpha$, then $(h(x), h(y)) \in E$ and $l_e(\langle h(x), h(y)\rangle) = \alpha$;

(iii) $l'^{+}(x) \subseteq l^{+}(h(x))$, for every $x \in N'$;

(iv) $l'^{-}(x) \subseteq l^{-}(h(x))$, for every $x \in N'$.

Note that the above definition without item (iv) coincides with the usual definition of homomorphism for tree patterns without negation (Miklau and Suciu, 2004).

Theorem 4.4.1. *Let P and Q be consistent child-only tree patterns with label negation. Then $P \subseteq Q$ if and only if there exists a homomorphism from Q to P.*

Proof. Let $P = (N, E, r, l_e, l^{+}, l^{-})$ and $Q = (N', E', r', l'_e, l'^{+}, l'^{-})$ be child-only tree patterns with label negation.

($\Leftarrow$). Assume $h : N' \to N$ is a homomorphism. Let $G = (Nodes(G), E'', \rho_e, \rho_n, r'')$ be a graph such that $G \models P$, i.e., there is an embedding $e : N \to Nodes(G)$ with $e(r) = r''$. Then we claim that $e' = e \circ h$ is an embedding of Q in G. We check the conditions (0)–(4) from the definition of embedding (except (2)):

(0) $e'(r') = e \circ h(r') = e(r) = r''$.

(1) Let $(x, x') \in E'$ and $l'_e(\langle x, x'\rangle) = \alpha$. Then $(h(x), h(x')) \in E$ and $l_e(\langle h(x), h(x')\rangle) = \alpha$, which implies $(e(h(x)), e(h(x'))) \in E''$ and $\alpha \in \rho_e(\langle e(h(x)), e(h(x'))\rangle)$.

(3) Let $x \in N'$ and $p \in l'^{+}(x)$. Then $p \in l^{+}(h(x))$, which implies that $p \in \rho_n(e(h(x)))$, as needed.

(4) Let $x \in N'$ and $p \in l'^{-}(x)$. Then $p \in l^{-}(h(x))$, which implies that $p \notin \rho_n(e(h(x)))$, as needed.

($\Rightarrow$). We show the contrapositive. Suppose there is no homomorphism from Q to P. We then construct a counter-example by taking a graph (tree) with the same structure as P. Let $P.y$ be the subtree of P rooted in y and $Q.x$ the subtree of Q rooted in x. By induction on the depth of $P.y$ we show

(IH) If there is no homomorphism from $Q.x$ to $P.y$, then there exists a tree T such that $T \models P.y$ and $T \not\models Q.x$.

Base of induction: $depth(P.y) = 0$. Then there is no homomorphism from $Q.x$ to $P.y$ if either

- there exists a label $p \in \Sigma$ such that either (i) $p \in l'^{+}(x)$ and $p \notin l^{+}(y)$, or (ii) $p \in l'^{-}(x)$ and $p \notin l^{-}(y)$. We define $T = (Nodes(T), E'', \rho_e, \rho_n, y)$, where $Nodes(T) = \{y\}$, $E'' = \emptyset$, and ρ_n is defined according to the above cases: (i) $\rho_n(y) = l^{+}(y)$ or (ii) $\rho_n(y) = l^{+}(y) \cup \{p\}$. By construction, we have $T \models P.y$. It also holds that $T \not\models Q.x$. Indeed, otherwise it would hold that $l'^{+}(x) \subseteq \rho_n(y)$ and $l'^{-}(x) \cap \rho_n(y) = \emptyset$, which contradicts with the definition of ρ_n.

- Or there exists x' such that $(x, x') \in E'$. In this case, we define $T = (Nodes(T), E'', \rho_e, \rho_n, y)$, where $Nodes(T) = \{y\}$, $E'' = \emptyset$, and $\rho_n(y) = l^+(y)$. By construction we have that $T \models P.y$. Moreover, $T \not\models Q.x$ since there is no child of the image of x in T.

Induction step: $depth(P.y) > 0$ and there is no homomorphism from $Q.x$ to $P.y$. It is because either

- there exists a label p such that either (i) $p \in l'^+(x)$ and $p \notin l^+(y)$, or (ii) $p \in l'^-(x)$ and $p \notin l^-(y)$. We define $T = (N_T, E_T, \rho_e, \rho_n, y)$, where $N_T = \{n \in N \mid P \models yE^*n\}$, i.e., the nodes below y in P, $E_T = E|_{N_T^2}$, $\rho_e = l_e$, and the node labeling ρ is defined as follows.
$$\rho(u) = \begin{cases} l^+(u) & \text{if } u = y \text{ and (i) holds,} \\ l^+(u) \cup \{p\} & \text{if } u = y \text{ and (ii) holds,} \\ l^+(u) & \text{if } u \neq y. \end{cases}$$
It is easy to see that $T \models P.y$ and $T \not\models Q.x$.

- Or there exists x' such that $(x, x') \in E'$ and for all y_i with $(y, y_i) \in E$ and $l_e(\langle y, y_i\rangle) = l'_e(\langle x, x'\rangle)$ it holds that there is no homomorphism from $Q.x'$ to $P.y_i$. Since $depth(P.y_i) < depth(P.y)$ for every such y_i, by the induction hypothesis, it holds that there exists a tree $T_i = (N_i, E_i, \rho_e^i, \rho_n^i, r_i)$ such that $T_i \models P.y_i$ and $T_i \not\models Q.x'$. Let $T'_j = (N'_j, E'_j, \rho_e'^j, \rho_n'^j, r'_j)$ be the canonical trees for the subtrees $P.u_j$ such that yEu_j and $l_e(\langle y, u_j\rangle) \neq l'_e(\langle x, x'\rangle)$. We can assume that these trees are pairwise disjoint. We then define the tree $T = (N_T, E_T, \rho_e, \rho_n, y)$ such that $N_T = \{y\} \cup \bigcup_i N_i \cup \bigcup_j N'_j$, $E_T = \bigcup_i (E_i \cup \{\langle y, y_i\rangle\}) \cup \bigcup_j (E'_j \cup \{\langle y, u_j\rangle\})$,
$$\rho_e(\langle u, v\rangle) = \begin{cases} l_e(\langle u, v\rangle) & \text{if } u = y, \\ \rho_e^i(\langle u, v\rangle) & \text{if } T_i \models uE_iv, \\ \rho'^j(\langle u, v\rangle) & \text{if } T'_j \models uE'_jv, \text{ and} \end{cases}$$
$$\rho(u) = \begin{cases} l^+(y) & \text{if } u = y, \\ \rho_i(u) & \text{if } T_i \models y_iE_i^*u, \\ \rho'_j(u) & \text{if } T'_j \models u_jE_j'^*u. \end{cases}$$
We claim that $T \models P.y$ and $T \not\models Q.x$. The former is by definition of T. For the latter, suppose $e : Q.x \to T$ is an embedding. In particular, that means that e is an embedding of $Q.x'$ to one of T_i, where y_i is a child of y. Thus, $T_i \models Q.x'$, which is a contradiction.

Thus, since there is no homomorphism from $Q.r'$ to $P.r$, there exists a tree such that $T \models P$ and $T \not\models Q$. □

Corollary 4.4.1. *Containment for child-only tree patterns with label negation is in* PTIME.

Proof. Let $P \subseteq Q$ be a containment problem, where P and Q are child-only tree patterns with label negation. We first check if P is consistent. If not, we output "yes". If it is consistent, we check if Q is consistent. If not, we output "no". Otherwise, by Theorem 4.4.1

it is enough to check existence of a homomorphism from Q to P. To this purpose, we reduce the problem to checking homomorphism for tree patterns without label negation. The latter can be done e.g., using a bottom-up procedure (Miklau and Suciu, 2004). For each negated label $\neg p$ occurring in P or Q, we introduce a new label $\tilde{p}$. For a tree pattern with label negation Q, by $\tilde{Q}$ we denote the result of replacing each negated label $\neg p$ with the corresponding label $\tilde{p}$. It is easy to verify that there is a homomorphism from Q to P if and only if there is a homomorphism from $\tilde{Q}$ to $\tilde{P}$. $\square$

Interestingly, using a similar homomorphism characterization, we can prove PTIME results for containment of descendant-only tree patterns with label negation and tree patterns with attribute comparisons. For each of the cases we introduce the corresponding notions of homomorphism.

Let $P = (N, E_{//}, r, l_e, l^+, l^-)$ and $Q = (N', E'_{//}, r', l'_e, l'^+, l'^-)$ be descendant-only tree patterns. A mapping $h : N' \to N$ is called a *d-homomorphism* of Q to P if it satisfies the conditions (i), (iii) and (iv) of the definition of a homomorphism above, and, furthermore, the following condition:

(ii') If $(x, y) \in E'_{//}$ and $l_e(\langle x, y\rangle) = \alpha$, then $(h(x), h(y)) \in E^+_{//}$ and every edge on the path from $h(x)$ to $h(y)$ in P is labeled with α.

Let $P = (N, E, r, l_e, l)$ and $Q = (N', E', r', l'_e, l')$ be child-only tree patterns with attribute comparisons. Then a mapping $h : N' \to N$ is called an *a-homomorphism* if it satisfies the conditions (i), (ii) and (iii') (where (iii') is obtained from (iii) by replacing l'^+ and l^+ with l' and l), and, furthermore, the following condition:

(iv') For every $x \in N'$, if $@_a \mathtt{op} c \in l'(x)$ then there must exist $@_a \mathtt{op}' c' \in l(h(x))$ for some $\mathtt{op}'$ and c', and, $C \models @_a \mathtt{op}\ c$, where C is the set of comparisons of a-attribute in $l(h(x))$.

The following theorem is proved similarly to Theorem 4.4.1 and given in Appendix 4.A.

Theorem 4.4.2. *Let P and Q be consistent descendant-only tree patterns with label negation (resp. child-only tree patterns with attribute comparisons). Then $P \subseteq Q$ if and only if there exists a d-homomorphism (a-homomorphism) of Q to P.*

Since the existence of d- and a-homomorphism can be checked in PTIME, we obtain the following.

Corollary 4.4.2. *The containment problem is in* PTIME *for the following classes of queries.*

- *Descendant-only tree patterns with label negation,*
- *Child-only tree patterns with attribute comparisons.*

As pointed before, child-only tree patterns with label negation is a fragment of pointed Berge-acyclic conjunctive queries with guarded negation. The precise complexity for the latter fragment is an open question. In the end of this section, we give an example showing that the homomorphism characterization fails for this fragment.

Example 4.4.1. Let Q_1 be the Boolean query $R(c,x), R(c,y), Q(y), R(z,x), R(z,w)$, $\neg Q(w)$ and Q_2 the Boolean query $R(c,u), Q(u)$, $R(v,u), R(v,t), \neg Q(t)$. Clearly, there is no such homomorphism mapping c to c, however the containment holds. Indeed, let I be an instance such that $I \models Q_1$, i.e., there is a satisfying variable assignment $\theta : Var(Q_1) \to dom(I)$. We then define a variable assignment $\theta' : Var(Q_2) \to I$ as the composition of $h : Var(Q_2) \to Var(Q_2)$ with θ, where h is defined according to the following cases.

- $I \models Q(\theta(x))$. In this case we define $h = \{c \to c, u \to x, v \to z, t \to w\}$.
- $I \not\models Q(\theta(x))$. In this case we define $h = \{c \to c, u \to y, v \to c, t \to x\}$.

It is straightforward to verify that θ is a satisfying assignment, thus $I \models Q_2$.

4.5 Conclusion and future work

In this chapter we considered the containment problem for conjunctive queries expanded with atomic negation and arithmetic comparisons. In particular, we studied the common acyclicity restriction on conjunctive queries and addressed the following research question:

RQ 2 Does acyclicity make the complexity of containment for conjuctive queries expanded with atomic negation or arithmetic comparison tractable? If not, what additional restrictions can be imposed to make it tractable?

Firstly, we have shown that complexity of containment for acyclic conjunctive queries with atomic negation can be lowered from Π_2^P to CONP if negation is guarded and the arity of negated atoms is bounded by a fixed constant. Secondly, we have shown several CONP lower bound proofs that indicate that much stronger restrictions than acyclicity need to be imposed to make containment tractable. As a result, we have defined a new fragment of acyclic conjunctive queries with atomic negation–pointed Berge acyclic conjunctive queries for which the complexity of containment is left open. On the positive side, for one particular restricted fragment, namely child-only tree patterns with label negation, containment is in PTIME. We have also shown that containment for child-only tree patterns with attribute comparisons and descendant-only tree patterns with label negation is in PTIME.

The two main remaining open problems are:

- What is the complexity of containment for pointed Berge acyclic conjunctive queries?
- What is the complexity of containment for acyclic conjunctive queries with guarded negation (with no bound on the arity of the negated atoms)?

In Chapter 5 we show that when both child and descendant are allowed in tree patterns with unrestricted label negation, than containment rises to PSPACE. In Chapter 3 we have seen that safe label negation does not increase the complexity of containment when added to tree languages.

4.A Polynomial time algorithms for containment

4.A.1 Descendant-only tree patterns with label negation

In this section we consider descendant-only tree patterns with label negation.

We prove Theorem 4.4.2 for the case of descendant-only tree patterns.

Proof. (of Theorem 4.4.2) Let $P = (N, E_{//}, r, l_e, l^+, l^-)$ and $Q = (N', E'_{//}, r', l'_e, l'^+, l'^-)$ be descendant-only tree patterns with label negation.

($\Leftarrow$). Assume $h : N' \to N$ is a d-homomorphism.

Let $G = (Nodes(G), E'', \rho_e, \rho_n, r'')$ be a graph such that $G \models P$, i.e. there is an embedding $e : N \to Nodes(G)$. Then we claim that $e' = e \circ h$ is an embedding of Q into G. We check the conditions (0)-(4), except (1):

(0) $e'(r') = e \circ h(r') = e(r) = r''$,

(2) Let $(x, x') \in E'_{//}$ and $l'_e(\langle x, y\rangle) = \alpha$. Then $(h(x), h(x')) \in E^+_{//}$ and every edge on the path from $h(x)$ to $h(x')$ in P is labeled with α, which implies $(e(h(x)), e(h(x'))) \in E''^+$, and every edge on the path is labeled with α,

(3) Let $x \in N'$ and $p \in l'^+(x)$. Then $p \in l^+(h(x))$, which implies that $p \in \rho(e(h(x)))$, as needed,

(4) Let $x \in N'$ and $p \in l'^-(x)$. Then $p \in l^-(h(x))$, which implies that $p \notin \rho(e(h(x)))$, as needed.

($\Rightarrow$). We show the contrapositive. Suppose there is no d-homomorphism from Q to P. We then construct a counter-example graph with the same structure as P. Let $P.y$ be the subtree of P rooted in y and $Q.x$ the subtree of Q rooted in x. By induction on the depth of $P.y$ we show

(IH) If there is no d-homomorphism from $Q.x$ to $P.y$, then there exists a tree T such that $T \models P.y$ and $T \not\models Q.x$.

Base of induction: $depth(P.y) = 0$. This part is practically the same as for child-only tree patterns. There is no d-homomorphism from $Q.x$ to $P.y$ if either

- there exists a label p such that either (i) $p \in l'^+(x)$ and $p \notin l^+(y)$, or (ii) $p \in l'^-(x)$ and $p \notin l^-(y)$. We define $T = (\{y\}, \emptyset, \rho_e, \rho_n, y)$, where the labeling function ρ_n is defined according to the above cases: (i) $\rho_n(y) = l^+(y)$ or (ii) $\rho_n(y) = l^+(y) \cup \{p\}$. By construction, we have $T \models P.y$. It also holds that $T \not\models Q.x$. Indeed, otherwise it would hold that $l'^+(x) \subseteq \rho_n(y)$ and $l'^-(x) \cap \rho_n(y) = \emptyset$, which contradicts with the definition of ρ.

- Or there exists x' such that $(x, x') \in E'_{//}$. In this case, we define $T = (\{y\}, \emptyset, \rho_e, \rho_n, y)$, where $\rho_n(y) = l^+(y)$. By construction we have that $T \models P.y$. Moreover, $T \not\models Q.x$ since there is no descendant of the image of x in T.

Induction step: $depth(P.y) > 0$ and there is no d-homomorphism from $Q.x$ to $P.y$. It is because either

- there exists a label p such that either (i) $p \in l'^{+}(x)$ and $p \notin l^{+}(y)$, or (ii) $p \in l'^{-}(x)$ and $p \notin l^{-}(y)$. We define $T = (N_T, E_T, \rho_e, \rho_n, y)$, where $N_T = \{n \in N \mid P \models yE^{*}n\}$, i.e., the nodes below y in P, $E_T = E_{//}|_{N_T^2}$, and the labeling $\rho_e = l_e$ and ρ_n is defined as follows.
$$\rho_n(u) = \begin{cases} l^{+}(u) & \text{if } u = y \text{ and (i) holds,} \\ l^{+}(u) \cup \{p\} & \text{if } u = y \text{ and (ii) holds,} \\ l^{+}(u) & \text{if } u \neq y. \end{cases}$$
It is easy to see that $T \models P.y$ and $T \not\models Q.x$.

- Or there exists x' such that $(x, x') \in E'_{//}$ and for all y' with $(y, y') \in E^{+}_{//}$ it holds that there is no d-homomorphism from $Q.x'$ to $P.y'$. Let $y_i, i \in \{1, \ldots, k\}$, be the direct successors of y'. Since $depth(P.y_i) < depth(P.y)$, by the induction hypothesis, it holds that for every y_i of y there exists a tree $T_i = (N_i, E_i, \rho_e^i, \rho_n^i, r_i)$ such that $T_i \models P.y_i$ and $T_i \not\models Q.x'$. Moreover, these trees have the shape of $P.y_i$. We then define the tree $T = (N_T, E_T, \rho_e, \rho_n, y)$ with the shape as P, i.e., such that $N_T = \{y\} \cup \bigcup_i N_i$, $E_T = \bigcup_i (E_i \cup \{\langle y, y_i \rangle\})$ and
$$\rho_n(u) = \begin{cases} l^{+}(y) & \text{if } u = y, \\ \rho_i(u) & \text{if } T_i \models y_i E^{*} u. \end{cases}$$
We claim that $T \models P.y$ and $T \not\models Q.x$. The former is by definition of T. For the latter, suppose $e : Q.x \to T$ is an embedding. In particular, that means that e is an embedding of $Q.x'$ to a subtree $T_i.y$, where y_i is a descendant of y. Thus, $T_i \models Q.x'$, which is a contradiction.

Thus, since there is no homomorphism from $Q.r'$ to $P.r$, there exists a tree such that $T \models P$ and $T \not\models Q$. □

Corollary 4.A.1. *Containment for descendant-only tree patterns with label negation is solvable in* PTIME.

Proof. Let $P \subseteq Q$, where P and Q are (descendant-only) tree patterns with negation, be an instance of the containment problem. We first check if the input tree patterns are consistent. Consistency check can be done in PTIME by simple check if no node contains a label p in both positive and negative labeling. If P is inconsistent, we output "yes". Otherwise, if Q is inconsistent we output "no". Otherwise, by Theorem 4.4.2 it is enough to check existence of a d-homomorphism from Q to P. We reduce this problem to checking a homomorphism from $\tilde{Q}$ to $\tilde{P}$, where $\tilde{P}$ and $\tilde{Q}$ are tree patterns without negated labels obtained from P and Q by replacing each $\neg p$ with $\tilde{p}$. From the definition of a homomorphism, it can be seen that existence of a homomorphism for positive descendant tree patterns amounts to existence of a homomorphism of a descendent-only tree pattern into a tree. The latter problem can be solved in PTIME (Götz et al., 2007). □

4.A.2 Tree patterns with attribute value comparisons

In this section we prove the second part of Theorem 4.4.2.

Proof. (of Theorem 4.4.2) For simplicity, we assume all edges have the same label, and thus l_e is a constant function. Let $P = (N, E, r, l)$ and $Q = (N', E', r', l')$ be tree patterns with attribute value comparisons.

($\Leftarrow$). Assume $h : N' \to N$ is an a-homomorphism. Let $G = (Nodes(G), E'', \rho_n, r'', att)$ be a graph such that $G \models P$, i.e., there is an embedding $e : N \to Nodes(G)$. Then we claim that $e' = e \circ h$ is an embedding of Q into G. We check the conditions (1)-(4):

(1) $e'(r') = e \circ h(r') = e(r) = r''$,

(2) Let $(x, x') \in E'$. Then $(h(x), h(x') \in E$, which implies $(e(h(x)), e(h(x'))) \in E''$,

(3) Let $x \in N'$ and $p \in l'(x)$. Then $p \in l(h(x))$, which implies that $p \in \rho_n(e(h(x)))$, as needed,

(4) Let $x \in N'$ and $@_a \mathtt{op} c \in l'(x)$. Then there exists $@_a \mathtt{op}' c' \in l(h(x))$ and, $C \models @_a \mathtt{op}\, c$, where C is the set of constraints of a in $l(h(x))$. Since e is an embedding, it holds that $att(e(h(x)), a)$ is defined and C is satisfied by $att(e(h(x)), a)$. Let $att(e(h(x)), a) = c''$.It holds that $c'' \mathtt{op}\ c$ and thus $G, e(h(x)) \models @_a \mathtt{op} c$, as needed.

($\Rightarrow$). We show the contrapositive. Suppose there is no a-homomorphism from Q to P. We then construct a counter-example with the same structure as P. Let $P.y$ be the subtree of P rooted in y and $Q.x$ the subtree of Q rooted in x. By induction on the depth of $P.y$ we show

(IH) If there is no homomorphism from $Q.x$ to $P.y$, then there exists a tree T such that $T \models P.y$ and $T \not\models Q.x$.

Base of induction: $depth(P.y) = 0$. Then there is no homomorphism from $Q.x$ to $P.y$ if either

- there exists a label p such that either $p \in l'(x)$ and $p \notin l(y)$.We define $T = (\{y\}, \emptyset, y, \rho)$, where the labeling function $\rho(y) = l(y)$. By construction, we have $T \models P.y$. It also holds that $T \not\models Q.x$. Indeed, otherwise it would hold that $l'(x) \subseteq \rho(y)$, which contradicts with the definition of ρ.
- there exists $@_a \mathtt{op} c \in l'(x)$ such that no $@_a \mathtt{op}' c' \in l(y)$ or $C \not\models @_a \mathtt{op}\ c$, where C is the set of constraints of a-attribute in $l(y)$. In the first case we take $T = (\{y\}, \emptyset, y, \rho, att)$ with $\rho(y) = l(y)$, $att(y, a)$ undefined and $att(y, b) = c_b$ for every $b \neq a$ such that c_b satisfies the constraint of b in $l(y)$. By construction, $T \models P.y$ and $T \not\models Q.x$. In the second case there is a constant c'' that satisfies C but not $c'' \mathtt{op}\ c$. Thus, $T \models P.y$ and $T \not\models Q.x$.
- Or there exists x' such that $(x, x') \in E'$. In this case, we define $T = (\{y\}, \emptyset, y, \rho)$, where $\rho(y) = l(y)$. By construction we have that $T \models P.y$. Moreover, $T \not\models Q.x$ since there is no child of the image of x in T.

Induction step: $depth(P.y) > 0$ and there is no a-homomorphism from $Q.x$ to $P.y$. It is because either

- there exists a label p such that $p \in l'(x)$ and $p \notin l(y)$. We define $T = (N_T, E_T, y, \rho, att)$, where $N_T = \{n \in N \mid P \models yE^*n\}$, i.e., the nodes below y in P, $E_T = E|_{N_T^2}$, the labeling ρ is defined as l, and att is defined canonically to satisfy constraints in P.

  It is easy to see that $T \models P.y$ and $T \not\models Q.x$.

- there exists $@_a\mathtt{op}c \in l'(x)$ such that either (i) no $@_a\mathtt{op}'c'$ is in $l(y)$ or (ii) $C \not\models @_a\mathtt{op}\ c$, where C is the set of constraints of a-attribute in $l(y)$. We define $T = (N_T, E_T, y, \rho, att)$, where $N_T = \{n \in N \mid P \models yE^*n\}$, i.e. the nodes below y in P, $E_T = E|_{N_T^2}$, the labeling ρ is defined as l. The attribute function is defined as follows. If $@_b\mathtt{op}\ c$, appeared in a node z in P, then $att(z, b) = c_b$ such that c_b satisfies the constraints of b-attribute in $l(z)$. Additionally, we take $att(y, a)$ as undefined in case (i) and the constant witnessing $C \not\models @_a\mathtt{op}c$ in case (ii). It holds that $T \models P.y$ and $T \not\models Q.x$.

- Or there exists x' such that $(x, x') \in E'$ and for all y_i with $(y, y_i) \in E$ it holds that there is no a-homomorphism from $Q.x'$ to $P.y_i$. Since $depth(P.y_i) < depth(P.y)$, by the induction hypothesis, it holds that for every child y_i of y there exists a tree $T_i = (N_i, E_i, r_i, \rho_i, att_i)$ such that $T_i \models P.y_i$ and $T_i \not\models Q.x'$. We can assume that these trees are pairwise disjoint. We then define the tree $T = (N_T, E_T, y, \rho, att)$ such that $N_T = \{y\} \cup \bigcup_i N_i$, $E_T = \bigcup_i (E_i \cup \{\langle y, y_i\rangle\})$ and

  $$\rho(u) = \begin{cases} l(y) & \text{if } u = y, \\ \rho_i(u) & \text{if } T_i \models y_iE^*u. \end{cases}$$

  and $att = \cup_i att_i$. We claim that $T \models P.y$ and $T \not\models Q.x$. The former is by definition of T. For the latter, suppose $e : Q.x \to T$ is an embedding. In particular, that means that e is an embedding of $Q.x'$ to one of T_i, where y_i is a child of y. Thus, $T_i \models Q.x'$, which is a contradiction.

Thus, since there is no homomorphism from $Q.r'$ to $P.r$, there exists a tree such that $T \models P$ and $T \not\models Q$. □

Corollary 4.A.2. *Containment for child-only tree patterns with attribute value comparisons is solvable in* PTIME.

Proof. We first check consistency of the input tree patterns which can be done in PTIME. If one of the tree patterns is inconsistent, the problem is trivial. Otherwise, according to the theorem above, it is enough to check existence of a homomorphism. The latter can be done in PTIME, since $<$ can be decided in PTIME. □

5 Containment for Conditional Tree Patterns

We have already seen tree patterns in Chapter 3 and Chapter 4. In this chapter we study an expansion of tree patterns. A Conditional Tree Pattern (CTP) expands a tree pattern with labels attached to the descendant edges. These labels can be XML element names or Boolean CTP's. The meaning of a descendant edge labelled by A and ending in a node labelled by B is a path of child steps ending in a B node such that all intermediate nodes are A nodes. In effect this expresses the *until B, A holds* construction from temporal logic.

This chapter studies the containment problem for CTP. For tree patterns (TP), this problem is known to be coNP-complete (Miklau and Suciu, 2004). We show that it is PSPACE-complete for CTP. This increase in complexity is due to the fact that CTP is expressive enough to encode an unrestricted form of label negation: $* \setminus a$, meaning "any node except an a-node". Containment of TP expanded with this type of negation is already PSPACE-hard.

CTP is a positive, forward, first order fragment of Regular XPath. Unlike TP, CTP expanded with disjunction is not equivalent to unions of CTP's. But like TP, CTP is a natural fragment to consider: CTP is closed under intersections and CTP with disjunction is equally expressive as positive existential first order logic expanded with the until operator.

5.1 Introduction

Tree Patterns are one of the most studied languages for XML documents and used in almost all aspects of XML data managment. Tree patterns are a natural language: over trees, unions of tree patterns are equally expressive as positive first order logic (Benedikt et al., 2005). Also, like relational conjunctive queries, the semantics of TP's can be given by embeddings from patterns to tree models (Amer-Yahia et al., 2002). Equivalence and containment of TP's is decidable in PTIME for several fragments (Amer-Yahia et al., 2002), and coNP complete in general (Miklau and Suciu, 2004). Tree patterns can be represented as trees, as in Figure 5.1, or be given a natural XPath-like syntax.

In this chapter, we study the expansion of tree patterns with the conditional descendant axis. We call this expansion *Conditional Tree Patterns*, abbreviated as CTP. Where the descendant axis in TP can be written as the transitive reflexive closure of the XPath step `child::*`, the conditional descendant axis is the transitive closure of

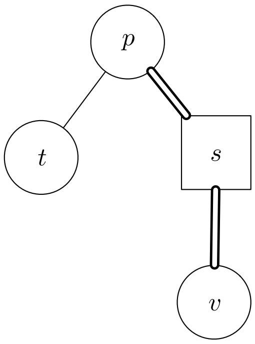


Figure 5.1: The tree pattern corresponding to the XPath expression `/p[t]//s[.//v]`. The node in the square box denotes the output node.

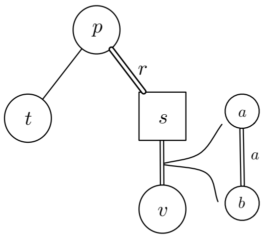


Figure 5.2: Conditional tree pattern corresponding to (5.1).

$\texttt{child} :: \mathtt{p}[\mathtt{F_1}][\mathtt{F_2}]\ldots[\mathtt{F_n}]$, where $n \geq 0$ and each F_i is an XPath expression which might contain the conditional descendent axis itself, followed by a child step. Syntactically, the expansion is straightforward: in the tree representation add labels representing conditional tree patterns themselves to the edges. Figure 5.2 contains an example which is equivalent to the following XPath-like formula in which we use $(\cdot)^*$ to denote the reflexive transitive closure of a path formula:

$$\begin{aligned}&\texttt{self} :: \mathtt{p}[\texttt{child} :: \mathtt{t}]/(\texttt{child} :: \mathtt{r})^*/\texttt{child} :: \mathtt{s}\\&\quad[(\texttt{child} :: \mathtt{a}[(\texttt{child} :: \mathtt{a})^*/\texttt{child} :: \mathtt{b}])^*/\texttt{child} :: \mathtt{v}]\end{aligned} \tag{5.1}$$

The edge in Figure 5.2 labeled by r corresponds to the expression $(\texttt{child} :: \mathtt{r})^*/\texttt{child} :: *$, and merely states that all intermediate nodes are labeled by r. The other labeled edge shows a nesting of patterns, corresponding to a nested transitive closure statement: all intermediate nodes have to be labeled by a and moreover have to start an a-labeled path ending in a b-node.

Conditional tree patterns are the forward fragment of Conditional XPath (Marx, 2005) without disjunction and negation. The conditional descendant axis is closely related to the strict until operator from temporal logic (Libkin and Sirangelo, 2010; Marx, 2005).

Main results

Our main results concern the expressivity of CTP and the complexity of the containment problem. We consider two types of models: the standard XML trees in which each node has exactly one label, and trees in which nodes can have an arbitrary number of labels. These latter, called *multi-labeled trees*, are the models considered in temporal logic. All our results hold for both semantics. Models with multiple labels are a convenient logical abstraction for reasoning about tree patterns expanded with attribute value equalities. These are expressions of the form $@_a = c$, where a is an attribute, c a constant, meaning that it holds at a node if and only if the value of the a-attribute of the node equals c. (With that we can express XPath formulas like `//table[@border = '1']`).

Containment of tree patterns has been studied in (Amer-Yahia et al., 2002; Miklau and Suciu, 2004; Neven and Schwentick, 2006). The most relevant results for us are that containment of TP's is coNP-complete in general (Miklau and Suciu, 2004) and PSPACE-complete when the domain of labels is finite (Neven and Schwentick, 2006). We show that containment of CTP's is PSPACE-complete (with both finite and infinite domains of labels). Interestingly, this increase in complexity is due to the fact that CTP is expressive enough to encode an unrestricted form of label negation: $* \setminus a$, meaning "any node except an a-node". We show that containment of TP's expanded with this form of negation is already PSPACE-hard. The matching upper bound for CTP containment is easily obtained by a translation into Existential CTL (Kupferman and Vardi, 2000). As a contrast we consider the expansion of TP with a *safe* form of propositional negation $n \setminus a$, which selects nodes with the label containing n and not a, instead of $* \setminus a$ (Bárány et al., 2011). Note that this construct only makes sense on models with multiple labels. With respect to expressivity, we show that most results for TP can be generalized to CTP. CTP's can be interpreted in trees by generalizing the TP-embeddings to simulations known from temporal logic. Similarly to the characterization for TP in (Benedikt et al., 2005), we show that CTP's with disjunction and union are equally expressive as positive first order logic expanded with an *until* operator. From this we obtain that like TP's, CTP's are closed under taking intersections.

Organization

The chapter is organized as follows. This section is continued with a few more comparisons between TP and CTP and a motivating example. Section 5.2 contains preliminaries. Section 5.3 contains the expressivity results and Section 5.4 the results on the complexity of the containment problem. We end with conclusions and open questions.

Comparing logical properties of TP and CTP

A characteristic difference between TP and Relational Conjunctive Queries (CQ) is the disjunction property: if $A \models B \vee C$, then $A \models B$ or $A \models C$. This holds for CQ, but not for TP. A counterexample is $//p \models /p \ \vee \ /*//p$. The languages TP and CTP differ on the following:

Unions TP expanded with disjunction is equally expressive as unions of TP (Benedikt et al., 2005). However, CTP with disjunction is more expressive than unions of CTP.

Countermodels If containment between two TP's fails, there is a countermodel for it of polynomial size (Miklau and Suciu, 2004). Countermodels for CTP containment may be exponential.

Complexity TP containment is coNP-complete (Miklau and Suciu, 2004) and CTP containment is PSpace-complete. For both languages these results remain true if we add disjunction to the language (for TP see (Neven and Schwentick, 2006)).

However, there are a few useful technical results that TP and CTP share.

Monotonicity TP and CTP formulas are preserved under extensions of models at the leaves (i.e. when the original model is a sub*tree* of the extension). This means that if a TP (CTP) formula holds in a tree, then it holds in the extensions of the tree.

Multiple output nodes The containment problem for TP and CTP formulas with multiple output nodes can be reduced to containment of Boolean TP and CTP formulas (i.e., when the formula has a single output node which is the root).

Containment for unions Containment for unions of TPs can be reduced in PTime to checking a set of containments between TP formulas (Miklau and Suciu, 2004). This reduction can be seen as a weak disjunction property. A similar property holds for CTP (see Proposition 5.4.2).

Multi-labeled models There is a PTime reduction from the containment problem over trees in which each node has exactly one label to the containment problem over multi-labeled trees. This holds for both TP and CTP.

For TP, most of these results are in (Miklau and Suciu, 2004). Here we show how their proofs can be generalized to CTP.

Motivation

Tree patterns exhibit a nice tradeoff between expressive power and the complexity of static analysis. However, there are natural scenarios where tree patterns are not powerful enough, e.g. in Example 5.1.1. The conditional axis gives us more querying capabilities while preserving some of the nice properties of tree patterns (see the comparison above).

Example 5.1.1. Conditional tree patterns are used to query tree shaped structures. As an example, take the tree structure of the UNIX file system. In this file system, every file and directory has different access permissions (read, write or execute) for different type of users (the user, the group and others). Thus, the file system can be modeled as a tree where each node corresponds to a directory or a file (labeled by "dir" and "file" respectively) and has a required attribute for each pair $(user, access\ right)$ which takes values from $\{0, 1\}$. A file can only be a leaf in the tree.

Assume we want to ask for all the files that are readable by the user. This means that we are looking for precisely those files for which the following permissions hold:

- the file is readable for the user,

- the directory in which the file resides is both readable and executable for the user,
- the same holds recursively for all the directories from the root to the file.

This query can be neatly expressed as a CTP path formula:

$$/(\texttt{child} :: \texttt{dir}[@_{(\texttt{user},\texttt{read})} = 1][@_{(\texttt{user},\texttt{execute})} = 1])^{*}/\texttt{child} :: \texttt{file}[@_{(\texttt{user},\texttt{read})} = 1].$$

Additionally, CTP can express non-trivial constraints over the tree representing the file system. For instance, the formula

$$//\texttt{self} :: \texttt{file}[@_{(\texttt{other},\texttt{read})} = 1] \to / \\ /(\texttt{child} :: \texttt{dir}[@_{(\texttt{other},\texttt{read})} = 1][@_{(\texttt{other},\texttt{execute})} = 1])^{*}/*$$

imposes the constraint that for all files that are readable it holds that every directory on the path from the root to this file must be both readable and executable by others.

It is known that the conditional axis cannot be expressed by tree patterns or in Core XPath (Marx, 2005). On the other hand, the queries from Example 5.1.1 can be expressed in the positive forward fragment of Regular XPath, which is more expressive than CTP. However, we believe this additional expressive power leads to an increase in the complexity (of containment) than for conditional tree patterns.

5.1.1 Related work

The complexity of tree patterns is studied in a number of papers and since (Neven and Schwentick, 2006) a virtually complete picture exists for the complexity of the containment problem for positive fragments of XPath. Miklau and Suciu (2004) show that the complexity of the containment problem for TP with filters, wildcard and descendant is coNP-complete. Containment and equivalence for fragments of TP were studied before. The most interesting result is that containment for TP without the wildcard is in PTIME (Amer-Yahia et al., 2002).

Our conditional tree patterns are a first order fragment of conjunctive regular path queries. Calvanese et al. (2000) show that containment of these queries is EXPSPACE-complete, but these are interpreted on general graph models.

Conditional XPath (Marx, 2005) and conditional tree patterns are closely related to branching time temporal logic CTL (Clarke et al., 1986). The conditional child axis and the strict until connective are interdefinable. Kupferman and Vardi (2000) show that the containment problem for $\exists$CTL, which is the restriction of CTL to formulas having only the $\exists$ path quantifier in front of them, is a PSPACE-complete problem. The positive $\exists$CTL-fragment without until was also studied by Miklau and Suciu (2004). They show that the containment problem for this fragment is equivalent to the TP-containment problem and thus also coNP-complete.

This chapter studies the containment problem without presence of schema information. A number of complexity results for containment in TP w.r.t DTDs are given in (Neven and Schwentick, 2006). In particular, containment for TP with filters, the wildcard and descendent w.r.t. DTD is EXPTIME-complete. This hardness result together with an EXPTIME upper bound for Conditional XPath (Marx, 2005), which contains CTP, gives us EXPTIME-completeness of containment for CTP in the presence of DTDs.

5.2 Preliminaries

We briefly review the basic definitions of XML trees, Regular XPath and its semantics. Then we present Tree Patterns and Conditional Tree Patterns as sublanguages of Regular XPath. Tree patterns have an alternative semantics in terms of embeddings (Miklau and Suciu, 2004). We give such an "embedding semantics" for Conditional Tree Patterns using "until-simulations" in Section 5.3.

5.2.1 Trees

We work with node-labeled unranked finite trees, where the node labels are elements of an infinite set Σ. Formally, a tree over Σ is a tuple (N, E, r, ρ), where N, the set of nodes of the tree, is a prefix closed set of finite sequences of natural numbers, $E = \{(\langle n_1, \ldots, n_k\rangle, \langle n_1, \ldots, n_k, n_{k+1}\rangle) \mid \langle n_1, \ldots, n_{k+1}\rangle \in N\}$ is the edge or child relation, r is the root of the tree, that is the empty sequence, and ρ is the function assigning to each node in N a finite subset of Σ. We refer to a tree over Σ just as a tree if Σ is clear from the context.

Trees in which $\rho(\cdot)$ is always a singleton are called *single-labeled* or *XML trees*. Trees without this restriction are called *multi-labeled trees*.

We denote by E^+ the descendant relation, which is the transitive closure of the edge relation E, and by E^* the reflexive and transitive closure of E, and by $E(x)$ the set of all children of the node x. A node n is a *leaf* if $E(n)$ is empty. A *path* from a node n to a node m is a sequence of nodes $n = n_0, \ldots, n_k = m$, with $k > 0$, such that for each $i \leq k$, $n_{i+1} \in E(n_i)$. We call a *branch* any maximal path starting from the root. If mE^+n, m is called an *ancestor* of n, and if mEn, m is called the *parent* of n.

If n is in N, by $T.n$ we denote the subtree of T rooted at n. A *pointed tree* is a pair T, n where T is a tree and n is a node of T. The *height* of a pointed tree T, n is the length of the longest path in $T.n$.

Given two trees $T_1 = (N_1, E_1, r_1, \rho_1)$ and $T_2 = (N_2, E_2, r_2, \rho_2)$ such that N_1 and N_2 are disjoint, we define the result of *fusion* of T_1 and T_2, denoted as $T_1 \oplus T_2$, as the tree obtained by joining the trees T_1 and T_2 without the roots under a new common root labeled by the union of the labels of the roots of T_1 and T_2. Formally, $T_1 \oplus T_2$ is the tree $T = (N, E, r, \rho)$, where $N = (N_1 \setminus \{r_1\}) \cup (N_2 \setminus \{r_2\}) \cup \{r\}$, $E = (E_1 \setminus \{\langle r_1, n\rangle \mid n \in N_1\}) \cup (E_2 \setminus \{\langle r_2, n\rangle \mid n \in N_2\}) \cup \{\langle r, n\rangle \mid \langle r_1, n\rangle \in E_1$ or $\langle r_2, n\rangle \in E_2\}$ and

$$\rho(n) = \begin{cases} \rho_1(r_1) \cup \rho_2(r_2) & \text{if } n = r, \\ \rho_1(n) & \text{if } n \in N_1 \setminus \{r_1\}, \\ \rho_2(n) & \text{if } n \in N_2 \setminus \{r_2\}. \end{cases}$$

5.2.2 XPath and Tree Patterns

We define Tree Patterns and Conditional Tree Patterns as sublanguages of Regular XPath (ten Cate, 2006).

Definition 5.2.1 (Forward Regular XPath)**.** *Let Σ be an infinite domain of labels. Forward Regular XPath,* RXPath *for short, consists of node formulas φ and path formulas α which are defined by the following grammar*

$$\begin{array}{lcl} \varphi & ::= & p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle\alpha\rangle\varphi \\ \alpha & ::= & \downarrow \mid ?\varphi \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha^*, \end{array}$$

where $p \in \Sigma$.

We will use α^+ as an abbreviation of $\alpha;\alpha^*$.

For the semantics of RXPath, given a tree $T = (N, E, r, \rho)$ over Σ, the relation $[\![\alpha]\!]_T \subseteq N \times N$ for a path expression α and the satisfaction relation $\models$ between pointed trees and node formulas are inductively defined as follows:

- $[\![\downarrow]\!]_T = E$,
- $[\![?\varphi]\!]_T = \{(n,n) \in N \times N \mid T, n \models \varphi\}$,
- $[\![\alpha;\beta]\!]_T = [\![\alpha]\!]_T \circ [\![\beta]\!]_T$,
- $[\![\alpha \cup \beta]\!]_T = [\![\alpha]\!]_T \cup [\![\beta]\!]_T$,
- $[\![\alpha^*]\!]_T = ([\![\alpha]\!]_T)^*$,

and

- $T, n \models \top$,
- $T, n \models p$ iff $p \in \rho(n)$,
- $T, n \models \neg\varphi$ iff $T, n \not\models \varphi$,
- $T, n \models \varphi \wedge \psi$ iff $T, n \models \varphi$ and $T, n \models \psi$,
- $T, n \models \varphi \vee \psi$ iff $T, n \models \varphi$ or $T, n \models \psi$,
- $T, n \models \langle\alpha\rangle\varphi$ iff there is a node m with $(n, m) \in [\![\alpha]\!]_T$ and $T, m \models \varphi$.

Sometimes we will write $T \models \varphi$ to denote $T, r \models \varphi$. If the latter holds, we say that T is a *model* of φ.

(Conditional) Tree Patterns

Tree patterns are the conjunctive fragment of RXPath without unions of paths and with a strongly restricted Kleene star operation. Node formulas correspond to Boolean tree patterns, and path formulas to tree patterns with one output node.

In the following definitions we again let Σ be an infinite domain of labels. We define Tree Patterns by restricting the syntax of RXPath as follows:

Definition 5.2.2 (Tree Pattern). *Tree Patterns (*TP*) consist of node formulas φ and path formulas α defined by the following grammar:*

$$\begin{array}{lcl} \varphi & ::= & p \mid \top \mid \varphi \wedge \varphi \mid \langle\alpha\rangle\varphi \\ \alpha & ::= & \downarrow \mid ?\varphi \mid \alpha;\alpha \mid \downarrow^+, \end{array}$$

where $p \in \Sigma$.

For example, the tree pattern from Figure 5.1 can be written as the path formula $\alpha = ?(p \wedge \langle\downarrow\rangle t); \downarrow^+; ?(s \wedge \langle\downarrow\rangle v)$. Conditional Tree Patterns can also be defined by restricting the syntax of RXPath, where we allow conditional descendant paths.

Definition 5.2.3 (Conditional Tree Pattern)**.** *Conditional Tree Patterns (*CTP*) consist of node formulas φ and path formulas defined by the following grammar:*

$$\begin{array}{lll} \varphi & ::= & p \mid \top \mid \varphi \wedge \varphi \mid \langle\alpha\rangle\varphi \\ \alpha & ::= & \downarrow \mid ?\varphi \mid \alpha;\alpha \mid (\downarrow; ?\varphi)^*; \downarrow \ , \end{array}$$

where $p \in \Sigma$.

The tree from Figure 5.2 can be written as the path formula

$$?(p \wedge \langle\downarrow\rangle t); (\downarrow; ?r)^*; \downarrow; ?(s \wedge \langle(\downarrow; ?(a \wedge \langle(\downarrow; ?a)^*; \downarrow\rangle b))^*; \downarrow\rangle v).$$

Note that the node formula $\langle(\downarrow; ?\varphi)^*; \downarrow\rangle\psi$ is exactly the strict until $\exists U(\psi, \varphi)$ from branching time temporal logic CTL (Kupferman and Vardi, 2000). We will abbreviate this formula simply as $\langle\varphi^+\rangle\psi$. Because of the equivalences $\langle\alpha;\beta\rangle\varphi \equiv \langle\alpha\rangle\langle\beta\rangle\varphi$ and $\langle?\varphi\rangle\psi \equiv \varphi \wedge \psi$, CTP node formulas can be given the following equivalent definition:

$$\varphi \quad ::= \quad p \mid \top \mid \varphi \wedge \varphi \mid \langle\downarrow\rangle\varphi \mid \langle\downarrow^+\rangle\varphi \mid \langle\varphi^+\rangle\varphi.$$

Even though the syntax is slightly different, CTP is the conjunctive forward only fragment of Conditional XPath (Marx, 2005).

Expansions

We consider two expansions of TP and CTP, with negated labels and with disjunction in node formulas together with union in paths. Negated labels is a restricted type of negation where only the construct $\neg p$ is allowed in the node formulas. We denote expansions of the language L with one or two of these features by L^S for $S \subseteq \{\neg, \vee\}$.

Query evaluation

In (Gottlob et al., 2005a) it was shown that the query evaluation problem for Core XPath is PTIME-complete in the combined complexity. As noted in (Marx, 2005), using results on model checking for Propositional Dynamic Logic, this can be extended to Regular XPath, and thus to all our defined fragments.

Fact 1 (Marx (2005))**.** *Let T be a tree, n_1, n_2 nodes in T, and α a Regular XPath path formula. The problem whether $(n_1, n_2) \in [\![\alpha]\!]_T$ is decidable in time $O(|T| \times |\alpha|)$ with $|T|$ the size of the tree and $|\alpha|$ the size of the formula.*

5.2.3 Containment

As we are considering two kinds of expressions, path and node expressions, we have different notions of containment.

Definition 5.2.4. *Let φ and ψ be two* RXPath*-node formulas. We say that φ is* contained *in ψ, notation $\varphi \subseteq \psi$, if for every T, n, $T, n \models \varphi$ implies $T, n \models \psi$. Let α and β be two* RXPath*-path formulas. We say that*

- α *is* contained as a binary query *in β, denoted $\alpha \subseteq^2 \beta$, if for any tree T, $[\![\alpha]\!]_T \subseteq [\![\beta]\!]_T$,*
- α *is* contained as a unary query *in β, denoted $\alpha \subseteq^1 \beta$, if for any tree T with root r, and any node n, $(r,n) \in [\![\alpha]\!]_T$ implies $(r,n) \in [\![\beta]\!]_T$.*

Containment over single-labeled trees is denoted by $\subseteq$, and containment over multi-labeled models by $\subseteq_{\mathsf{ML}}$.

Luckily, these three notions are closely related, and containment of path formulas can be reduced to containment of node formulas and vice versa (cf. also (Miklau and Suciu, 2004; Neven and Schwentick, 2006)).

Proposition 5.2.1. *Let α and β be two* RXPath*-path formulas, φ and ψ* RXPath*-node formulas, in which negation is restricted to labels only.*

(i) $\alpha \subseteq^2 \beta$ iff $\alpha \subseteq^1 \beta$,

(ii) Let p be a label not occurring in α or in β. Then, $\alpha \subseteq^2 \beta$ iff $\langle\alpha\rangle\langle\downarrow\rangle p \subseteq \langle\beta\rangle\langle\downarrow\rangle p$,

(iii) $\varphi \subseteq \psi$ iff $?\varphi \subseteq^2 ?\psi$.

All the above items also hold for the case of multi-labeled trees.

Proof. (i) ($\Rightarrow$) Let $T = (N, E, r, \rho)$ be a tree such that $(r,n) \in [\![\alpha]\!]_T$. By the assumption, we have $[\![\alpha]\!]_T \subseteq [\![\beta]\!]_T$. This implies that $(r,n) \in [\![\beta]\!]_T$, which was required to show.

($\Leftarrow$) Let $T = (N, E, r, \rho)$ be a tree, n_1 and n_2 in N such that $(n_1, n_2) \in [\![\alpha]\!]_T$. Since α is from the forward fragment of Regular XPath, we have that $(n_1, n_2) \in [\![\alpha]\!]_{T.n_1}$. Using the assumption, we have $(n_1, n_2) \in [\![\beta]\!]_{T.n_1}$. Then by monotonicity, we obtain $(n_1, n_2) \in [\![\beta]\!]_T$, which was desired.

(ii) ($\Rightarrow$) Suppose $\alpha \subseteq^2 \beta$, and let $T, n \models \langle\alpha\rangle\langle\downarrow\rangle p$. Thus there is a descendant m of n in T such that $(n,m) \in [\![\alpha]\!]_T$ and $T, m \models \langle\downarrow\rangle p$. By the hypothesis we therefore have $(n,m) \in [\![\beta]\!]_T$ and thus $T, n \models \langle\beta\rangle\langle\downarrow\rangle p$.

($\Leftarrow$) Assume $\langle\alpha\rangle\langle\downarrow\rangle p \subseteq \langle\beta\rangle\langle\downarrow\rangle p$. Consider a tree $T = (N, E, r, \rho)$ and a pair $(n,m) \in [\![\alpha]\!]_T$. Since p does not occur in α, we may assume that $T, n \not\models p$ for every node $n \in N$. We then define the tree $T' = (N', E', r, \rho')$, where

- $N' := N \cup \{x\}$,
- $E' := E' \cup \{\langle m, x\rangle\}$,
- $\rho'(y) := \begin{cases} p & \text{if } y = x, \\ \rho(y) & \text{otherwise.} \end{cases}$

By definition of T', we have $T', m \models \langle\downarrow\rangle p$ and $(n,m) \in [\![\alpha]\!]_{T'}$. Hence, $T', n \models \langle\alpha\rangle\langle\downarrow\rangle p$. By assumption, this implies that $T', n \models \langle\beta\rangle\langle\downarrow\rangle p$. Since p holds at x only, and x is a child of m, we have that $(n,m) \in [\![\beta]\!]_{T'}$. By definition of T', the path between n and m is also in T. Thus, $(n,m) \in [\![\beta]\!]_T$, as desired.

Item (iii) easily follows from the definitions. □

In light of Proposition 5.2.1, from now on we consider the containment problem of node formulas only. We now give an interesting example of CTP containment.

Example 5.2.1. Let us consider the CTP node formulas

$$\begin{aligned}\varphi &= \langle(\langle b^+\rangle d)^+\rangle\langle a^+\rangle b \text{ and}\\ \psi &= \langle(\langle c^+\rangle d)^+\rangle\langle a^+\rangle b.\end{aligned}$$

Although it is hard to see from a first glance, $\varphi \subseteq \psi$ holds. Indeed, in every model T of φ either there is a direct child of the root where $\langle a^+\rangle b$ holds, or there is a path $r = v_1, \dots, v_n$ in T to the node v_n where $\langle a^+\rangle b$ holds and $\langle b^+\rangle d$ holds in every node v_i $(1 < i < n)$ on the path. In the first case, ψ holds at the root since $\langle a^+\rangle b$ holds at the direct child.

Now let's consider the second case. Let $j \in \{2, \dots, n-1\}$ be the least number with the property that v_j has a non-empty b-path to the d-descendant. If there is no such number, then $\langle c^+\rangle d$ holds at every $v_i, 1 < i < n$ (as each of them has a direct d-child) and, thus, $T, r \models \psi$.

Assume there is such a j. Since v_j has a b-node as a child, we have that $T, v_j \models \langle a^+\rangle b$. Moreover, $\langle c^+\rangle d$ holds at each v_i for $i < j$ since v_i has a d-node as a child. Thus in this case we obtain that $T, r \models \psi$ holds too.

5.3 Expressivity

We extend the semantics of TP given by embeddings of queries into trees to CTP. Instead of embeddings we need a simulation known from temporal logic.

5.3.1 Interpreting Conditional Tree Patterns by simulations

The semantics of conditional tree patterns can be defined using simulations developed for LTL (Blackburn et al., 2001). These simulations generalize the embeddings for tree patterns from (Amer-Yahia et al., 2002; Miklau and Suciu, 2004) with an additional clause for checking the labels on the edges.

We start by defining the tree pattern analogues of CTP node and path expressions.

Definition 5.3.1. *A* conditional tree pattern *is a node and edge labeled finite tree* $(N, E, r, \bar{o}, \rho_N, \rho_E)$*, where* N *is the set of nodes of the tree,* $E \subseteq N \times N$ *is the set of edges,* r *is the root of the tree,* $\bar{o}$ *is a* k*-tuple (*$k > 0$*) of output nodes,* ρ_N *is the function assigning to each node in* N *a finite set of labels from* Σ *and* ρ_E *is the function assigning to each pair in* E *either* $\downarrow$ *or a Boolean conditional tree pattern.*
A Boolean conditional tree pattern *is a conditional tree pattern with a single output node which equals the root.*
A conditional tree pattern is said to have multiple output nodes *if the number* $k = |\bar{o}|$ *is greater than 1.*
A tree pattern *is a conditional tree pattern whose edges are only labeled by* $\downarrow$ *or* $\top$.

To be consistent with the pictorial representation of TP, in CTP an edge labeled with $\downarrow$ is drawn as a single line, while an edge labeled with a CTP node formula is drawn as a double line with a CTP as the label (e.g. as in Figure 5.2). The output nodes have the square shape.

CTP node and path expressions can be translated into (Boolean) conditional tree patterns with one output node and vice-versa. The translations are given in Appendix 5.A. We denote the equivalent (Boolean) conditional tree pattern of a CTP path or node expression α or φ by $\mathsf{c}(\alpha)$ and $\mathsf{c}(\varphi)$, respectively.

Next we generalize the notion of TP-embeddings (Miklau and Suciu, 2004) to CTP-simulations.

Definition 5.3.2. *Let $T = (N, E, r, \bar{o}, \rho_N, \rho_E)$ be a conditional tree pattern as in the previous definition and $T' = (N', E', r', \rho')$ a tree. A total function $f : N \to N'$ is called a* simulation *from the pattern T into the pointed tree T', r' if it satisfies the following properties:*

root preserving *$f(r) = r'$;*

label preserving *if $p \in \rho_N(n)$, then $p \in \rho'(f(n))$;*

child edge preserving *if nEn' and $\rho_E(n, n') = '\downarrow'$, then $f(n)Ef(n')$;*

conditional edge simulation *if nEn' and $\rho_E(n, n')$ is not equal to $\downarrow$, then $f(n)E'^{+}f(n')$ and for every x such that $f(n)E'^{+}xE'^{+}f(n')$ there is a simulation from the Boolean conditional tree pattern $\rho_E(n, n')$ into $T'.x$ (the subtree of T' rooted at x).*

When a pattern is a tree pattern, $\downarrow$ and $\top$ are the only labels on edges. For the label $\top$, the "conditional edge simulation" clause trivializes to checking that f is an embedding for the descendant edges. Simulations for tree patterns are thus equivalent to tree pattern embeddings (Amer-Yahia et al., 2002; Miklau and Suciu, 2004).

The next theorem states that simulations can be used to evaluate conditional tree patterns.

Theorem 5.3.3. *Let φ and α be a* CTP *node and path expression, respectively. Let T be a tree and n a node in T.*

(i) *$T, n \models \varphi$ if and only if there is a simulation from $\mathsf{c}(\varphi)$ into $T.n$.*

(ii) *$(n, n') \in [\![\alpha]\!]_T$ if and only if there is a simulation from $\mathsf{c}(\alpha)$ into $T.n$ which relates the output node of $\mathsf{c}(\alpha)$ to node n'.*

(iii) *Items (i) and (ii) also hold when T is an infinite tree.*

The proof is by mutual induction on the node and path expressions.

Remark 5.3.1. *The notions of (conditional) tree pattern and embedding for tree patterns and simulation for conditional tree patterns are easily extended to work for expansions of these languages with negated labels (denoted by* $\mathsf{TP}^{\neg}$ *and* $\mathsf{CTP}^{\neg}$*, respectively).*

For (conditional) tree patterns, add a second node labelling function $\rho_N^{\neg}(\cdot)$ that also assigns each node a finite set of labels. These are interpreted as the labels that are false at the node.

For the embeddings and simulations, in Definition 5.3.2 we add a clause stating that also negated labels are preserved:

negated label preserving *If $p \in \rho_N^{\neg}(n)$, then $p \notin \rho'(f(n))$.*

With these modifications, Theorem 5.3.3 also holds for $\mathsf{TP}^{\neg}$ *and* $\mathsf{CTP}^{\neg}$.

5.3.2 Expressivity characterization

In this section we give an expressivity characterization for CTP similar to the one for various fragments of XPath in (Benedikt et al., 2005). The exact logical characterization allows one to compare different fragments of (Regular) XPath and derive non-trivial closure properties such as closure under intersection. We show that CTP path formulas correspond to a natural fragment of first order logic (**FO**) and are closed under intersections.

For φ a formula, let $desc_\varphi(x, y)$ be an abbreviation of the "until" formula $desc(x, y) \wedge \forall z(desc(x, z) \wedge desc(z, y) \rightarrow \varphi(z))$. Let $\exists^+(child, desc_\varphi)$ be the fragment of first order logic built up from the binary relations $child$ and $desc$, label predicates $p(x)$ for each label $p \in \Sigma$ and equality $'='$, by closing under $\wedge$, $\vee$ and $\exists$ as well as under the rule:

$$\text{if } \varphi(x) \in \exists^+(child, desc_\varphi), \text{ then } desc_\varphi(x, y) \in \exists^+(child, desc_\varphi). \tag{5.2}$$

We use $\exists^+(child, desc_\varphi)(c, s)$ to denote $\exists^+(child, desc_\varphi)$ formulas with exactly two variables c and s free. Following Benedikt et al. (2005), we restrict the **FO** fragment to its downward fragment $\exists^+(child, desc_\varphi)[down](c, s)$ by requiring that every bound variable as well as s is syntactically restricted to be a descendant of c or equal to c.

Note that $\exists^+(child, desc_\varphi)[down](c, s)$ without the rule (5.2) is the logic $\exists^+(child, desc)[down](c, s)$ from Benedikt et al. (2005), and shown there to be equivalent to unions of tree patterns. Even though $\exists^+(child, desc_\varphi)[down](c, s)$ does not contain negation, not every formula is satisfiable (e.g. $desc(x, x)$).

We can now characterize conditional tree patterns in terms of $\exists^+(child, desc_\varphi)[down](c, s)$.

Theorem 5.3.4. *The following languages are equivalent in expressive power:*

- *unions of the false symbol and* CTP *paths which can have disjunctions in the node formulas*
- $\exists^+(child, desc_\varphi)[down](c, s)$.

Proof. Translating a CTP path formula α into an equivalent formula in $\exists^+(child, desc_\varphi)[down](c, s)$ can be done via a standard translation $TR_{xy}(\cdot)$. The translation is essentially the semantics of path and node formulas written in the first order language.

$$\begin{array}{lcl}
TR_{xy}(\emptyset) & = & child(x, y) \wedge x = y \\
TR_{xy}(\downarrow) & = & child(x, y) \\
TR_{xy}(?\varphi) & = & x = y \wedge TR_x(\varphi) \\
TR_{xy}(\alpha_1; \alpha_2) & = & \exists z.(TR_{xz}(\alpha_1) \wedge TR_{zy}(\alpha_2)), \\
 & & \text{where } z \text{ is a fresh variable} \\
TR_{xy}((\downarrow; ?\varphi)^*; \downarrow) & = & desc_{TR_z(\varphi)}(x, y) \\
TR_{xy}(\alpha_1 \cup \alpha_2) & = & TR_{xy}(\alpha_1) \vee TR_{xy}(\alpha_2) \\
 & & \\
TR_x(p) & = & p(x) \\
TR_x(\top) & = & x = x \\
TR_x(\varphi_1 \wedge \varphi_2) & = & TR_x(\varphi_1) \wedge TR_x(\varphi_2) \\
TR_x(\varphi_1 \vee \varphi_2) & = & TR_x(\varphi_1) \vee TR_x(\varphi_2) \\
TR_x(\langle\alpha\rangle\varphi) & = & \exists y.(TR_{xy}(\alpha) \wedge TR_y(\varphi)), \\
 & & \text{where } y \text{ is a fresh variable.}
\end{array}$$

By definition, $TR_{cs}(\alpha)$ is a formula in $\exists^+(child, desc_\varphi)[down](c,s)$ and equivalent to α.

For the other direction, let $\theta \in \exists^+(child, desc_\varphi)[down](c,s)$. We use our earlier notation $desc_\psi(x,y)$ for the "until" formulas. First we introduce two special formulas $\top(x)$ and $\bot(x)$ which stand for $x = x$ and $desc(x,x)$, respectively.

We will modify θ in several steps. First we replace $child(x,y)$ and $desc(x,y)$ by the equivalent $desc_\bot(x,y)$ and $desc_\top(x,y)$, respectively. Then eliminate all equalities by renaming variables. Bring all existential quantifiers inside θ to the front, bring the body into disjunctive normal form and distribute the disjunctions over the quantifiers. We then end up with a disjunction of formulas of the form $\exists\bar{x}\varphi(c,s)$, with φ a conjunction of formulas $desc_\psi(x,y)$, $p(x)$, $\top(x)$ and $\bot(x)$. If φ contains $\bot$ replace $\exists\bar{x}\varphi(c,s)$ with $\bot$.

As our target language is closed under unions and $\bot$, we only need to translate the "conjunctive queries" $\exists\bar{x}\varphi(c,s)$. Consider the graph of the variables in φ in which two variables x, y are related if φ contains an atom $desc_\psi(x,y)$. If the graph is cyclic, it cannot be satisfied on a tree and $\exists\bar{x}\varphi(c,s)$ is equivalent to $\bot$. If it is a tree, c will be the root. Assuming that we can rewrite all ψ in the atoms $desc_\psi(x,y)$, we can rewrite it as a conditional tree pattern. If ψ is a boolean combination of $p(z)$, $\top(z)$ and $\bot(z)$ atoms, this is not hard: Simply bring it into disjunctive normal form, and remove each conjunct containing $\bot$. If the result is not empty, then translate to a union of (trivial) conditional tree patterns. If it is empty, translate $desc_\psi(x,y)$ to a child step. If ψ contains subformulas of the form $desc_\varphi(x,y)$, we apply the current procedure to it.

Thus assume that the variable graph is a directed acyclic graph. We eliminate undirected cycles step by step. The length of a cycle equals the number of variables in it. Consider that the graph contains two paths π_1 and π_2 both going from x to y and without other common variables. We show that this subgraph is equivalent to a union of subgraphs with no or smaller cycles.

In the simplest case, both paths are of length 1 and thus consist of a $desc_\psi(x,y)$ atom. But then we can use the following equivalence to remove the cycle:

$$desc_\varphi(x,y) \wedge desc_\psi(x,y) \equiv desc_{\psi\wedge\psi}(x,y) \tag{5.3}$$

So assume the cycle looks like Figure 5.3.

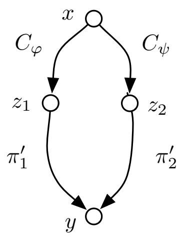


Figure 5.3: Undirected cycle in φ.

If this is satisfied on a tree (N, E, r, ρ) with assignment g, there are three possibilities: $g(z_1) = g(z_2)$, $g(z_1)E^+g(z_2)$ or $g(z_2)E^+g(z_1)$. In each case our original formula is equivalent to one with a smaller cycle. The three possibilities are

when $g(z_1) = g(z_2)$ $desc_{\varphi\wedge\psi}(x, z_1) \wedge z_1\pi'_1 y \wedge z_1\pi'_2 y$

when $g(z_1)E^+g(z_2)$ $desc_{\varphi\wedge\psi}(x, z_1) \wedge \psi(z_1) \wedge z_1\pi'_1 y \wedge desc_{\psi}(z_1, z_2) \wedge z_2\pi'_2 y$

when $g(z_2)E^+g(z_1)$ $desc_{\varphi\wedge\psi}(x, z_2) \wedge \varphi(z_2) \wedge z_2\pi'_2 y \wedge desc_{\varphi}(z_2, z_1) \wedge z_1\pi'_1 y.$

In the first case the length of the cycle decreased by two, in the two other cases by one.

Thus their disjunction is equivalent to the formula of Figure 5.3. We replace that formula by this disjunction, bring the result in disjunctive normal form and distribute the disjuncts out. We again have a disjunction of "conjunctive queries". As the new cycles are smaller, this procedure will terminate, and results in a (big) disjunction of trees. $\square$

An important consequence of this result is that CTP patterns are closed under intersection:

Theorem 5.3.5. *The intersection of two* CTP *paths is equivalent to* $\perp$ *or a union of* CTP *paths.*

Theorem 5.3.4 together with the translation of Regular XPath into $\mathbf{FO}^*(c, s)$ from ten Cate (2006) implies that every union of CTP patterns with disjunctions is in the intersection of first order logic and positive downward $\mathbf{FO}^*(c, s)$. It is an intriguing open problem whether the converse also holds.

5.4 Containment

Before we determine the exact complexity of the containment problem for CTP and expansions we prove a number of reductions. Most are generalizations from TP to CTP. The key new result is the encoding of negated labels in CTP.

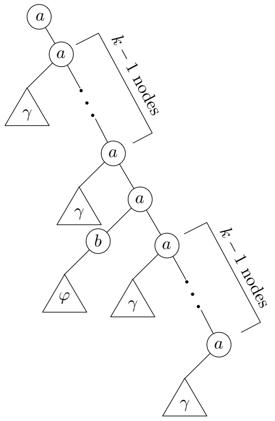
5.4.1 Containment preliminaries

The following reductions will be used later in our upper and lower bound proofs.

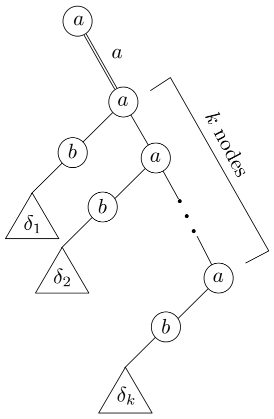
Multiple output nodes

The main difference between tree patterns and their XPath formulation is that tree patterns can have multiple output nodes. Kimelfeld and Sagiv (2008) (Proposition 5.2) show that for the TP containment problem the number of output nodes is not important: the problem can be PTIME reduced to a containment problem of Boolean TPs. This is achieved, given $\varphi, \psi \in$ TP, by adding a child labeled with a new label a_i to every output node X_i in both φ and ψ. Second, to every leaf that is not an output node or a newly added node, we add a child labeled with $\top$. The same result, using the same argument, holds for CTP.

Proposition 5.4.1. *Let* $S \subseteq \{\vee, \neg\}$. *For* CTP^S *patterns with multiple output nodes* φ, ψ *there are* PTIME *computable Boolean* CTP^S, φ', ψ' *such that* $\varphi \subseteq \psi$ *iff* $\varphi' \subseteq \psi'$. *The same holds for multi-labeled trees.*



5.4(a) Pattern φ'.



5.4(b) Pattern ψ.

Figure 5.4: Patterns φ' and ψ from Proposition 5.4.2.

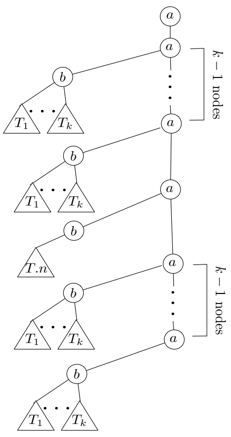


Figure 5.5: Model for pattern φ' from Proposition 5.4.2.

Disjunctions in the consequent

We now show that the containment problem for both unions of $\mathsf{CTP}^{\neg}$ and unions of $\mathsf{TP}^{\neg}$ can be reduced to containments without unions. This is useful in lower bound proofs, as we can use a union in the consequent to express multiple constraints.

The proof of the following proposition is a slight modification of the proof for TP in (Miklau and Suciu, 2004).

Proposition 5.4.2. *Let φ be a* CTP$^\neg$ *(*TP$^\neg$*) formula and Δ a finite set of* CTP$^\neg$ *(resp.* TP$^\neg$*) formulas. Then there are* PTIME *computable* CTP$^\neg$ *(*TP$^\neg$*) formulas φ' and ψ such that $\varphi \subseteq \bigvee \Delta$ iff $\varphi' \subseteq \psi$.*
If φ and Δ are in CTP *(*TP*), φ' and ψ are in* CTP *(*TP*) as well.*
The same holds for containment over multi-labeled trees.

Proof. First, for the case of TP$^\neg$, the proof from Miklau and Suciu (2004) can be readily applied here (using embeddings which also preserve negated labels, cf Remark 5.3.1).

Now we prove the proposition for CTP$^\neg$. For simplicity, we use the same letter to denote a CTP$^\neg$ formula and its corresponding tree pattern representation. Assume Δ is $\{\delta_1, \ldots, \delta_k\}$. If ι is a CTP$^\neg$ pattern, by ι^a we denote the CTP$^\neg$ pattern defined as having the root labeled by a, whose unique child is the root of ι. Consider new labels a and b occurring neither in φ nor Δ. Let $\gamma = \bigwedge_{0<\ell\leq k} \delta_\ell^b$. Note that γ is consistent if every δ_ℓ is consistent. Now define φ' as the CTP$^\neg$ tree pattern in Figure 5.4(a), and ψ as the CTP$^\neg$ tree pattern in Figure 5.4(b).

We verify that $\varphi \subseteq \bigvee \Delta$ iff $\varphi' \subseteq \psi$. Without loss of generality, we assume that each of $\varphi, \delta_1, \ldots, \delta_k$ is consistent.

For the direction from left to right, assume $\varphi \subseteq \bigvee \Delta$ and $T, n \models \varphi'$ for an arbitrary T, n. This means that there is a path $n = m_0 \ldots m_{2k-1}$ of nodes in T that are labeled with a and such that:

- for every $0 < \ell \leq k$: $T, m_i \models (\delta_\ell^b)^a$, with $i \in \{1, \ldots, k-1\} \cup \{k+1, \ldots, 2k-1\}$,
- $T, m_k \models (\varphi^b)^a$.

By the assumption, it follows that there is an index j such that $T, m_k \models (\delta_j^b)^a$. Then, by Theorem 5.3.3, there is a simulation f' from $(\delta_j^b)^a$ into $T.m_k$. The simulation f' can be extended to a simulation f from ψ into T, n in the obvious way such that f is a simulation from $(\delta_i^b)^a$ into T, m_{k-j+i} and the a-descendant edge in ψ is simulated on the path $m_0, \ldots m_{k-j+1}$. Thus, $T, n \models \psi$.

For the other direction, assume $\varphi' \subseteq \psi$ and $T, n \models \varphi$ for an arbitrary T rooted at n. As all δ_ℓ are consistent, there are trees $T_\ell, n_\ell \models \delta_\ell$, for each $\ell \in \{1, \ldots, k\}$. Let T' be the tree created from $T.n$ and the T_ℓ's depicted in Figure 5.5 with root r. Then $T', r \models \varphi'$. Because we assumed $\varphi' \subseteq \psi$, then also $T', r \models \psi$, which by Theorem 5.3.3 implies the existence of a simulation g from ψ into $T'.r$. In particular, the span of k a-nodes in ψ has to be simulated on an a-path of length k in T'. This means that for some $j \in \{1, \ldots, k\}$, g is a simulation from δ_j into $T.n$. By Theorem 5.3.3 again, $T, n \models \delta_j$, and thus $T, n \models \bigvee \Delta$.

Note that if φ and Δ are in CTP, then the constructed φ' and ψ are in CTP too. □

From single-labeled to multi-labeled trees

We now show that the containment problem over single-labeled trees can be reduced to a containment problem over multi-labeled trees.

Proposition 5.4.3. *Let $S \subseteq \{\neg, \vee\}$. Given* CTP^S *(*TP^S*) patterns φ, ψ, there are* PTIME *computable* CTP^S *(*TP^S*) patterns φ', ψ' such that $\varphi \subseteq \psi$ iff $\varphi' \subseteq_{\mathsf{ML}} \psi'$.*

Proof. Let $p_1, \ldots, p_n$ be the labels from Σ occurring in φ or ψ. We take φ'' as φ, and $\psi'' := \psi \vee \bigvee_{1 \le i < j \le n}(p_i \wedge p_j) \vee \bigvee_{1 \le i < j \le n}\langle\downarrow^+\rangle(p_i \wedge p_j)$. The disjuncts added to ψ ensure that every node of a counterexample is labeled with at most one label. Now we show that $\varphi \subseteq \psi$ iff $\varphi'' \subseteq_{\mathsf{ML}} \psi''$

($\Rightarrow$) We show the contraposition. Let T be a multi-labeled tree such that $T \models \varphi''$ and $T \not\models \psi''$. Then let T' be the tree obtained from T by restricting the labels $\rho'(v) := \rho(v) \cap \{p_1, \ldots, p_n\}$ for every node v. Since T does not satisfy ψ'', the label of every node in T' contains at most one symbol. Because we did not change the valuation of the labels $p_1, \ldots, p_n$, for each formula θ constructed from these tags, it holds that $T, v \models \theta$ iff $T', v \models \theta$. From this and the fact that the label of each node of T' is of size at most one, it follows that $T' \models \varphi''$ and $T' \not\models \psi''$. In order to make a single-labeled tree out of T', we add a dummy symbol q in the label of a node if its label in T' is empty. For the resulting tree T'' it holds $T'' \models \varphi$ and $T'' \not\models \psi$.

($\Leftarrow$) Let T be a single-labeled tree such that $T \models \varphi$ and $T \not\models \psi$. Then this tree can be considered as a multi-labeled tree. Moreover, $T \models \varphi''$ and $T \not\models \psi''$ since T does not satisfy any of the disjuncts in ψ''.

Our transformation does not introduce negations, but only disjunctions in ψ''. Thus the proposition is proven for the cases when S contains disjunction, where we take $\varphi' := \varphi''$ and $\psi' := \psi''$. For the remaining cases ($S \subseteq \{\neg\}$), although ψ'' contains disjunctions, by Proposition 5.4.2, there exist two PTIME computable CTP^S (TP^S) patterns φ' and ψ' such that $\varphi'' \subseteq_{\mathsf{ML}} \psi''$ iff $\varphi' \subseteq_{\mathsf{ML}} \psi'$. This concludes the proof. $\square$

Negated labels

We now show that CTP is expressive enough to encode label negation, as far as the containment problem is concerned.

The trick of the encoding lies in the fact that two nodes connected by a descendent edge are in either child or descendant of a child relation. Additionally, if we require that neither of these relations occur at the same time, we can faithfully encode label negation.

Proposition 5.4.4. *Let φ and ψ be $\mathsf{CTP}^{\neg}$ patterns. There are* PTIME *computable* CTP *patterns φ' and ψ' such that*

$$\varphi \subseteq \psi \text{ iff } \varphi' \subseteq \psi'.$$

This also holds for containment over multi-labeled trees.

Proof. Given a containment problem $\varphi \subseteq \psi$ in $\mathsf{CTP}^{\neg}$, we construct an equivalent containment problem $\varphi^\bullet \subseteq \psi^\circ$ with $\varphi^\bullet$ in CTP and ψ° a union of CTP patterns. Applying Proposition 5.4.2 then yields the desired result.

Let $p_1, \neg p_1, \ldots, p_n, \neg p_n$ be the labels appearing in φ and ψ and their negations. Let s be a new label and let ζ be the formula $\langle\downarrow\rangle(s \wedge \bigwedge_i \langle\downarrow^+\rangle p_i)$. We define the translation

$(\cdot)^\bullet$ inductively:

$$\begin{aligned}
\top^\bullet &= \zeta \\
(p_i)^\bullet &= \zeta \wedge \langle\downarrow\rangle(s \wedge \langle\downarrow\rangle p_i) \\
(\neg p_i)^\bullet &= \zeta \wedge \langle\downarrow\rangle(s \wedge \langle\downarrow\rangle\langle\downarrow^+\rangle p_i) \\
(\theta \wedge \sigma)^\bullet &= \theta^\bullet \wedge \sigma^\bullet \\
(\langle\downarrow\rangle\varphi)^\bullet &= \zeta \wedge \langle\downarrow\rangle\varphi^\bullet \\
(\langle\varphi^+\rangle\psi)^\bullet &= \zeta \wedge \langle\varphi^{\bullet +}\rangle\psi^\bullet.
\end{aligned}$$

The translation is defined in such a way that for any φ, $\varphi^\bullet$ implies ζ. Obviously $(\cdot)^\bullet$ is a PTIME translation. Let $\langle\downarrow^*\rangle\varphi$ denote $\varphi \vee \langle\downarrow^+\rangle\varphi$. Let AX be the disjunction of the following formulas:

$$\langle\downarrow^*\rangle(s \wedge \langle\downarrow^*\rangle\zeta) \tag{5.4}$$

$$\bigvee_i \langle\downarrow^*\rangle(p_i^\bullet \wedge (\neg p_i)^\bullet) \tag{5.5}$$

$$\bigvee_{i\neq j} \langle\downarrow^*\rangle(p_i^\bullet \wedge p_j^\bullet). \tag{5.6}$$

In our intended models, none of these disjuncts are true at the root. The disjunct (5.4) is technical, (5.5) states that no node makes the encodings of both p and $\neg p$ true, and (5.6) ensures that no node makes the encoding of two different labels true. We define $\psi^\circ = \psi^\bullet \vee AX$. In the case of multi-labeled trees the translation is the same, except that we do not include the disjunction (5.6) in the definition of ψ°. Note that although in ψ° we have disjunctions within the scope of a modality, we can (in PTIME) rewrite the formula into an equivalent union of disjunction free formulas.

We now show that $\varphi \subseteq \psi$ iff $\varphi^\bullet \subseteq \psi^\circ$.

($\Leftarrow$) We show the contrapositive. Given a single-labeled tree $T = (N, E, r, l)$ such that $T, r \models \varphi$ and $T, r \not\models \psi$, we build a single-labeled tree $T' = (N', E', r', l')$ as follows. The set of nodes N' is the maximal E'-connected subset of $N \times \{0, 1, 2, 3\} \times \{p_1, \ldots, p_n, \sharp\}$ that contains the root $r' = (r, 0, \sharp)$. The relation E' is defined as follows: $(n', 0, \sharp)$ is the parent of $(n, 0, \sharp)$ when n' is the parent of n in T; $(n, 0, \sharp)$ is the parent of $(n, 1, \sharp)$; $(n, 1, \sharp)$ is the parent of $(n, 2, p_i), i = 1, \ldots, n$, and $(n, 2, p_i)$ is the parent of $(n, 3, p_i), i = 1, \ldots, n$. The labeling l' is defined as follows: s only labels nodes of the form $(n, 1, \sharp)$. Then, p_i labels $(n, 2, p_i)$ if $T, n \models p_i$ and p_i labels $(n, 3, p_i)$ if $T, n \not\models p_i$. No other nodes are labeled by p_i. In all other cases, we label a node by a fresh label z.

It is clear from the definition that T' is a single-labeled tree. Note that $T' \not\models AX$ and that ζ is true at all nodes of type $(n, 0, \sharp)$, and only at these nodes. Let θ be a formula in variables $\{p_1, \ldots, p_n\}$, and $n \in N$. By induction on θ, we show $T, n \models \theta$ iff $T', (n, 0, \sharp) \models \theta^\bullet$.

- $\theta = \top$. This holds because by construction ζ is true at all nodes of type $(n, 0, \sharp)$.
- $\theta = p$. We have $T, n \models p \Leftrightarrow l(n) = p \Leftrightarrow l'(n, 2, p) = p \Leftrightarrow$ (since $(n, 2, p)$ is the only child of $(n, 1, \sharp)$ labeled with p and $(n, 1, \sharp)$ is labeled by s)

$T', (n, 1, \sharp) \models s \wedge \langle\downarrow\rangle p \Leftrightarrow$ (since $(n, 1, \sharp)$ is the only child of $(n, 0, \sharp)$ and ζ is always true at $(n, 0, \sharp)$) $T', (n, 0, \sharp) \models \zeta \wedge \langle\downarrow\rangle(s \wedge \langle\downarrow\rangle p) \Leftrightarrow T', (n, 0, \sharp) \models p^\bullet$.

- $\theta = \neg p$. We have $T, n \models \neg p \Leftrightarrow l(n) \neq p \Leftrightarrow l'(n, 3, p) = p \Leftrightarrow$ (since $(n, 3, p)$ is the only descendent of $(n, 1, \sharp)$ labeled with p and $(n, 1, \sharp)$ is labeled by s) $T', (n, 1, \sharp) \models s \wedge \langle\downarrow\rangle\langle\downarrow^+\rangle p \Leftrightarrow$ (since $(n, 1, \sharp)$ is the only child of $(n, 0, \sharp)$ and ζ is always true at $(n, 0, \sharp)$) $T', (n, 0, \sharp) \models \zeta \wedge \langle\downarrow\rangle(s \wedge \langle\downarrow\rangle\langle\downarrow^+\rangle p) \Leftrightarrow T', (n, 0, \sharp) \models (\neg p)^\bullet$.
- $\theta = \varphi_1 \wedge \varphi_2$. We have $T, n \models \varphi_1 \wedge \varphi_2 \Leftrightarrow T, n \models \varphi_1$ and $T, n \models \varphi_2 \Leftrightarrow$ (by the induction hypothesis) $T', (n, 0, \sharp) \models \varphi_1^\bullet$ and $T', (n, 0, \sharp) \models \varphi_2^\bullet \Leftrightarrow T', (n, 0, \sharp) \models \varphi_1^\bullet \wedge \varphi_2^\bullet \Leftrightarrow T', (n, 0, \sharp) \models (\varphi_1 \wedge \varphi_2)^\bullet$.
- $\theta = \langle\downarrow\rangle\varphi$. We only show the right to left direction. $T', (n, 0, \sharp) \models (\langle\downarrow\rangle\varphi)^\bullet \Leftrightarrow T', (n, 0, \sharp) \models \zeta \wedge \langle\downarrow\rangle\varphi^\bullet \Leftrightarrow \exists m \in N'$ such that $(n, 0, \sharp)E'm$ and $T', m \models \varphi^\bullet$. But then $T', m \models \zeta$ because $\varphi^\bullet$ implies ζ. Thus m must be of the form $(n', 0, \sharp)$ as only these make ζ true. But then $n' \in N$ and nEn' and we may apply the inductive hypothesis to get $T, n' \models \varphi$, and thus $T, n \models \langle\downarrow\rangle\varphi$.
- $\theta = \langle\varphi^+\rangle\psi$. We only show the right to left direction. $T', (n, 0, \sharp) \models (\langle\varphi^+\rangle\psi)^\bullet \Leftrightarrow T', (n, 0, \sharp) \models \zeta \wedge \langle\varphi^{\bullet +}\rangle\psi^\bullet \Leftrightarrow \exists m \in N'$ such that $(n, 0, \sharp)E'^+m$ and $T', m \models \psi^\bullet$ and $\forall m'.(n, 0, \sharp)E'^+m'E'^+m$ it holds $T', m' \models \varphi^\bullet$. Now, because $\varphi^\bullet$ implies ζ for all φ, the node m and all the nodes m' are of the form $(n', 0, \sharp)$ such that n' is in the original set N. Moreover they stand in the same way in the E relation. Thus we can apply the inductive hypothesis and obtain $T, n \models \langle\varphi^+\rangle\psi$.

As a special case, $T', (r, 0, \sharp)$ satisfies $\varphi^\bullet$ but not $\psi^\bullet$. Recall that $T', (r, 0, \sharp)$ does not satisfy the other disjuncts AX of ψ° either and thus, $T', (r, 0, \sharp) \models \varphi^\bullet$ and $T', (r, 0, \sharp) \not\models \psi^\circ$, as desired.

($\Rightarrow$) Again we show the contrapositive. Suppose there is a model $T = (N, E, r, l)$ satisfying $\varphi^\bullet$ but not ψ° at the root r. Then in particular $T, r \not\models AX$. Without loss of generality, we can assume that the simulation from (the conditional tree pattern corresponding to) $\varphi^\bullet$ into T is surjective. (Otherwise the image of $\varphi^\bullet$ is a subtree of T and thus by monotonicity ψ° cannot be satisfied at the root of this subtree.) In this model, as a consequence of the fact that $\varphi^\bullet$ implies ζ, every branch has an initial segment satisfying ζ, immediately followed by a node labeled by s, which is followed by a segment where ζ is never satisfied because of the first disjunct (5.4) of AX.

We define a tree $T' = (N', E', r', l')$ whose set of nodes N' consists of the nodes of T where ζ is satisfied, i.e. $N' = \{n \in N \mid T, n \models \zeta\}$; E' is simply the restriction of E to N' and $r' = r$. We define $l'(n) = p_i$ iff $T, n \models p_i^\bullet$, and if $T, n \not\models p_i^\bullet$ for any i, then we label n with a fresh variable z. This definition of l' is well-defined because the falsity of the disjunction (5.6) in AX ensures that each node makes at most one $p_i^\bullet$ true in T.

Let θ be a formula in variables $\{p_1, \ldots, p_n\}$, and $n \in N$. By induction on θ, we show that

$$T, n \models \theta^\bullet \text{ iff } T', n \models \theta. \tag{5.7}$$

- $\theta = \top$. Then $T, n \models \top^\bullet \Leftrightarrow$ (by definition of $(\cdot)^\bullet$) $T, n \models \zeta \Leftrightarrow$ (by definition of N') $\Leftrightarrow T', n \models \top$.

- $\theta = p$. Then $T, n \models p^\bullet \Leftrightarrow$ (by definition of the labeling l') $l'(n) = p \Leftrightarrow T', n \models p$.
- $\theta = \neg p$. If $T, n \models (\neg p)^\bullet$, then by the falsity of AX, $T, n \not\models p^\bullet$, and thus $l'(n) \neq p$ and $T', n \not\models p$ and thus $T', n \models \neg p$. Conversely, $T', n \models \neg p \Leftrightarrow l'(n) \neq p \Leftrightarrow T, n \not\models p^\bullet$. But $T, n \models \zeta$ and thus either $p^\bullet$ or $(\neg p)^\bullet$ must hold at T, n. Thus $T, n \models (\neg p)^\bullet$.
- $\theta = \varphi_1 \wedge \varphi_2$. Then $T, n \models (\varphi_1 \wedge \varphi_2)^\bullet \Leftrightarrow T, n \models \varphi_1^\bullet$ and $T, n \models \varphi_2^\bullet \Leftrightarrow$ ($n \in N'$ because $\varphi^\bullet$ implies ζ and thus by the inductive hypothesis) $T', n \models \varphi_1$ and $T', n \models \varphi_2 \Leftrightarrow T', n \models \varphi_1 \wedge \varphi_2$.
- $\theta = \langle\downarrow\rangle\varphi$. Then $T, n \models (\langle\downarrow\rangle\varphi)^\bullet \Leftrightarrow T, n \models \zeta$ and $T, n \models \langle\downarrow\rangle\varphi^\bullet \Leftrightarrow$ there exists $n' \in N$ such that nEn' and $T, n' \models \varphi^\bullet \Leftrightarrow$ (by the fact $T, n' \models \zeta$ and, thus, $n' \in N'$ and the inductive hypothesis) there exists $n' \in N'$ such that $nE'n'$ and $T', n' \models \varphi \Leftrightarrow T', n \models \langle\downarrow\rangle\varphi$, as desired.
- $\theta = \langle\theta^+\rangle\tau$. Assume $T, n \models (\langle\theta^+\rangle\tau)^\bullet$. Then, by definition of $(\cdot)^\bullet$, $T, n \models \zeta \wedge \langle\theta^{\bullet+}\rangle\tau^\bullet$. That means there exists n' in T with nE^+n' such that $T, n' \models \tau^\bullet$ and for all n'' with $nE^+n''E^+n'$ it holds $T, n'' \models \theta^\bullet$. By definition of $(\cdot)^\bullet$, the nodes n, n' and all the nodes between n and n' satisfy ζ and thus belong to T'. By inductive hypothesis, $T', n' \models \tau$ and $T', n'' \models \theta$ for all $nE^+n''E^+n'$, which means $T', n \models \langle\theta^+\rangle\tau$ holds. Conversely, assume $T', n \models \langle\theta^+\rangle\tau$. By definition of T', we have that $T, n \models \zeta$, which is the first conjunct of $(\langle\theta^+\rangle\tau)^\bullet$. The second conjunct follows by the inductive hypothesis.

As T is a counterexample for $\varphi^\bullet \subseteq \psi^\circ$, (5.7) implies that T' is a counterexample of $\varphi \subseteq \psi$, as desired.

The same argument applies for multi-labeled trees. The only change is to remove the last disjunction (5.6) from AX. $\square$

5.4.2 Lower bounds

In this section we show that the containment for $\mathsf{TP}^\neg$ is PSPACE hard. This lower bound will carry over to the containment problem for CTP.

Theorem 5.4.1. *(i) The containment problem for* CTP *is* PSPACE*-hard.*

(ii) The containment problem for $\mathsf{TP}^\neg$ *is* PSPACE*-hard,*

(iii) Both results also hold for multi-labeled trees.

Proof. (i) follows from (ii) by Proposition 5.4.4. (iii) follows from (i) and (ii) by Proposition 5.4.3. For proving (ii), we reduce the corridor tiling problem, which is known to be hard for PSPACE (Chlebus, 1986; van Emde Boas, 1997), to the containment problem for $\mathsf{TP}^\neg$. We use the construction from the PSPACE-hardness proof for the containment problem of TP with disjunction over a finite alphabet in (Neven and Schwentick, 2006).

The corridor tiling problem is formalized as follows. Let $\mathsf{Til} = (D, H, V, \bar{b}, \bar{t}, n)$ be a tiling system, where $D = \{d_1, \ldots, d_m\}$ is a finite set of tiles, $H, V \subseteq D^2$ are horizontal and vertical constraints, n is a natural number given in unary notation, $\bar{b}$ and $\bar{t}$ are tuples over D of length n. Intuitively, given a corridor of width n, the goal is to construct a tiling of the corridor using the tiles from D so that the horizontal and vertical constraints are satisfied and the bottom and top rows are tiled by $\bar{b}$ and $\bar{t}$, respectively. We say that a tiling satisfies the horizontal constraints H(respectively the vertical constraints V) if for every $1 \leq i \leq n, j \in \mathbb{N}$ it holds that if d_1 and d_2 are the tiles on the positions (i, j) and $(i+1, j)$ of the corridor (respectively (i, j) and $(i, j+1)$), then $(d_1, d_2) \in H$ ($(d_1, d_2) \in V$).

Now we construct in PTIME in the length of Til, two TP$^\neg$ expressions φ and ψ such that the following holds:

$$\varphi \not\subseteq \psi \text{ iff there exists a tiling for Til.} \tag{5.8}$$

By Proposition 5.4.2, we may without loss of generality construct ψ as a disjunction of TP$^\neg$ expressions. To this purpose, we use the string representation of a tiling. Each row of the considered tiling is represented by the tiles it consists of. If the tiling of a corridor of width n has k rows, it is represented by its rows separated by the special symbol $\sharp$. The top row is followed by the symbol \$. Thus, a tiling is a word of the form $u_1\sharp u_2\sharp \cdots \sharp u_k\$$, where each u_i is the word of length n corresponding to the i-th row in the tiling. In particular $u_1 = \bar{b}$ and $u_k = \bar{t}$. For the sake of readability, for expression r, r^i denotes the path formula $?r; \downarrow; ?r; \ldots; \downarrow; ?r$ with i occurrences of r.

Let φ be

$$\langle ?b_1; \downarrow; ?b_2; \ldots; \downarrow; ?b_n; \downarrow; ?\sharp;\ \downarrow^+; ?t_1, \downarrow; \ldots \downarrow; ?t_n; \downarrow\rangle\$.$$

Intuitively, this expression enforces a tiling to start with a path starting with $\bar{b}$ and finishing with $\bar{t}$ and the final symbol \$. Now the formula ψ defines all incorrect tilings and additional constraints. It is the disjunction of the following TP$^\neg$ formulas.

(0) $\langle \downarrow^+; ?(\neg d_1 \wedge \ldots \wedge \neg d_m \wedge \neg\sharp);\ \downarrow^+\rangle\$$. There is a position that is labeled neither by a tile nor a delimiter.

(1) Incorrect length of a row.

  (1a) $\bigvee_{i=0}^{n-1} \langle \downarrow^+; ?\sharp; \top^i;\ \downarrow\rangle\sharp$, a row is too short;

  (1b) $\langle \downarrow^+; (\neg\sharp)^{n+1}\rangle\top$, a row is too long;

(2) $\bigvee_{(d_1,d_2)\notin H} \langle \downarrow^+; ?d_1; \downarrow; ?d_2\rangle\top$, some horizontal constraints are violated;

(3) $\bigvee_{(d_1,d_2)\notin V} \langle \downarrow^+; ?d_1; \downarrow; \top^n; \downarrow; ?d_2\rangle\top$, some vertical constraints are violated.

Note that negated labels are used in (0) and (1b). Also note that the size of φ and ψ is bounded by a polynomial in the size of Til.

We now show (5.8).

($\Leftarrow$). Assume that there exists a tiling of the corridor. Let s be the string representation of it. Then, $s = u_1\sharp u_2\sharp \ldots \sharp u_k\$$, where $|u_i| = n$, $u_i \in D^n$, $u_1 = \bar{b}$ and $u_k = \bar{t}$. Moreover, on the one hand if $x \cdot y$, is an infix of some u_i, then $(x, y) \in H$, and on the other hand for every infix $x \cdot u' \cdot y$ of length $n+1$ of $u_i\sharp \cdot u_{i+1}$, it holds that $(x, y) \in V$. Let T_s be the corresponding tree, i.e. a single path of $|s|$ nodes $\{v_1, \ldots, v_{|s|}\}$ where the labeling is set in accordance with s, i.e. $l(v_i) = s_i$. Clearly, T_s is a model of φ and not of ψ.

($\Rightarrow$). Let T be a tree such that $T, r \models \varphi$ and $T, r \not\models \psi$. Since $T, r \models \varphi$, there must exist a path $r = v_1, \ldots, v_m$ in T which starts with $\bar{b}$ and finishes with $\bar{t}\$$. Moreover, either $\sharp$ or a symbol from D is the label of every node $v_i, 1 \leq i < m$, according to (0).

We define a tiling function $g : \{0, \ldots, n-1\} \times \mathbb{N} \to D$ assigning a tile to every position in the corridor as follows: $g(i, j) = l(v_{(n+1)\times j+i+1}), 1 \leq i \leq n$, where l is the labeling function of T. Indeed, this function is well defined, as (1) ensures the correct counting. By formulas (2) and (3) the tiling defined by g satisfies the horizontal and vertical constraints. □

The difference between CTP patterns and TP patterns is that CTP descendent edges can be labeled by patterns. One might ask whether a bound on the degree of such a labeling nesting can lead to a lower complexity of the containment problem. Define the until nesting depth un : CTP $\to \mathbb{N}$ as follows.

- $un(p) = un(\top) = 0$,
- $un(\langle\downarrow\rangle\varphi) = un(\langle\downarrow^+\rangle\varphi) = 1$,
- $un(\varphi_1 \wedge \varphi_2) = \max\{un(\varphi_1), un(\varphi_2)\}$,
- $un(\langle\varphi^+\rangle\psi) = un(\varphi) + 1$.

Unfortunately, a close examination of the encoding in the lower bound proof and the encodings in Propositions 5.4.2 and 5.4.4 gives a negative answer to the question. Thus we obtain

Theorem 5.4.2. *The* CTP *containment problem for formulas of until nesting depth one is* PSPACE*-hard.*

5.4.3 Upper bounds

In this section, we show a matching PSPACE upper bound for CTP containment.

The complexity of CTP$^{\neg,\vee}$ containment follows from a translation into existential CTL ($\exists$CTL), whose containment problem is known to be PSPACE-complete (Kupferman and Vardi, 2000). The only small technical issue is that $\exists$CTL formulas are interpreted over *infinite* finitely branching trees. We solve that by relativizing formulas with a new propositional variable s, whose interpretation will provide the desired finite tree.

Theorem 5.4.3. *The containment problem for* CTP$^{\neg,\vee}$ *is decidable in* PSPACE. *This also holds for multi-labeled trees.*

Proof. We prove the upper bound for the case of multi-labeled trees. The upper bound for single-labeled trees then follows by Proposition 5.4.3.

We recall from Kupferman and Vardi (2000) the definition of $\exists$CTL. Let Prop be a set of propositional variables. $\exists$CTL node formulas φ and path formulas α are defined by:

$$\varphi ::= \top \mid \bot \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists\alpha$$
$$\alpha ::= \varphi \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid X\alpha \mid (\alpha U \alpha) \mid (\alpha \hat{U} \alpha)$$

where $p \in \mathsf{Prop}$, and asking that in node formulas, the operators X("next"), U("until") and $\hat{U}$("dual of until") are always immediately preceded by $\exists$.

The semantics for $\exists$CTL is given by *infinite finitely branching trees*. $\exists$CTL path formulas are interpreted at a (possibly infinite) paths of the tree and $\exists$CTL node formulas are interpreted at nodes of the tree. Our reduction will translate $\mathsf{CTP}^{\neg,\vee}$ formulas to $\exists$CTL node formulas, with occurrences of formulas of the form $\exists X\varphi$ and $\exists\psi U\varphi$. Semantics of such formulas is defined as follows, given an infinite tree $T = (N, E, r, \rho)$ and a node $n \in N$.

$$\begin{aligned}
T, n \models \exists X\varphi &\text{ iff there exists a node } m \in N \text{ such that } nEm \text{ and } T, m \models \varphi,\\
T, n \models \exists\psi U\varphi &\text{ iff there exists a node } m \in N \text{ such that } nE^*m \text{ and } T, m \models \varphi,\\
&\text{and for all } n' \in N \text{ with } nE^*n'E^+m \text{ it holds } T, n' \models \psi.
\end{aligned}$$

The CTP node formula $\langle\varphi^+\rangle\psi$ corresponds to the strict until operator, which is expressible in $\exists$CTL as $\exists X\exists\psi U\varphi$.

We denote by $\subseteq_\infty$ the containment relation for $\exists$CTL node formulas over finitely branching trees where each branch is infinite.

Let φ, ψ be in $\mathsf{CTP}^{\neg,\vee}$ and let s be a new propositional variable not in φ, ψ. The translation $(\cdot)^s$ from $\mathsf{CTP}^{\neg,\vee}$ to $\exists$CTL node formulas relativizes every subformula with s and adjusts the syntax.

$$\begin{aligned}
(\top)^s &= s \wedge \top\\
(p)^s &= s \wedge p\\
(\neg p)^s &= s \wedge \neg p\\
(\varphi_1 \wedge \varphi_2)^s &= \varphi_1^s \wedge \varphi_2^s\\
(\varphi_1 \vee \varphi_2)^s &= \varphi_1^s \vee \varphi_2^s\\
(\langle\downarrow\rangle\varphi)^s &= s \wedge \exists X\varphi^s\\
(\langle\psi^+\rangle\varphi)^s &= s \wedge \exists X\exists\psi^s U\varphi^s
\end{aligned}$$

We claim that $\varphi \subseteq \psi$ iff $\varphi^s \subseteq_\infty \psi^s$.

The proof is by contraposition. First assume that for some finite tree $T = (N, E, r, \rho)$ and node $n \in N$ it holds $T, n \models \varphi$ and $T, n \not\models \psi$. We then construct an infinite tree which is a counterexample. Let s be a propositional variable not occurring in φ or ψ. Let T^s_∞ be the infinite tree obtained from T by adding to each leaf in T an infinite path. The labeling is changed as follows: all new nodes have the empty label set and we add the label s to the label set of all old nodes. Formally, $T^s_\infty = (N_\infty, E_\infty, r, \rho_\infty)$, where $N_\infty = N \cup \{n_1^m, n_2^m, \ldots \mid m \text{ is a leaf in } T\}$ is the set of nodes, $E_\infty = E \cup \{(m, n_1^m) \mid n_1^m \in N_\infty\} \cup \{(n_i^m, n_{i+1}^m) \mid n_i^m, n_{i+1}^m \in N_\infty, i \geq 1\}$ is the set of edges, r is the root and ρ_∞ is the labeling function defined as follows.

$$\rho_\infty(n) = \begin{cases} \rho(n) \cup \{s\} & \text{if } n \in N,\\ \emptyset & \text{if } n \in N_\infty \setminus N.\end{cases}$$

We show by induction on the structure of the formula that for every node $n \in N$ and every $\mathsf{CTP}^{\neg,\vee}$ formula θ, $T, n \models \theta$ iff $T^s_\infty, n \models \theta^s$.

To show this we will use the following two facts which are easy consequences of the definitions of T^s_∞ and the translation $(\cdot)^s$:

$$T^s_\infty, n \models s \;\Leftrightarrow\; n \in N, \tag{$*$}$$

and

$$T^s_\infty, n \models \varphi^s \;\Rightarrow\; T^s_\infty, n \models s. \tag{$**$}$$

- $\theta = \top$. We have $T, n \models \top$ iff $n \in N$ iff $n \in N_\infty$ and $s \in \rho_\infty(n)$ iff $T^s_\infty, n \models s \wedge \top$ iff $T^s_\infty, n \models (\top)^s$.
- $\theta = p, p \in \Sigma$. We have $T, n \models p$ iff $p \in \rho(n)$ iff (since $n \in N$) $\{p, s\} \subseteq \rho_\infty(n)$ iff $T^s_\infty, n \models s \wedge p$ iff $T^s_\infty, n \models (p)^s$.
- $\theta = \neg p, p \in \Sigma$. We have $T, n \models \neg p$ iff $p \notin \rho(n)$ iff (since $n \in N$ and, thus, $s \in \rho_\infty(n)$) $p \notin \rho_\infty(n)$ and $s \in \rho_\infty(n)$ iff $T^s_\infty, n \models s \wedge \neg p$ iff $T^s_\infty, n \models (\neg p)^s$.
- $\theta = \varphi_1 \wedge \varphi_2$. We have $T, n \models \varphi_1 \wedge \varphi_2$ iff $T, n \models \varphi_1$ and $T, n \models \varphi_2$ iff (by the induction hypothesis, (*) and (**)) $T^s_\infty, n \models \varphi^s_1$ and $T^s_\infty, n \models \varphi^s_2$ iff $T^s_\infty, n \models \varphi^s_1 \wedge \varphi^s_2$ iff $T^s_\infty, n \models \varphi^s$.
- $\theta = \varphi_1 \vee \varphi_2$. By the same argument as for $\varphi_1 \wedge \varphi_2$.
- $\theta = \langle\downarrow\rangle\varphi$. We have $T, n \models \langle\downarrow\rangle\varphi$ iff there exists $m \in N$ such that nEm and $T, m \models \varphi$ iff (by the induction hypothesis, (*), (**) and the fact that $n \in N$) $T^s_\infty, n \models s$ and there exists $m \in N_\infty$ such that $nE_\infty m$ and $T^s_\infty, m \models \varphi^s$ iff $T^s_\infty, n \models s \wedge \exists X\varphi^s$ iff $T^s_\infty \models (\langle\downarrow\rangle\varphi)^s$.
- $\theta = \langle\psi^+\rangle\varphi$. We have $T, n \models \langle\psi^+\rangle\varphi$ iff there exists $m \in N$ such that $T, m \models \varphi$ and for all $n' \in N$ with $nE^+n'E^+m$ it holds $T, n' \models \psi$. Then by the induction hypothesis, (*) and (**), the latter is equivalent to existence of $m \in N_\infty$ such that $T^s_\infty, m \models \varphi^s$ and for all $n' \in N_\infty$ with $nE^+_\infty n'E^+_\infty m$ it holds $T^s_\infty \models \psi^s$. The latter implication holds because for every node n' with $nE^+_\infty n'E^+_\infty m$ it holds $n' \in N$ and $nE^+n'E^+m$. Equivalently, there exists $n_1 \in N_\infty$ and $m \in N_\infty$ such that $nE_\infty n_1 E^*_\infty m$, $T^s_\infty, m \models \varphi^s$ and for all n' with $n_1E^*_\infty n'E^+_\infty m$ it holds $T^s_\infty, n' \models \psi^s$. This is equivalent to $T^s_\infty, n \models s \wedge \exists X \exists \psi^s U \varphi^s$ as desired.

Thus, we obtain both $T^s_\infty, n \models \varphi^s$ and $T^s_\infty, n \not\models \psi^s$.

For the other direction, let $T_\infty = (N_\infty, E_\infty, r, \rho_\infty)$ be some infinite tree over Σ such that $T^\infty, r \models \varphi^s$ and $T^\infty, r \not\models \psi^s$. We then remove from the model all nodes without s and their descendants. Formally, $T = (N, E, r, \rho)$, where $N = N_\infty \setminus \{n \mid \exists n' \in N_\infty . s \notin \rho_\infty(n') \wedge n'E^*_\infty n\}$ is the set of nodes, $E = E_\infty|_{N\times N}$ is the set of edges, r is the root, and $\rho(n) = \rho_\infty(n)$ is the labeling function.

By induction we can show that for every $\mathsf{CTP}^{\neg,\vee}$ formula θ and node $n \in N$, it holds $T_\infty, n \models \theta^s$ if and only if $T, n \models \theta^s$. The two non-trivial cases are the following.

- $\theta = \langle\downarrow\rangle\varphi$. We have $T_\infty, n \models s \wedge \exists X\varphi^s$ iff $T_\infty, n \models s$ and there exists $m \in N_\infty$ such that $nE_\infty m$ and $T_\infty, m \models \varphi^s$ iff $T, n \models s$ and there exists $m \in N$ such that nEm and $T, m \models \varphi^s$. The direction from right to left is obvious, while the direction from left to right because of the following. Since $T_\infty, m \models \varphi^s$, it follows that $T_\infty, m \models s$, i.e. $s \in \rho_\infty(m)$. Then since $n \in N$, we have that $m \in N$ too by the definition. The latter also implies that nEm holds in T and thus that $T, n \models s \wedge \exists X \varphi^s$.

- $\theta = \langle \psi^+ \rangle \varphi$. We have $T_\infty, n \models s \wedge \exists X \exists \psi^s U \varphi^s$ iff $T_\infty, n \models s$ and there exists $m \in N_\infty$ such that $T_\infty, m \models \varphi^s$, $nE^+_\infty m$ and for all $n' \in N_\infty$ with $nE^+_\infty n' E^+_\infty m$ it holds $T_\infty, n' \models \psi^s$. This is equivalent to $T, n \models s$ and there exists $m \in N$ such that $T, m \models \varphi^s$ and for all n' with $nE^+ n' E^+ m$ it holds $T, m \models \psi^s$, which means $T, n \models s \wedge \exists X \exists \psi^s U \varphi^s$ as desired. The direction from left to write is trivial since T is a substructure of T_∞. The direction from right to left follows from the fact that $m \in N$ and all the nodes n' such that $nE^+_\infty n' E^+_\infty m$ belong to N as well and, moreover, it holds $nE^+ n' E^+ m$.

Thus, we obtain that $T, r \models \varphi^s$ and $T, r \not\models \psi^s$. Because all nodes of T now make s true, we can discard the relativization with s in the formulas. Thus obtaining $T, r \models \varphi$ and $T, r \not\models \psi$. The only problem is that T is infinite. But using the simulations from Theorem 5.3.3 it is easy to see that we can turn T into a finite counterexample.

Thus, we have shown that $\varphi \subseteq \psi$ if and only if $\varphi^s \subseteq_\infty \psi^s$. The latter containment problem in ∃CTL is known to be decidable in PSPACE. This concludes the proof. □

Corollary 5.4.1. *The containment problem for* $\mathsf{TP}^\neg$ *is* PSPACE *complete.*

However, restricting negation to a safe negation $p \wedge \neg q_1 \wedge \ldots \wedge \neg q_n$ keeps the complexity of the containment problem for tree patterns in coNP. Note that $p \wedge \neg q_1 \wedge \ldots \wedge \neg q_n$ only adds expressive power over multi-labeled models, because on single labeled models $p \equiv p \wedge \neg q_1 \wedge \ldots \wedge \neg q_n$. The proof of the following result can be found in Chapter 3 and (Marx and Sherkhonov, 2015). Here $\mathsf{TP}^{\vee,\neg^s}$ is tree patterns expanded with disjunction and the safe negation construct.

Theorem 5.4.4. *The containment problem for* $\mathsf{TP}^{\vee,\neg^s}$ *over multi-labeled trees is in* coNP.

5.5 Conclusion

We have shown that adding conditions on the edges of tree patterns gives a boost in expressive power which comes with the price of a higher – PSPACE – complexity for the containment problem than for tree patterns. We located the source of the extra complexity in the fact that unrestricted negations of labels can be coded in Conditional Tree Patterns. Adding negations of labels to tree patterns causes an increase in complexity of the containment problem from coNP to PSPACE. In Chapter 3 we showed that if negation is restricted to be safe, then containment for PosXPath and CQ stays in coNP and Π^P_2, respectively. In Chapter 4 we show that containment for child-only tree patters with label negation is solvable in PTIME.

This work is a first step in exploring “Regular Tree Patterns” and fragments of it. We mention some directions for future work.

Miklau and Suciu (2004) mention that the existence of a homomorphism between tree patterns is a necessary but not sufficient condition for containment in TP. Can we extend the simulations between conditional tree patterns and trees to simulations between queries, partly capturing containment as for tree patterns?

What is the complexity of containment of regular tree patterns, i.e., the positive fragment of Regular XPath without disjunction and union? As the satisfiability (and thus also the containment) problem for Regular XPath is known to be EXPTIME-complete, it must lie in between PSPACE and EXPTIME.

There are also interesting characterization questions: what is the exact **FO** fragment corresponding to CTP or CTP with disjunction and union? ten Cate (2006) showed that full Regular XPath (with all four axis relations) expanded with path equalities is equally expressive as binary $\mathbf{FO}^*$ (First-order logic extended with a transitive closure operator that can be applied to formulas with exactly two free variables). Is every positive forward binary $\mathbf{FO}^*$ formula expressible as a Regular Tree Pattern? Is CTP with disjunction and union equally expressive as **FO** intersected with positive $\mathbf{FO}^*$? Are unions of CTP equivalent to unions of the first order fragment of conjunctive regular path queries (Calvanese et al., 2000)?

5.A Translations between CTP and *ctp*

The following constructions are similar to the tree and graph representations for tree patterns and conjunctive queries over trees (Björklund et al., 2011; Miklau and Suciu, 2004)

We define the translation function $\mathsf{c}(\cdot)$ which assigns an equivalent conditional tree pattern with one output node to every CTP formula. The output node of $\mathsf{c}(\varphi)$, where φ is a node formula, equals the root.

Let α and φ be path and node CTP formulas. We define $\mathsf{c}(\alpha)$ and $\mathsf{c}(\varphi)$ by mutual induction on the complexity of the path and node formulas. We take $\mathsf{c}(\alpha)$ or $\mathsf{c}(\varphi)$ to be the tree $(N, E, r, o, \rho_N, \rho_E)$, where the components are defined according to the cases.

- $\alpha = \downarrow$. Then N consists of two nodes r and n. The edge relation E is defined as $\{\langle r, n\rangle\}$. Moreover, $o := n$, $\rho_N(v) = \emptyset$ for $v \in \{r, n\}$ and $\rho_E(\langle r, n\rangle) = \downarrow$,
- $\alpha = ?\varphi$. Then $\mathsf{c}(\alpha) := \mathsf{c}(\varphi)$,
- $\alpha = \alpha_1; \alpha_2$. Let $\mathsf{c}(\alpha_1)$ and $\mathsf{c}(\alpha_2)$ be the conditional tree patterns for α_1 and α_2. Then $c(\alpha)$ is the tree obtained as follows. We fuse the root of $\mathsf{c}(\alpha_2)$ with the output node of $\mathsf{c}(\alpha_1)$ and declare the output node of $\mathsf{c}(\alpha_2)$ as the output node of $\mathsf{c}(\alpha)$. The label of the fusion node is the union of the labels of the output node of $\mathsf{c}(\alpha_1)$ and the root of $\mathsf{c}(\alpha_2)$. Labels of other nodes and the edges remain the same as in $\mathsf{c}(\alpha_1)$ and $\mathsf{c}(\alpha_2)$.
- $\alpha = (\downarrow; ?\varphi)^*; \downarrow$. Then N consist of two nodes r and n. The edge relation E is defined as $\{\langle r, n\rangle\}$. Moreover, $o := n$, $\rho_N(v) = \emptyset$ for $v \in \{r, n\}$ and $\rho_E(\langle r, n\rangle) = \mathsf{c}(\varphi)$.

For node formulas, the output node of the translation result is always defined as the root r.

- $\varphi = p$. Then N consists of a single node r, E is empty, the labeling $\rho_N(r) = \{p\}$.
- $\varphi = \top$. Similar to the previous case, with the exception that $\rho_N(r) = \emptyset$
- $\varphi = \varphi_1 \wedge \varphi_2$. Then $\mathsf{c}(\varphi) := \mathsf{c}(\varphi_1) \oplus \mathsf{c}(\varphi_2)$, i.e. the fusion of the conditional tree patterns $\mathsf{c}(\varphi_1)$ and $\mathsf{c}(\varphi_2)$.
- $\varphi = \langle\alpha\rangle\varphi_1$. Let $\mathsf{c}(\alpha)$ and $\mathsf{c}(\varphi_1)$ be the corresponding conditional tree patterns. Then $\mathsf{c}(\varphi)$ is obtained by fusing the output node of $\mathsf{c}(\alpha)$ with the root of $\mathsf{c}(\varphi)$. The labeling of the fusion node is defined as the union of the labels of the root of $\mathsf{c}(\varphi)$ and the output node of $\mathsf{c}(\alpha)$. Labels of the other nodes and edges remain the same as in $\mathsf{c}(\alpha)$ and $\mathsf{c}(\varphi)$.

Translation $f(\cdot)$ works the other way around. Let $t = (N, E, r, o, \rho_N, \rho_E)$ be a conditional tree pattern with one output node. We define $f(t)$ by induction on the nesting depth and the depth of the tree. We first define a mapping φ from nodes $v \in N$ to CTP node formulas. The mapping is defined inductively starting from the leaves as follows. Here, for a finite set $S = \{p_1, \ldots, p_n\}$, by $\bigwedge S$ we denote the finite conjunction $p_1 \wedge \ldots \wedge p_n$. We take $\bigwedge \emptyset$ to be $\top$. For a conditional tree pattern t, by r_t we denote the root of t. For v a node in pattern t:

$$\begin{aligned}\varphi(v) = {} & \wedge\rho_N(v) \wedge \\ & \bigwedge_{\langle v,v'\rangle \in E,\ \rho_E(v,v')=t'} \langle ?\varphi(r_{t'})\rangle\varphi(v') \wedge \\ & \bigwedge_{\langle v,v'\rangle \in E,\ \rho_E(v,v')=\downarrow} \langle\downarrow\rangle\varphi(v')\end{aligned}$$

Now let $r = v_1, \ldots, v_n = o$ be the path from the root r to the output node o in t. Let the expression $d_i, 1 \leq i \leq n-1$ be defined by: $d_i = \downarrow$ if $\rho_E(v_i, v_{i+1}) = \downarrow$ and $d_i = (\downarrow; ?\varphi(r_{t'}))^*; \downarrow$ if $\rho_E(v_i, v_{i+1}) = t'$. Then the result of the translation $f(t)$ of the conditional tree pattern t is the CTP path formula:

$$?\varphi(v_1); d_1; ?\varphi(v_2); d_2; \ldots; ?\varphi(v_{n-1}); d_{n-1}; ?\varphi(v_n).$$

For the next proposition we need the definition of equivalence between conditional tree patterns. Let t be a conditional tree pattern with output nodes $\bar{o}$,$|\bar{o}|$=k, and T a tree. Then the *answer set* of t over T is the set $Out(t, T) = \{\langle g(o_1), \ldots, g(o_k)\rangle \mid g \text{ is a simulation of } t \text{ in } T\}$. We say that two conditional tree patterns t_1 and t_2 are *equivalent*, denoted as $t_1 \simeq t_2$, if $Out(t_1, T) = Out(t_2, T)$ for every tree T.

Proposition 5.A.1. *Let φ and α be* CTP *node and path formulas, t a conditional tree pattern. Then it holds that*

(i) $f(\mathsf{c}(\varphi)) \equiv \varphi$ *and* $f(\mathsf{c}(\alpha)) \equiv \alpha$,

(ii) $\mathsf{c}(f(t)) \simeq t$.

The proof of (i) is by mutual induction on α and φ, and the proof of (ii) is by induction on nesting depth and depth of t.

Part II

Application: Why-not Explanations

6 High-Level Why-Not Explanations using Ontologies

In this chapter we propose a foundational framework for *why-not explanations*, that is, explanations for why a tuple is missing from a query result. Our why-not explanations leverage concepts from an ontology to provide high-level and meaningful reasons for why a tuple is missing from the result of a query.

A key algorithmic problem in our framework is that of *computing a most-general explanation* for a why-not question, relative to an ontology, which can either be provided by the user, or it may be automatically derived from the data and/or schema. We study the complexity of this problem and associated problems, and present concrete algorithms for computing why-not explanations, thus addressing **RQ 4**. In the case where an external ontology is provided, we first show that the problem of deciding the existence of an explanation to a why-not question is NP-complete in general. However, the problem is solvable in polynomial time for queries of bounded arity, provided that the ontology is specified in a suitable language, such as a member of the DL-Lite family of description logics, which allows for efficient concept subsumption checking. Furthermore, we show that a most-general explanation can be computed in polynomial time in this case. In addition, we address **RQ 3** by proposing a method for deriving a suitable (virtual) ontology from a database and/or a schema. We will show that this is the same as solving the containment problem. Thus, the framework proposed here can be seen as a use case for the containment problem.

We also present an algorithm for computing a most-general explanation to a why-not question, relative to such derived ontologies. This algorithm runs in polynomial-time in the case when concepts are defined in a selection-free language or if the underlying schema is fixed. Finally, we also study the problem of computing *short* most-general explanations, and we briefly discuss alternative definitions of what it means to be an explanation, and to be most general.

6.1 Introduction and results

An increasing number of databases are derived, extracted, or curated from disparate data sources. Consequently, it becomes more and more important to provide data consumers with mechanisms that will allow them to gain an understanding of the data that they are confronted with. An essential functionality towards this goal is the capability to provide meaningful explanations about why data is present or missing from the result of a query. Explanations help data consumers gauge how much trust one can place on the result. Perhaps more importantly, they provide useful information for debugging the query or data that led to incorrect results.

This is particularly the case in scenarios where complex data analysis tasks are specified through large collections of nested views (i.e., views that may be defined in terms of other views). For example, schemas with nested view definitions and integrity constraints capture the core of LogiQL (Green, 2015; Green et al., 2012; Halpin and Rugaber, 2014) (where view definitions may, in general, involve not only relational operations, but also aggregation, machine learning and mathematical optimization tasks). LogiQL is a language developed and used at LogicBlox (Aref et al., 2015) for developing data-intensive "self service" applications involving complex data analytics workflows. Similar recent industrial systems include Datomic[1] and Google's Yedalog (Chin et al., 2015). In each of these systems, nested view definitions (or, Datalog programs) are used to specify complex workflows to drive data-analytics tasks. Explanations for unexpected query results (such as an unexpected tuple or a missing tuple) are very useful in such settings, since the source of an error can be particularly hard to track.

There has been considerable research on the topic of deriving explanations for why a tuple belongs to the output of a query. Early systems were developed in (Arora et al., 1993; Shmueli and Tsur, 1990) to provide explanations for answers to logic programs in the context of a deductive database. The presence of a tuple in the output is explained by enumerating all possible derivations, that is, instantiations of the logic rules that derive the answer tuple. In (Shmueli and Tsur, 1990), the system also explains missing answers, by providing a partially instantiated rule, based on the missing tuple, and leaving the user to figure out how the rest of the rule would have to be instantiated. In the last decade or so, there have been significant efforts to characterize different notions of provenance (or lineage) of query answers (see, e.g., (Cheney et al., 2009; Green et al., 2007b)) which can also be applied to understand why an answer is in the query result. More details on this can be found in Chapter 2.

There have also been extensive studies on the *why-not problem* (e.g., more recent studies include (Baid et al., 2015; Chapman and Jagadish, 2009; Herschel et al., 2009; Huang et al., 2008a; Meliou et al., 2010; Tran and Chan, 2010)). The why-not problem is the problem of explaining why an answer is missing from the output. Since (Shmueli and Tsur, 1990), the *why-not problem* was also studied in (Herschel et al., 2009; Huang et al., 2008a) in the context of debugging results of data extracted via select-project-join queries, and, subsequently, a larger class of queries that also includes union and aggregation operators. Unlike (Shmueli and Tsur, 1990) which is geared towards providing explanations for answers and missing answers, the goal in (Huang et al., 2008a) is to propose modifications to underlying database I, yielding another database I' based on the provenance of the missing tuple, constraints, and trust specification at hand, so that the missing tuple appears in the result of the same query q over the updated database I'. In contrast to the *data-centric* approach of updating the database to derive the missing answer, another line of research (Bidoit et al., 2014a; Chapman and Jagadish, 2009; Tran and Chan, 2010) follows a *query-centric* approach whereby the query q at hand is

[1]`www.datomic.com`

modified to q' (without modifying the underlying database) so that the missing answer appears in the output of $q'(I)$. More details can be found in Chapter 2.

A new take on why-not questions

In this chapter, we develop a novel foundational framework for why-not explanations that is principally different from prior approaches. Our approach is neither data-centric nor query-centric. Instead, we derive high-level explanations via an ontology that is either provided, or is derived from the data or schema. Our immediate goal is not to compute repairs of the underlying database or query so that the missing answer would appear in the result. Rather, as in (Shmueli and Tsur, 1990), our primary goal is to provide understandable explanations for why an answer is missing from the query result. As we will illustrate, explanations that are based on an ontology have the potential to be high-level and provide meaningful insight to why a tuple is missing from the result. This is because an ontology abstracts a domain in terms of concepts and relationships amongst concepts. Hence, explanations that are based on concepts and relationships from an ontology will embody such high-level abstractions. As we shall describe, our work considers two cases. The first is when an ontology is provided externally, in which case explanations will embody external knowledge about the domain. The second is when an ontology is not provided. For the latter, we allow an ontology to be derived from the schema, and hence explanations will embody knowledge about the domain through concepts and relationships that are defined over the schema.

Formally, an explanation for why a tuple $\overline{a}$ is not among the results of a query $q(I)$, in our framework, is a tuple of concepts from the ontology whose extension includes the missing tuple $\overline{a}$ and, at the same time, does not include any tuples from $q(I)$. For example, a query may ask for all products that each store has in stock, in the form of (product ID, store ID) pairs, from the database of a large retail company. A user may then ask why is the pair *(P0034, S012)* not among the result of the query. Suppose *P0034* refers to a bluetooth headset product and *S012* refers to a particular store in San Francisco. If *P0034* is an instance of a concept *bluetooth headsets* and *S012* is an instance of a concept *stores in San Francisco*, and suppose that no pair (x, y), where x is an instance of *bluetooth headset* and y is an instance of *stores in San Francisco*, belongs to the query result. Then the pair of concepts (*bluetooth headset*, *stores in San Francisco*) is an explanation for the given why-not question. Intuitively, it signifies the fact that "*none of the stores in San Francisco has any bluetooth headsets on stock*".

There may be multiple explanations for a given why-not question. In the above example, this would be the case if, for instance, *S012* belongs also to a more general concept *stores in California*, and that none of the stores in California have bluetooth headsets on stock. Our goal is to compute a *most-general explanation*, that is, an explanation that is not strictly subsumed by any other explanation. We study the complexity of computing a most-general explanation to a why-not question. Formally, we define a *why-not instance* (or, *why-not question*) to be a quintuple $(\mathbf{S}, I, q, \mathit{Ans}, \overline{a})$ where $\mathbf{S}$ is a *schema*, which may include integrity constraints; I is an instance of $\mathbf{S}$; q is a query over $\mathbf{S}$; $\mathit{Ans} = q(I)$; and $\overline{a} \notin q(I)$.

As mentioned earlier, a particular scenario where why-not questions easily arise is when querying schemas that include a large collection of views, and where each view may be nested, that is, defined in terms of other views. Our framework captures this setting, since view definitions can be expressed by means of constraints.

Our framework supports a very general notion of an ontology, which we call $\mathbf{S}$-*ontologies*. For a given relational schema $\mathbf{S}$, an $\mathbf{S}$-ontology is a triple $(\mathcal{C}, \sqsubseteq, ext)$ that defines the set of concepts, the subsumption relationship between concepts, and respectively, the extension of each concept w.r.t. an instance of the schema $\mathbf{S}$. We use this general notion of an $\mathbf{S}$-ontology to formalize the key notions of *explanation* and *most-general explanation*, and we show that $\mathbf{S}$-ontologies capture two different types of ontologies.

The first type of ontologies we consider are those that are defined externally, provided that there is a way to associate the concepts in the externally defined ontology to the instance at hand. For example, the ontology may be represented in the form of an Ontology-Based Data Access (OBDA) specification (Poggi et al., 2008). More precisely, an OBDA specification consists of a set of concepts and subsumption relation specified by means of a description logic terminology, and a set of *mapping assertions* that relates the concepts to a relational database schema at hand. Every OBDA specification induces a corresponding $\mathbf{S}$-ontology. If the concepts and subsumption relation are defined by a TBox in a tractable description logic such as *DL-Lite*$_{\mathcal{R}}$, and the mapping assertions are Global-As-View (GAV) assertions, the induced $\mathbf{S}$-ontology can in fact be computed from the OBDA specification in polynomial time. We present an algorithm for computing most-general explanations to a why-not question, given an external $\mathbf{S}$-ontology. The algorithm runs in polynomial time when the arity of the query is bounded, and it executes in exponential time in general. We show that the exponential running time is unavoidable, unless P=NP, because the problem of deciding whether or not there exists an explanation to a why-not question given an external $\mathbf{S}$-ontology is NP-complete in general.

The second type of ontologies that we consider are ontologies that are derived either (a) from a schema $\mathbf{S}$, or (b) from an instance of the schema. In both cases, the concepts of the ontology are defined through concept expressions in a suitable language $L_{\mathbf{S}}$ that we develop. Specifically, our concepts are obtained from the relations in the schema, through selections, projections, and intersections. The difference between the two cases (a) and (b) lies in the way the subsumption relation $\sqsubseteq$ is defined. In the former, a concept C is considered to be subsumed by another concept C' if the extension of C is contained in the extension of C' over all instances of the schema. For the latter, subsumption is considered to hold if the extension of C is contained in the extension of C' with respect to the given instance of the schema. The $\mathbf{S}$-ontology induced by a schema $\mathbf{S}$, or instance I, denoted $\mathcal{O}_{\mathbf{S}}$ or $\mathcal{O}_I$, respectively, is typically infinite, and is not intended to be materialized. Instead, we present an algorithm for directly computing a most-general explanation with respect to $\mathcal{O}_I$. The algorithm runs in exponential time in general. However, if the schema is of bounded arity, the algorithm runs in polynomial time. As for computing most-general explanations with respect to $\mathcal{O}_{\mathbf{S}}$, we identify restrictions on the integrity constraints under which the problem is decidable, and we present complexity upper bounds for these cases.

More related work

The use of ontologies to facilitate access to databases is not new. A prominent example is OBDA, where queries are either posed directly against an ontology, or an ontology is used to enrich a data schema against which queries are posed with additional relations (namely, the concepts from the ontology) (Bienvenu et al., 2013; Poggi et al., 2008). Answers are computed based on an open-world assumption and using the mapping assertions and ontology provided by the OBDA specification. As we described above, we make use of OBDA specifications as a means to specify an external ontology and with a database instance through mapping assertions. However, unlike in OBDA, we consider queries posed against a database instance under the traditional closed-world semantics, and the ontology is used only to derive why-not explanations.

The problems of providing why explanations and why-not explanations have also been investigated in the context of OBDA in (Borgida et al., 2008) and (Calvanese et al., 2013), respectively. The why-not explanations of (Calvanese et al., 2013) follow the *data-centric* approach to why-not provenance as we discussed earlier where their goal is to modify the assertions that describe the extensions of concepts in the ontology so that the missing tuple will appear in the query result.

There has also been prior work on extracting ontologies from data. For example, in (Lubyte and Tessaris, 2009), the authors considered heuristics to automatically generate an ontology from a relational database by defining project-join queries over the data. Other examples on ontology extraction from data include publishing relational data as RDF graphs or statements (e.g., D2RQ (Bizer and Seaborne, 2004), Triplify (Auer et al., 2009)). We emphasize that our goal is not to extract and materialize ontologies, but rather, to use an ontology that is derived from data to compute why-not explanations.

Organization

After the preliminaries, in Section 6.3 we present our framework for why-not explanations. In Section 6.4 we discuss in detail the two ways of obtaining an $\mathbf{S}$-ontology. In Section 6.5 we present our main algorithmic results. Finally, in Section 6.6, we study variatations of our framework, including the problem of producing *short* most-general explanations, and alternative notions of *explanation*, and of what it means to be *most general*.

6.2 Preliminaries

A *schema* is a pair $(\mathbf{S}, \Sigma)$, where $\mathbf{S}$ is a set $\{R_1, \ldots, R_n\}$ of relation names, where each relation name has an associated arity, and Σ is a set of first-order sentences over $\mathbf{S}$, which we will refer to as *integrity constraints*. Abusing the notation, we will write $\mathbf{S}$ for the schema $(\mathbf{S}, \Sigma)$. A *fact* is an expression of the form $R(b_1, \ldots, b_k)$, where $R \in \mathbf{S}$ is a relation of arity k, and for $1 \leq i \leq k$, we have $b_i \in \mathbf{Const}$, where $\mathbf{Const}$ is a countably infinite set of constants. We assume a dense linear order $<$ on $\mathbf{Const}$. An *attribute* A of an k-ary relation name $R \in \mathbf{S}$ is a number i such that $1 \leq i \leq k$. For a fact $R(\overline{b})$ where $\overline{b} = b_1, \ldots, b_k$, we sometimes write $\pi_{A_1,\ldots,A_k}(\overline{b})$ to mean the tuple $(b_{A_1}, \ldots, b_{A_k})$. An *atom over* $\mathbf{S}$ is an expression $R(x_1, \ldots, x_n)$, where $R \in \mathbf{S}$ and every $x_i, i \in \{1, \ldots, n\}$ is a variable or a constant.

A *database instance*, or simply an *instance*, I over $\mathbf{S}$ is a set of facts over $\mathbf{S}$ satisfying the integrity constraints Σ. Equivalently, an instance I is a map that assigns to each k-ary relation name $R \in \mathbf{S}$ a finite set of k-tuples over $\mathbf{Const}$ such that the integrity constraints are satisfied. By R^I we denote the set of these tuples. We write $\text{Inst}(\mathbf{S})$ to denote the set of all database instances over $\mathbf{S}$, and $\text{adom}(I)$ to denote the active domain of I, i.e., the set of all constants occurring in facts of I.

Queries A *conjunctive query* (CQ) over $\mathbf{S}$ is a query of the form $\exists \overline{y}.\varphi(\overline{x}, \overline{y})$ where φ is a conjunction of atoms over $\mathbf{S}$. Given an instance I and a CQ q, we write $q(I)$ to denote the set of answers of q over I. We allow conjunctive queries containing comparisons to constants, that is, comparisons of the form $x \,\mathsf{op}\, c$, where $\mathsf{op} \in \{=, <, >, \leq, \geq\}$ and $c \in \mathbf{Const}$. We show that all upper bounds hold for the case of CQs with such comparisons, and all lower bounds hold without the use of comparisons (unless explicitly specified otherwise). We do *not* allow comparisons between variables.

Integrity constraints We consider different classes of integrity constraints, including functional dependencies and inclusion dependencies. We also consider UCQ-view definitions and nested UCQ-view definitions, which can be expressed using integrity constraints as well.

A *functional dependency* (FD) on a relation $R \in \mathbf{S}$ is an expression of the form $R: X \to Y$ where X and Y are subsets of the set of attributes of R. We say that an instance I over $\mathbf{S}$ satisfies the FD if for every $\overline{a}_1$ and $\overline{a}_2$ from R^I if $\pi_A(\overline{a}_1) = \pi_A(\overline{a}_2)$ for every $A \in X$, then $\pi_B(\overline{a}_1) = \pi_B(\overline{a}_2)$ for every $B \in Y$.

An *inclusion dependency* (ID) is an expression of the form

$$R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n],$$

where $R, S \in \mathbf{S}$, each A_i and B_j is an attribute of R and S respectively. We say that an instance I over $\mathbf{S}$ satisfies the ID if

$$\{\pi_{A_1,\ldots,A_n}(\overline{a}) \mid \overline{a} \in R^I\} \subseteq \{\pi_{B_1,\ldots,B_n}(\overline{b}) \mid \overline{b} \in S^I\}.$$

Note that functional and integrity constraints can equivalently be written as first-order sentences (Abiteboul et al., 1995).

View Definitions To simplify our presentation, we treat view defintions as a special case of integrity constraints.

A set of integrity constraints Σ over $\mathbf{S}$ is said to be a *collection of* UCQ-*view definitions* if there exists a partition $\mathbf{S} = \mathbf{D} \cup \mathbf{V}$ such that for every $P \in \mathbf{V}$, Σ contains exactly one first-order sentence of the form:

$$P(\bar{x}) \leftrightarrow \bigvee_{i=1}^{k} \varphi_i(\bar{x}), \qquad (*)$$

where each φ_i is a conjunctive query (with comparisons to constants) over $\mathbf{D}$.

Similarly, a set of integrity constraints Σ over $\mathbf{S}$ is said to be a *collection of nested* UCQ-*view definitions* if there exists a partition $\mathbf{S} = \mathbf{D} \cup \mathbf{V}$ such that for every $P \in \mathbf{V}$, Σ

Data schema $\mathbf{D}$:

{Cities(name, population, country, continent),
Train-Connections(city_from, city_to)}

View schema $\mathbf{V}$:

{BigCity(name), EuropeanCountry(name),
Reachable(city_from, city_to)}

UCQ-view definitions:

BigCity(x)	$\leftrightarrow$	Cities(x,y,z,w) $\wedge$ $y \geq 5000000$
EuropeanCountry(z)	$\leftrightarrow$	Cities(x,y,z,w) $\wedge$ $w =$ Europe
Reachable(x,y)	$\leftrightarrow$	Train-Connections(x,y) $\vee$
		(Train-Connections(x,z) $\wedge$ Train-Connections (z,y))

Functional and inclusion dependencies:

country	$\rightarrow$	continent
BigCity[name]	$\subseteq$	Train-Connections[city_from]
Train-Connections[city_from]	$\subseteq$	Cities[name]
Train-Connections[city_to]	$\subseteq$	Cities[name]

Figure 6.1: Example of a schema $\mathbf{S}$.

contains exactly one first-order sentence of the form (*), where each φ_i is now allowed to be a conjunctive query over $\mathbf{D} \cup \mathbf{V}$, but subject to the following acyclicity condition. Let us say that $P \in \mathbf{V}$ *depends on* $R \in \mathbf{V}$, if R occurs in the view definition of P, that is, in the sentence of Σ that is of the form (*) for P. We require that the "depends on" relation is acyclic. If, in the view definition of every $P \in \mathbf{V}$, each disjunct φ_i contains at most one atom over $\mathbf{V}$, then we say that Σ is a collection of *linearly* nested UCQ-view definitions.

Note that a collection of nested UCQ-view definitions (in the absence of comparisons) can be equivalently viewed as a non-recursive Datalog program and vice versa (Benedikt and Gottlob, 2010). In particular, a collection of linearly nested UCQ-view definitions corresponds to a linear non-recursive Datalog program.

Example 6.2.1. As an example of a schema, consider $\mathbf{S} = \mathbf{D} \cup \mathbf{V}$ with the integrity constraints in Figure 6.1. An instance I of the schema $\mathbf{S}$ is given in Figure 6.2. $\square$

6.3 Why-not explanations

Next, we introduce our ontology-based framework for explaining why a tuple is not in the output of a query. Our framework is based on a general notion of an ontology. As we shall describe in Section 6.4, the ontology that is used may be an external ontology (for example, an existing ontology specified in a description logic), or it may be an ontology that is derived from a schema. Both are a special case of our general definition of an $\mathbf{S}$-ontology.

Definition 6.3.1 ($\mathbf{S}$-ontology)**.** *An* $\mathbf{S}$-ontology *over a relational schema* $\mathbf{S}$ *is a triple* $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$*, where*

- $\mathcal{C}$ *is a possibly infinite set, whose elements are called* concepts*,*

Cities

name	population	country	continent
Amsterdam	779,808	Netherlands	Europe
Berlin	3,502,000	Germany	Europe
Rome	2,753,000	Italy	Europe
New York	8,337,000	USA	N.America
San Francisco	837,442	USA	N.America
Santa Cruz	59,946	USA	N.America
Tokyo	13,185, 000	Japan	Asia
Kyoto	1,400,000	Japan	Asia

Train-Connections

city_from	city_to
Amsterdam	Berlin
Berlin	Rome
Berlin	Amsterdam
New York	San Francisco
San Francisco	Santa Cruz
Tokyo	Kyoto

BigCity

name
New York
Tokyo

EuropeanCountry

name
Netherlands
Germany
Italy

Reachable

city_from	city_to
Amsterdam	Berlin
Berlin	Rome
Berlin	Amsterdam
New York	San Francisco
San Francisco	Santa Cruz
Tokyo	Kyoto
Amsterdam	Rome
Amsterdam	Amsterdam
Berlin	Berlin
New York	Santa Cruz

Figure 6.2: Example of an instance I of $\mathbf{S}$.

- $\sqsubseteq$ *is a pre-order (i.e., a reflexive and transitive binary relation) on $\mathcal{C}$, called the* subsumption relation, *and*
- $ext : \mathcal{C} \times \mathrm{Inst}(\mathbf{S}) \to \wp(\mathbf{Const})$ *is a polynomial-time computable function that will be used to identify instances of a concept in a given database instance ($\wp(\mathbf{Const})$ denotes the powerset of $\mathbf{Const}$).*

More precisely, we assume that ext is specified by a Turing machine that, given $C \in \mathcal{C}$, $I \in \mathrm{Inst}(\mathbf{S})$ and $c \in \mathbf{Const}$, decides in polynomial time if $c \in ext(C, I)$.

A database instance $I \in \mathrm{Inst}(\mathbf{S})$ is consistent *with $\mathcal{O}$ if, for all $C_1, C_2 \in \mathcal{C}$ with $C_1 \sqsubseteq C_2$, we have $ext(C_1, I) \subseteq ext(C_2, I)$.*

An example of an $\mathbf{S}$-ontology $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ is shown in Figure 6.3, where the concept subsumption relation $\sqsubseteq$ is depicted by means of a Hasse diagram. Note that, in this example, $ext(C, I)$ is independent of the database instance I (and, as a consequence, every $\mathbf{S}$-instance is consistent with $\mathcal{O}$). In general, this is not the case (for example, the extension of a concept may be determined through mapping assertions, cf. Section 6.4.1).

We define our notion of an ontology-based explanation next.

Definition 6.3.2 (Explanation). *Let $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ be an $\mathbf{S}$-ontology, I an $\mathbf{S}$-instance consistent with $\mathcal{O}$. Let q be an m-ary query over $\mathbf{S}$, and $\overline{a} = \langle a_1, \ldots, a_m \rangle$ a tuple of constants such that $\overline{a} \notin q(I)$. Then a tuple of concepts $(C_1, \ldots, C_m)$ from $\mathcal{C}^m$ is called an* explanation *for $\overline{a} \notin q(I)$ with respect to $\mathcal{O}$ (or an* explanation *in short) if:*

- *for every $1 \leq i \leq m$, $a_i \in ext(C_i, I)$, and*
- $(ext(C_1, I) \times \ldots \times ext(C_m, I)) \cap q(I) = \emptyset$.

In other words, an explanation is a tuple of concepts whose extension includes the missing tuple $\overline{a}$ (and thus explains $\overline{a}$) but, at the same time, it does not include any tuple in $q(I)$ (and thus does not explain any tuple in $q(I)$). Intuitively, the tuple of concepts is an explanation that is orthogonal to existing tuples in $q(I)$ but relevant for the missing tuple $\overline{a}$, and thus forms an explanation for why $\overline{a}$ is not in $q(I)$. There can be multiple explanations in general and the "best" explanations are the ones that are the most general.

Definition 6.3.3 (Most-general explanation)**.** *Let $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ be an* **S***-ontology, and let $E = (C_1, \ldots, C_m)$ and $E' = (C'_1, \ldots, C'_m)$ be two tuples of concepts from $\mathcal{C}^m$.*

- *We say that E is* less general *than E' with respect to $\mathcal{O}$, denoted as $E \leq_{\mathcal{O}} E'$, if $C_i \sqsubseteq C'_i$ for every $i, 1 \leq i \leq m$.*
- *We say that E is* strictly less general *than E' with respect to $\mathcal{O}$, denoted as $E <_{\mathcal{O}} E'$, if $E \leq_{\mathcal{O}} E'$, and $E' \not\leq_{\mathcal{O}} E$.*
- *We say that E is a* most-general explanation *for $\overline{a} \notin q(I)$ if E is an explanation for $\overline{a} \notin q(I)$, and there is no explanation E' for $\overline{a} \notin q(I)$ such that $E' >_{\mathcal{O}} E$.*

As we will formally define in Section 6.5, a *why-not problem* asks the question: "why is the tuple $\langle a_1, \ldots, a_m \rangle$ not in the output of a query q over an instance I of schema **S**?" The following example illustrates the notions of explanations and most-general explanations in the context of a why-not problem.

Example 6.3.1. Consider the instance $I_{\mathbf{D}}$ of the relational schema **S** = {Cities(name, population, country, continent), Train-Connections(city_from, city_to)} shown in Figure 6.2.

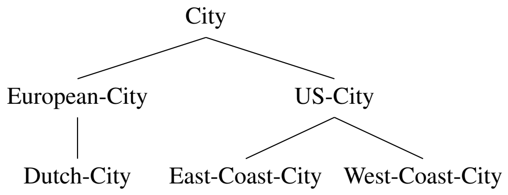
Suppose q is the query $\exists z.$ Train-Connections(x, z) $\wedge$ Train-Connections(z, y). That is, the query asks for all pairs of cities that are connected via a city. Then $q(I)$ returns tuples

$$\{\langle \text{Amsterdam, Rome} \rangle, \langle \text{Amsterdam, Amsterdam} \rangle,$$
$$\langle \text{Berlin, Berlin} \rangle, \langle \text{New York, Santa Cruz} \rangle\}.$$

A user may ask why is the tuple $\langle$Amsterdam, New York$\rangle$ not in the result of $q(I)$ (i.e., why is $\langle$Amsterdam, New York$\rangle \notin q(I)$?). Based on the **S**-ontology defined in Figure 6.3, we can derive the following explanations for $\langle$Amsterdam, New York$\rangle \notin q(I)$:

$$\begin{aligned} E_1 &= \langle \text{Dutch-City, East-Coast-City} \rangle \\ E_2 &= \langle \text{Dutch-City, US-City} \rangle \\ E_3 &= \langle \text{European-City, East-Coast-City} \rangle \\ E_4 &= \langle \text{European-City, US-City} \rangle \end{aligned}$$

E_1 is the simplest explanation, i.e., the one we can build by looking at the lower level of the hierarchy in our **S**-ontology. Each subsequent explanation is more general than at least one of the prior explanations w.r.t. the **S**-ontology. In particular, we have $E_4 >_{\mathcal{O}} E_2 >_{\mathcal{O}} E_1$, and $E_4 >_{\mathcal{O}} E_3 >_{\mathcal{O}} E_1$. Thus, the most-general explanation for why $\langle$Amsterdam, New York$\rangle \notin q(I)$ w.r.t. our **S**-ontology is E_4, which intuitively informs that the reason is because Amsterdam is a city in Europe while New York is a city in the US (and hence, they are not connected by train). Note that all the other possible combinations of concepts are not explanations because they intersect with $q(I)$. □



$$\begin{aligned}
ext(\text{City}, I) &= \{\text{Amsterdam, Berlin, Rome, New York,} \\
&\quad\ \text{San Francisco, Santa Cruz, Tokyo, Kyoto}\} \\
ext(\text{European-City}, I) &= \{\text{Amsterdam, Berlin, Rome}\} \\
ext(\text{Dutch-City}, I) &= \{\text{Amsterdam}\} \\
ext(\text{US-City}, I) &= \{\text{New York, San Francisco, Santa Cruz}\} \\
ext(\text{East-Coast-City}, I) &= \{\text{New York}\} \\
ext(\text{West-Coast-City}, I) &= \{\text{Santa Cruz, San Francisco}\}
\end{aligned}$$

Figure 6.3: Example ontology.

As we will see in Example 6.4.3, there may be more than one most-general explanations in general.

Generalizing the above example, we can informally define the problem of explaining why-not questions via ontologies as follows: *given an instance I of schema* $\mathbf{S}$*, a query q over* $\mathbf{S}$*, an* $\mathbf{S}$*-ontology $\mathcal{O}$ (consistent with I) and a tuple $\overline{a} \notin q(I)$, compute a most-general explanation for $\overline{a} \notin q(I)$, if one exists, w.r.t. $\mathcal{O}$.* As we shall describe in Section 6.5, in addition to the above problem of computing one most-general explanation, we will also investigate the corresponding decision problem that asks whether or not an explanation for a why-not problem exists, and whether or not a given tuple of concepts is a most-general explanation for a why-not problem. In our framework, the $\mathbf{S}$-ontology $\mathcal{O}$ may be given explicitly as part of the input, or it may be derived from a given database instance or a given schema. We will introduce the different scenarios by which an ontology may be obtained in the next section, before we describe our algorithms for computing most-general explanations in Section 6.5.

6.4 Obtaining ontologies

In this section we discuss two approaches by which $\mathbf{S}$-ontologies may be obtained. The first approach allows one to leverage an external ontology, provided that there is a way to relate a concept in the ontology to a database instance. In this case, the set $\mathcal{C}$ of concepts is specified through a description logic such as $\mathcal{ALC}$ or *DL-Lite*; $\sqsubseteq$ is a partial order on the concepts defined in the ontology, and the function *ext* may be given through *mapping assertions*. The second approach considers an $\mathbf{S}$-ontology that is derived from a specific database instance, or from a schema. This approach is useful as it allows one to define an ontology to be used for explaining why-not questions in the absence of an external ontology.

In either case, we study the complexity of deriving such $\mathbf{S}$-ontologies based on the language on which concepts are defined, the subsumption between concepts, and the function ext, which is defined according to the semantics of the concept language.

6.4.1 Leveraging an external ontology

We first consider the case where we are given an external ontology that models the domain of the database instance, and a relationship between the ontology and the instance. We will illustrate in particular how *description logic ontologies* are captured as a special case of our framework.

In what follows, our exposition borrows notions from the Ontology-Based Data Access (OBDA) framework. Specifically, we will make crucial use of the notion of an *OBDA specification* (Di Pinto et al., 2013), which consists of a description logic ontology, a relational schema, and a collection of mapping assertions. To keep the exposition simple, we restrict our discussion to one particular description logic, called *DL-Lite*$_\mathcal{R}$, which is a representative member of the *DL-Lite* family of description logics (Calvanese et al., 2007). *DL-Lite*$_\mathcal{R}$ is the basis for the OWL 2 QL[2] profile of OWL 2, which is a standard ontology language for Semantic Web adopted by W3C. As the other languages in the *DL-Lite* family, *DL-Lite*$_\mathcal{R}$ exhibits a good trade off between expressivity and complexity bounds for important reasoning tasks such as subsumption checking, instance checking and query answering.

TBox and Mapping Assertions. In the description logic literature, an ontology is typically formalized as a TBox (Terminology Box), which consists of finitely many *TBox axioms*, where each TBox axiom expresses a relationship between concepts. Alongside TBoxes, ABoxes (Assertion Boxes) are sometimes used to describe the extension of concepts. To simplify the presentation, we do not consider ABoxes here.

Definition 6.4.1 (*DL-Lite*$_\mathcal{R}$)**.** *Fix a finite set Φ_C of* "atomic concepts" *and a finite set Φ_R of* "atomic roles".

- *The* concept expressions *and* role expressions *of DL-Lite*$_\mathcal{R}$ *are defined as follows:*

$$\begin{array}{ll}
\textit{Basic concept expression:} & B ::= A \mid \exists R \\
\textit{Basic role expression:} & R ::= P \mid P^- \\
\textit{Concept expressions:} & C ::= B \mid \neg B \\
\textit{Role expressions} & E ::= R \mid \neg R
\end{array}$$

  *where $A \in \Phi_C$ and $P \in \Phi_R$. Formally, a (Φ_C, Φ_R)-*interpretation *$\mathcal{I}$ is a map that assigns to every atomic concept in Φ_C a unary relation over $\mathbf{Const}$ and to every atomic role in Φ_R a binary relation over $\mathbf{Const}$. The map $\mathcal{I}$ naturally extends to arbitrary concept expressions and role expressions:*

$$\begin{array}{ll}
\mathcal{I}(P^-) = \{(x,y) \mid (y,x) \in \mathcal{I}(P)\} & \mathcal{I}(\exists P) = \pi_1(\mathcal{I}(P)) \\
\mathcal{I}(\neg P) = \mathbf{Const}^2 \setminus \mathcal{I}(P) & \mathcal{I}(\neg A) = \mathbf{Const} \setminus \mathcal{I}(A)
\end{array}$$

  Observe that $\mathcal{I}(\exists P^-) = \pi_2(\mathcal{I}(P))$.

[2] `http://www.w3.org/TR/owl2-profiles/#OWL_2_QL`

DL-Lite TBox axiom	(first-order translation)
EU-City $\sqsubseteq$ City	$\forall x$ EU-City$(x) \rightarrow$ City(x)
Dutch-City $\sqsubseteq$ EU-City	$\forall x$ Dutch-City$(x) \rightarrow$ EU-City(x)
N.A.-City $\sqsubseteq$ City	$\forall x$ N.A.-City$(x) \rightarrow$ City(x)
EU-City $\sqsubseteq \neg$ N.A.-City	$\forall x$ EU-City$(x) \rightarrow \neg$N.A.-City(x)
US-City $\sqsubseteq$ N.A.-City	$\forall x$ US-City$(x) \rightarrow$ N.A.-City(x)
City $\sqsubseteq \exists$ hasCountry	$\forall x$ City$(x) \rightarrow \exists y$ hasCountry(x, y)
Country $\sqsubseteq \exists$ hasContinent	$\forall x$ Country$(x) \rightarrow \exists y$ hasContinent(x, y)
$\exists$hasCountry$^-$ $\sqsubseteq$ Country	$\forall x$ ($\exists y$ hasCountry$(y, x)) \rightarrow$ Country(x)
$\exists$hasContinent$^-$ $\sqsubseteq$ Continent	$\forall x$ ($\exists y$ hasContinent$(y, x)) \rightarrow$ Continent(x)
$\exists$connected $\sqsubseteq$ City	$\forall x$ ($\exists y$ connected$(x, y)) \rightarrow$ City(x)
$\exists$connected$^-$ $\sqsubseteq$ City	$\forall x$ ($\exists y$ connected$(y, x)) \rightarrow$ City(x)

GAV mapping assertions (universal quantifiers omitted for readability):

Cities$(x, z, w,$ "Europe"$)$	$\rightarrow$	EU-City(x)
Cities$(x, z,$ "Netherlands"$, w)$	$\rightarrow$	Dutch-City(x)
Cities$(x, z, w,$ "N.America"$)$	$\rightarrow$	N.A.-City(x)
Cities$(x, z,$ "USA"$, w)$	$\rightarrow$	US-City(x)
Cities(x, y, z, w)	$\rightarrow$	Continent(w)
Cities(x, k, y, w)	$\rightarrow$	hasCountry(x,y)
Cities(x, k, w, y)	$\rightarrow$	hasContinent$(x$,$y)$
Train-Connection(x, y), Cities(x, x_1, x_2, x_3), Cities(y, y_1, y_2, y_3)	$\rightarrow$	connected$(x$,$y)$

Figure 6.4: Example DL-Lite ontology with mapping assertions.

- *A* TBox (Terminology Box) *is a finite set of* TBox axioms *where each TBox axiom is an inclusion assertion of the form $B \sqsubseteq C$ or $R \sqsubseteq E$, where B is a basic concept expression, C is a concept expression, R is a basic role expression and E is a role expression. An (Φ_C, Φ_R)-interpretation $\mathcal{I}$* satisfies *a TBox if for each axiom $X \sqsubseteq Y$, it holds $\mathcal{I}(X) \subseteq \mathcal{I}(Y)$.*
- *For concept expressions C_1, C_2 and a TBox $\mathcal{T}$, we say that C_1* is subsumed by *C_2* relative to *$\mathcal{T}$ (notation: $\mathcal{T} \models C_1 \sqsubseteq C_2$) if, for all interpretations $\mathcal{I}$ satisfying $\mathcal{T}$, we have that $\mathcal{I}(C_1) \subseteq \mathcal{I}(C_2)$.*

An example of a *DL-Lite$_\mathcal{R}$* TBox is given at the top of Figure 6.4. For convenience, we have listed next to each TBox axiom, its equivalent semantics in first-order notation.

Next we describe what mapping assertions are. Given an ontology and a relational schema, we can specify mapping assertions to relate the ontology language to the relational schema, which is similar to how mappings are used in OBDA (Poggi et al., 2008). In general, mapping assertions are first order sentences over the schema $\mathbf{S} \cup \Phi_C \cup \Phi_R$ that express relationships between the symbols in $\mathbf{S}$ and those in Φ_C and Φ_R. Among the different schema mapping languages that can be used, we restrict our attention, for simplicity, to the class of *Global-As-View (GAV) mapping assertions* (*GAV mapping assertions or GAV constraints* or *GAV source-to-target tgds*).

Definition 6.4.2 (GAV mapping assertions)**.** *A GAV mapping assertion over $(\mathbf{S}, (\Phi_C \cup \Phi_R))$ is a first-order sentence ψ of the form*

$$\forall \vec{x}\ (\varphi_1(\vec{x_1}), \ldots, \varphi_n(\vec{x_n})) \rightarrow \psi(\vec{x}),$$

where $\vec{x} \subseteq \vec{x_1} \cup \ldots \cup \vec{x_n}$, $\varphi_1, \ldots, \varphi_n$ are atoms over $\mathbf{S}$ and ψ is an atomic formula of the form $A(x_i)$ (for $A \in \Phi_C$) or $P(x_i, x_j)$ (for $P \in \Phi_R$). Let I be an $\mathbf{S}$-instance and $\mathcal{I}$ an

(Φ_C, Φ_R)-interpretation. We say that the pair $(I, \mathcal{I})$ satisfies *the GAV mapping assertion (notation: $(I, \mathcal{I}) \models \psi$) if it holds that for any tuple of elements $\bar{a}$ from $adom(I)$, with $\bar{a} = \bigcup_{1 \leq k \leq n} \bar{a_k}$, if $I \models \varphi_1(\bar{a_1}), \ldots, \varphi_n(\bar{a_n})$, then $a_i \in \mathcal{I}(A)$, with $a_i \in \bar{a}$ (if $\psi = A(x_i)$) or $(a_i, a_j) \in \mathcal{I}(P)$, with $a_i, a_j \in \bar{a}$ (if $\psi = P(x_i, x_j)$).*

Intuitively, a GAV mapping assertion associates a conjunctive query over $\mathbf{S}$ to an element (concept or atomic role) of the ontology. A set of GAV mapping assertions associates, in general, a union of conjunctive queries to an element of the ontology. Examples of GAV mapping assertions are given at the bottom of Figure 6.4.

OBDA induced ontologies

Definition 6.4.3 (OBDA specification)**.** *Let $\mathcal{T}$ be a TBox, $\mathbf{S}$ a relational schema, and $\mathcal{M}$ a set of mapping assertions from $\mathbf{S}$ to the concepts of $\mathcal{T}$. We call the triple $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ an* OBDA specification.

An (Φ_C, Φ_R)-interpretation $\mathcal{I}$ is said to be a solution *for an $\mathbf{S}$-instance I with respect to the OBDA specification $\mathcal{B}$ if the pair $(I, \mathcal{I})$ satisfies all mapping assertions in $\mathcal{M}$ and $\mathcal{I}$ satisfies $\mathcal{T}$.*

Note that our notion of an OBDA specification is a special case of the one given in (Di Pinto et al., 2013), where we do not consider view inclusion dependencies. Also, as mentioned earlier, our OBDA specifications assume that $\mathcal{T}$ is a *DL-Lite*$_\mathcal{R}$ TBox and $\mathcal{M}$ is a set of GAV mappings. These restrictions allow us to achieve good complexity bounds for explaining why-not questions with ontologies. In particular, it is not hard to see that, for the OBDA specifications we consider, every $\mathbf{S}$-instance I has a solution.

Theorem 6.4.1. *[Calvanese et al. (2007); Poggi et al. (2008)] Let $\mathcal{T}$ be a DL-Lite$_\mathcal{R}$ TBox.*

(i) There is a PTIME*-algorithm for deciding subsumption. That is, given $\mathcal{T}$ and two concepts C_1, C_2, decide if $\mathcal{T} \models C_1 \sqsubseteq C_2$.*

(ii) There is an algorithm that, given an OBDA specification $\mathcal{B}$, an instance I over $\mathbf{S}$ *and a concept C, computes $certain(C, I, \mathcal{B}) = \bigcap\{\mathcal{I}(C) \mid \mathcal{I}$ is a solution for I w.r.t. $\mathcal{B}\}$. For a fixed OBDA specification, the algorithm runs in* PTIME *(*AC^0 *in data complexity).*

Every OBDA specification induces an $\mathbf{S}$-ontology as follows.

Definition 6.4.4. *Every OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$, where $\mathcal{T}$ is a DL-Lite$_\mathcal{R}$ TBox and $\mathcal{M}$ is a set of GAV mappings gives rise to an $\mathbf{S}$-ontology where:*

- *$\mathcal{C}_{\mathcal{O}_\mathcal{B}}$ is the set of all basic concept expressions occurring in $\mathcal{T}$;*
- *$\sqsubseteq_{\mathcal{O}_\mathcal{B}} = \{(C_1, C_2) \mid \mathcal{T} \models C_1 \sqsubseteq C_2\}$;*
- *$ext_{\mathcal{O}_\mathcal{B}}$ is the polynomial-time computable function given by $ext_{\mathcal{O}_\mathcal{B}}(C, I) = \bigcap\{\mathcal{I}(C) \mid \mathcal{I}$ is a solution for I w.r.t. $\mathcal{B}\}$*

Note that the fact that $ext_{\mathcal{O}_\mathcal{B}}$ is polynomial-time computable follows from Theorem 6.4.1.

We remarked earlier that, for the ODBA specifications $\mathcal{B}$ that we consider, it holds that every input instance has a solution. It follows that every input instance I is consistent with the corresponding $\mathbf{S}$-ontology $\mathcal{O}_\mathcal{B}$.

Theorem 6.4.2. *The* $\mathbf{S}$*-ontology* $\mathcal{O}_{\mathcal{B}} = (\mathcal{C}_{\mathcal{O}_{\mathcal{B}}}, \sqsubseteq_{\mathcal{O}_{\mathcal{B}}}, ext_{\mathcal{O}_{\mathcal{B}}})$ *can be computed from a given OBDA specification* $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ *in* PTIME *if* $\mathcal{T}$ *is a DL-Lite*$_{\mathcal{R}}$ *TBox and* $\mathcal{M}$ *is a set of GAV mappings.*

We are now ready to illustrate an example where a why-not question is explained via an external ontology.

Example 6.4.1. Consider the OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ where $\mathcal{T}$ is the TBox consisting of the *DL-Lite*$_{\mathcal{R}}$ axioms given in Figure 6.4, $\mathbf{S}$ is the schema from Example 6.3.1, and $\mathcal{M}$ is the set of mapping assertions given in Figure 6.4. These together induce an $\mathbf{S}$-ontology $\mathcal{O}_{\mathcal{B}} = (\mathcal{C}_{\mathcal{O}_{\mathcal{B}}}, \sqsubseteq_{\mathcal{O}_{\mathcal{B}}}, ext_{\mathcal{O}_{\mathcal{B}}})$. The set $\mathcal{C}_{\mathcal{O}_{\mathcal{B}}}$ consists of the following basic concept expressions:

City, EU-City, N.A.-City, Dutch-City,
US-City, Country, Continent,
$\exists$ hasCountry, $\exists$ hasCountry$^-$, $\exists$ hasContinent,
$\exists$ hasContinent$^-$, $\exists$ connected, $\exists$ connected$^-$.

The set $\sqsubseteq_{\mathcal{O}_{\mathcal{B}}}$ includes the pairs of concepts of the TBox $\mathcal{T}$ given in Figure 6.4. We use the mappings to compute the extension of each concept in $\mathcal{C}_{\mathcal{O}_{\mathcal{B}}}$ using the instance I on the left of Figure 6.2. We list a few extensions here:

$$\begin{array}{lcl}
ext_{\mathcal{O}_{\mathcal{B}}}(\text{City}, I) & = & \{\text{Amsterdam, Berlin, Rome, New York,} \\
 & & \ \text{San Francisco, Santa Cruz, Tokyo, Kyoto}\} \\
ext_{\mathcal{O}_{\mathcal{B}}}(\text{EU-City}, I) & = & \{\text{Amsterdam, Berlin, Rome}\} \\
ext_{\mathcal{O}_{\mathcal{B}}}(\text{N.A.-City}, I) & = & \{\text{New York, San Francisco, Santa Cruz}\} \\
ext_{\mathcal{O}_{\mathcal{B}}}(\exists\text{hasCountry}^-, I) & = & \{\text{Netherlands, Germany, Italy, USA, Japan}\} \\
ext_{\mathcal{O}_{\mathcal{B}}}(\exists\text{connected}, I) & = & \{\text{Amsterdam, Berlin, New York}\}
\end{array}$$

Now consider the query $q(x, y) = \exists z.$ Train-Connections$(x, z) \wedge$ Train-Connections(z, y), and $q(I)$ as in Example 6.3.1. As before, we would like to explain why is $\langle$Amsterdam, New York$\rangle \notin q(I)$. This time, we use the induced $\mathbf{S}$-ontology $\mathcal{O}_{\mathcal{B}}$ described above to derive explanations for $\langle$Amsterdam, New York$\rangle \notin q(I)$:

$$\begin{array}{llll}
E_1 & = \langle\text{EU-City, N.A.-City}\rangle & E_2 & = \langle\text{Dutch-City,N.A.-City}\rangle \\
E_3 & = \langle\text{EU-City, US-City}\rangle & E_4 & = \langle\text{Dutch-City,US-City}\rangle.
\end{array}$$

Among the four explanations above, E_1 is a most general one. $\square$

6.4.2 Ontologies derived from a schema

We now move to the second approach where an ontology is derived from an instance or a schema. The ability to derive an ontology through an instance or a schema is useful in the context where an external ontology is unavailable. To this purpose we first introduce a simple but suitable concept language that can be defined over the schema $\mathbf{S}$.

Specifically, our concept language, denoted as $L_{\mathbf{S}}$, makes use of two relational algebra operations, projection (π) and selection (σ). We first introduce and motivate the language. We will then describe our complexity results for testing whether one concept is subsumed by another, and for obtaining an ontology from a given instance or a schema. We will make use of these results later on in Sections 6.5.2 and 6.5.3.

Definition 6.4.5 (The Concept Language $L_{\mathbf{S}}$)**.** *Let* $\mathbf{S}$ *be a schema. A concept in* $L_{\mathbf{S}}$ *is an expression* C *defined by the following grammar.*

$$D ::= R \mid \sigma_{A_1 \,\mathsf{op}\, c_1, \ldots, A_n \,\mathsf{op}\, c_n}(R)$$
$$C := \top \mid \{c\} \mid \pi_A(D) \mid C \sqcap C$$

In the above, R is a predicate name from $\mathbf{S}$, $A, A_1, \ldots, A_n$ are attributes in R, not necessarily distinct, $c, c_1, \ldots, c_n \in \mathbf{Const}$, and each occurrence of op *is a comparison operator belonging to $\{=, <, >, \leq, \geq\}$. For $\mathbf{C} = \{C_1, \ldots, C_k\}$ a finite set of concepts, we denote by $\sqcap\mathbf{C}$ the conjunction $C_1 \sqcap \ldots \sqcap C_k$. If $\mathbf{C}$ is empty, we take $\sqcap\mathbf{C}$ to be $\top$.*

Given a finite set of constants $\mathcal{K} \subset \mathbf{Const}$, we define $L_{\mathbf{S}}[\mathcal{K}]$ as the concept language $L_{\mathbf{S}}$ whose concept expressions only use constants from $\mathcal{K}$. By selection-free *$L_{\mathbf{S}}$, we mean the language $L_{\mathbf{S}}$ where σ is not allowed. Similarly, by* intersection-free *$L_{\mathbf{S}}$, we mean the language $L_{\mathbf{S}}$ where $\sqcap$ is not allowed, and by $L_{\mathbf{S}}^{\min}$, we mean the minimal concept language $L_{\mathbf{S}}$ where both σ and $\sqcap$ are not allowed.*

Observe that the $L_{\mathbf{S}}$ grammar defines a concept in the form $C_1 \sqcap \ldots \sqcap C_n$ where each C_i is $\top$ or $\{c\}$ or $\pi_A(R)$ or $\pi_A(\sigma_{A_1 \,\mathsf{op}\, c_1, \ldots, A_n \,\mathsf{op}\, c_n}(R))$. A concept of the form $\{c\}$ is called a *nominal*. A nominal $\{c\}$ is the "most specific" concept for the constant c. Given a tuple $\overline{a}$ that is not in the output, the corresponding tuple of nominal concepts forms a default, albeit trivial, explanation for why not $\overline{a}$.

As our next example illustrates, even though our concept language $L_{\mathbf{S}}$ appears simple, it is able to naturally capture many intuitive concepts over the domain of the database.

Example 6.4.2. We refer back to our schema **S** in Figure 6.1. Suppose we do not have access to an external ontology such as the one given in Example 6.3.1. We show that even so, we can still construct meaningful concepts directly from the database schema using the concept language described above. We list a few semantic concepts that can be specified with $L_{\mathbf{S}}$ in Figure 6.5, where we also show the corresponding SELECT-FROM-WHERE style expressions and intuitive meaning. □

Example 6.4.2 shows that, even though $L_{\mathbf{S}}$ is a simple language where concepts are essentially intersections of unary projections of relations and nominals, it is already sufficiently expressive to capture natural concepts that can be used to build meaningful explanations. It is worth noting that, for minor extensions of the language $L_{\mathbf{S}}$, such as with $\neq$-comparisons and disjunction, the notion of a most-general explanation becomes trivial, in the sense that, for each why-not question, there is a most-general explanation that essentially enumerates all tuples in the query answer.

By using $L_{\mathbf{S}}$, we are able to define an ontology whose atomic concepts are derived from the schema itself. This approach allows us to provide explanations using a vocabulary that is already familiar to the user. We believe that this leads to intuitive and useful why-not explanations.

If we view each expression $\pi_A(D)$ as an atomic concept, then the language $L_{\mathbf{S}}$ corresponds to a very simple concept language, whose concepts are built from atomic concepts and nominals using only intersection. In this sense, $L_{\mathbf{S}}$ can be considered to be a fragment of *DL-Lite*$_{core,\sqcap}$ with nominals (also known as *DL-Lite*$_{horn}$ (Artale et al., 2009)), i.e., the description logic obtained by enriching *DL-Lite*$_{core}$ (the simplest language in the DL-Lite family) with conjunction.

The precise semantics of $L_{\mathbf{S}}$ is as follows. Given a concept C that is defined in $L_{\mathbf{S}}$ and an instance I over **S**, the extension of C in I, denoted by $[\![C]\!]^I$, is inductively defined

$L_{\mathbf{S}}$ **concept expression**	SELECT-FROM-WHERE **formulation**	**Intuitive meaning**
$\pi_{\text{name}}(\text{Cities})$	name from Cities	City
$\pi_{\text{name}}(\sigma_{\text{continent}=\text{“Europe”}}(\text{Cities}))$	name from Cities where continent=“Europe”	European City
$\pi_{\text{name}}(\sigma_{\text{continent}=\text{“N.America”}}(\text{Cities}))$	name from Cities where continent=“N.America”	N.American City
$\pi_{\text{name}}(\sigma_{\text{population}>1000000}(\text{Cities}))$	name from Cities where population>1000000	Large City
$\pi_1(\text{BigCity})$	name from BigCity	name of BigCity
$\{\text{“Santa Cruz”}\}$	“Santa Cruz”	Santa Cruz
$\pi_{\text{name}}(\sigma_{\text{population}<1000000}(\text{Cities}))\sqcap$ $\pi_{\text{city\_to}}(\sigma_{\text{city\_from}=\text{“Amsterdam”}}(\text{Reachable}))$	name from Cities where population<1000000 AND city_from from Reachable where city_to=“Amsterdam”	Small City that is reachable from Amsterdam.

Figure 6.5: Example of concepts specified in $L_{\mathbf{S}}$.

below. Intuitively, the extension of C in I is the result of evaluating the query associated with C over I.

$$\begin{array}{rcl}
[\![R]\!]^I & = & R^I \\
[\![\sigma_{A_1 \mathtt{op}_1 c_1,\ldots,A_n \mathtt{op}_n c_n}(R)]\!]^I & = & \{\bar{b} \in R^I \mid \pi_{A_i}(\bar{b})\mathtt{op}_i c_i, 1 \leq i \leq n\} \\
[\![\top]\!]^I & = & \mathbf{Const} \\
[\![\{c\}]\!]^I & = & \{c\} \\
[\![\pi_A(D)]\!]^I & = & \pi_A([\![D]\!]^I) \\
[\![C_1 \sqcap C_2]\!]^I & = & [\![C_1]\!]^I \cap [\![C_2]\!]^I
\end{array}$$

The notion of when one concept is subsumed by another is defined according to the extensions of the concepts. There are two notions, corresponding to concept subsumption w.r.t. an instance or subsumption w.r.t. a schema. More precisely, given two concepts C_1, C_2,

- we say that C_2 *subsumes* C_1 *w.r.t. an instance* I (notation: $C_1 \sqsubseteq_I C_2$) if $[\![C_1]\!]^I \subseteq [\![C_2]\!]^I$.
- we say that C_2 *subsumes* C_1 *w.r.t. a schema* $\mathbf{S}$ (notation: $C_1 \sqsubseteq_{\mathbf{S}} C_2$), if for every instance I of $\mathbf{S}$, we have that $C_1 \sqsubseteq_I C_2$.

Note that the latter item is precisely the definition of containment of the query corresponding to C_1 in the query corresponding to C_2.

We are now ready to define the two types of ontologies, which are based on the two notions of concept subsumption described above, that can be derived from an instance or a schema.

Definition 6.4.6 (Ontologies derived from a schema)**.** *Let* $\mathbf{S}$ *be a schema, and let* I *be an instance of* $\mathbf{S}$*. Then the ontologies derived from* $\mathbf{S}$ *and* I *are defined respectively as*

- $\mathcal{O}_{\mathbf{S}} = (L_{\mathbf{S}}, \sqsubseteq_{\mathbf{S}}, ext)$ *and*
- $\mathcal{O}_I = (L_{\mathbf{S}}, \sqsubseteq_I, ext)$*,*

where ext *is the function given by* $ext(C, I') = [\![C]\!]^{I'}$ *for all instances* I' *over* $\mathbf{S}$*. By* $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ *we denote the ontology* $(L_{\mathbf{S}}[\mathcal{K}], \sqsubseteq_{\mathbf{S}}, ext)$*, and by* $\mathcal{O}_I[\mathcal{K}]$ *we denote the ontology* $(L_{\mathbf{S}}[\mathcal{K}], \sqsubseteq_I, ext)$*.*

It is easy to verify that the subsumption relations $\sqsubseteq_{\mathbf{S}}$ and $\sqsubseteq_I$ are indeed pre-orders (i.e., reflexive and transitive relations), and that, for every fixed schemas $\mathbf{S}$, the function $[\![C]\!]^{I'}$ is polynomial-time computable. Hence, the above definition is well-defined even

Constraints	Complexity of subsumption for $L_{\mathbf{S}}$
UCQ-view def. (no comparisons)	NP-complete
UCQ-view def.	Π_2^P-complete
linearly nested UCQ-view def.	Π_2^P-complete
nested UCQ-view def.	coNEXPTIME-complete
Functional Dependencies (FDs)	in PTIME
Inclusion Dependencies (IDs)	? (in PTIME for selection-free $L_{\mathbf{S}}$)
IDs + FDs	Undecidable

All stated lower bounds already hold for $L_{\mathbf{S}}^{\min}$ concept expressions.

Table 6.1: Complexity of concept subsumption.

though the ontologies obtained in this way are typically infinite. From the definition, it is easy to verify that if $C_1 \sqsubseteq_{\mathbf{S}} C_2$, then $C_1 \sqsubseteq_I C_2$.

The following result about deciding $\sqsubseteq_I$ is immediate, as one can always execute the queries that are associated with the concepts and then test for subsumption, which can be done in polynomial time.

Proposition 6.4.1. *The problem of deciding, given an instance I of a schema $\mathbf{S}$ and given two $L_{\mathbf{S}}$ concept expressions C_1, C_2, whether $C_1 \sqsubseteq_I C_2$, is in* PTIME.

On the other hand, the complexity of deciding $\sqsubseteq_{\mathbf{S}}$ depends on the type of integrity constraints that are used in the specification of $\mathbf{S}$. Table 6.1 provides a summary of relevant complexity results.

Theorem 6.4.3. *Let $\mathcal{W}$ be one of the different classes of schemas with integrity constraints listed in Table 6.1. The complexity of the problem to decide, given a schema $\mathbf{S}$ in $\mathcal{W}$ and two $L_{\mathbf{S}}$ concept expressions C_1, C_2, whether $C_1 \sqsubseteq_{\mathbf{S}} C_2$, is as indicated in the second column of the corresponding row in Table 6.1.*

For example, given two concepts C_1, C_2, and a schema $(\mathbf{S}, \Sigma)$ where Σ is a collection of nested UCQ-view definitions, the complexity of deciding $C_1 \sqsubseteq_{\mathbf{S}} C_2$ is coNEXPTIME-complete. The lower bound already holds for concepts specified in $L_{\mathbf{S}}^{\min}$. We conclude this section with an analysis of the number of distinct concepts that can be formulated in a given concept language and an example that illustrates explanations that can be computed from such derived ontologies.

Proposition 6.4.2. *Given a schema $\mathbf{S}$ and a finite set of constants $\mathcal{K} \subset$ $\mathbf{Const}$, the number of unique concepts (modulo logical equivalence)*

- *in $L_{\mathbf{S}}^{\min}[\mathcal{K}]$ is polynomial in the size of $\mathbf{S}$ and $\mathcal{K}$,*
- *in selection-free or intersection-free $L_{\mathbf{S}}[\mathcal{K}]$ is single exponential in the size of $\mathbf{S}$ and $\mathcal{K}$.*
- *in $L_{\mathbf{S}}[\mathcal{K}]$ is double exponential in the size of $\mathbf{S}$ and $\mathcal{K}$.*

Example 6.4.3. Let $\mathbf{S}$ and I be the schema and instance from Figure 6.1 and Figure 6.2. Suppose the concept language $L_{\mathbf{S}}$ is used to define among others the concepts from Figure 6.5. The following concept subsumptions can be derived from $\mathbf{S}$. Note that subsumption $\sqsubseteq_{\mathbf{S}}$ implies $\sqsubseteq_{I}$.

$$\begin{array}{lll}
\pi_{\text{name}}(\sigma_{\text{continent=“Europe”}}(\text{Cities})) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{name}}(\text{Cities}) \\
\pi_{\text{name}}(\sigma_{\text{population}>7000000}(\text{Cities})) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{name}}(\text{BigCity}) \\
\pi_{\text{name}}(\text{BigCity}) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{name}}(\text{Cities}) \\
\pi_{\text{name}}(\text{BigCity}) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{city\_from}}(\text{Train-Connections})
\end{array}$$

The first and second subsumptions follow from definitions. The third one holds because according to Π, a BigCity is a city with population more than 5 million. The fourth subsumption follows from the inclusion dependency that each BigCity must have a train departing from it. There are subsumptions that hold in $\mathcal{O}_I$ but not in $\mathcal{O}_{\mathbf{S}}$. For instance,

$$\pi_{\text{city\_to}}(\sigma_{\text{city\_from=Amsterdam}}(\text{Reachable})) \sqsubseteq_I \pi_{\text{city\_to}}(\sigma_{\text{city\_from=Berlin}}(\text{Reachable})),$$

holds w.r.t. $\mathcal{O}_I$, where I is the instance given in Figure 6.2, but does not hold w.r.t. $\mathcal{O}_{\mathbf{S}}$, since one can construct an instance where not all cities that are reachable from Amsterdam are reachable from Berlin.

We now give examples of most-general explanations w.r.t. $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. As before, let $q(x,y) = \exists z.$ Train-Connections$(x,z) \wedge$ Train-Connections(z,y) be a query with $q(I) = \{\langle\text{Amsterdam, Rome}\rangle, \langle\text{Amsterdam, Amsterdam}\rangle, \langle\text{Berlin, Berlin}\rangle, \langle\text{New York, Santa Cruz}\rangle\}$. We would like to explain why $\langle\text{Amsterdam, New York}\rangle \notin q(I)$ using the derived ontologies $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. Note that if E is an explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$, then it is also an explanation w.r.t. $\mathcal{O}_I$ and vice versa. Some possible explanations are:

$$\begin{array}{lll}
E_1 & = & \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \\
& & \quad \pi_{\text{city\_from}}(\sigma_{\text{city\_to = San Francisco}}(\text{Train-Connections}))\rangle \\
E_2 & = & \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \\
& & \quad \pi_{\text{name}}(\sigma_{\text{continent=N.America}}(\text{Cities}))\rangle \\
E_3 & = & \langle \pi_{\text{city\_to}}(\sigma_{\text{city\_from = Berlin}}(\text{Reachable})), \\
& & \quad \pi_{\text{city\_from}}(\sigma_{\text{city\_to = Santa Cruz}}(\text{Reachable}))\rangle \\
E_4 & = & \langle \{\text{Amsterdam}\}, \pi_{\text{name}}(\sigma_{\text{population}>7000000}(\text{Cities}))\rangle \\
E_5 & = & \langle \pi_{\text{name}}(\sigma_{\text{country=Netherlands}}(\text{Cities})), \\
& & \quad \pi_{\text{name}}(\text{BigCity}) \sqcap \pi_{\text{name}}(\sigma_{\text{continent=N.America}}(\text{Cities}))\rangle \\
E_6 & = & \langle \{\text{Amsterdam}\}, \{\text{New York}\}\rangle \\
E_7 & = & \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \pi_{\text{name}}(\text{BigCity})\}\rangle \\
E_8 & = & \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \\
& & \quad \pi_{\text{name}}(\sigma_{\text{population}>7000000}(\text{Cities}))\}\rangle
\end{array}$$

For example, E_1 states the reason is that Amsterdam is a European city and New York is a city that has a train connection to San Francisco, and there is no train connection between such cities via a city. The trivial explanation E_6 is less general than any other explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$ (and $\mathcal{O}_I$ too). It can be verified that E_2 and E_7 are most-general explanations w.r.t. both $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. In particular, $E_2 >_{\mathcal{O}_I} E_5$ and $E_2 \geq_{\mathcal{O}_I} E_3$, but $E_2 \not>_{\mathcal{O}_{\mathbf{S}}} E_5$ and $E_2 \not>_{\mathcal{O}_{\mathbf{S}}} E_3$ since there might be an instance of $\mathbf{S}$ where Netherlands is not in Europe or where Berlin is reachable from a non-European city. □

In general, if E is an explanation w.r.t. $\mathcal{O}_I$ then E is also an explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$, and vice versa. The following proposition also describes the relationship between most-general explanations w.r.t $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$.

Proposition 6.4.3. *Let* $\mathbf{S}$ *be a schema, and let* I *be an instance of* $\mathbf{S}$.

(i) Every explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$ *is an explanation w.r.t.* $\mathcal{O}_I$ *and vice versa.*

(ii) A most-general explanation w.r.t $\mathcal{O}_{\mathbf{S}}$ *is not necessarily a most-general explanation w.r.t.* $\mathcal{O}_I$*, and likewise vice versa.*

Proof. Statement (i) follows from Definition 6.3.2 and the definition of ext for $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. That is, ext is the same on the input instance I for both $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$, and the conditions of Definition 6.3.2 use only the value of ext on I. Going back to Example 6.4.3, E_1 is a most-general explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$, but it is not a most-general explanation w.r.t. $\mathcal{O}_I$ (since E_3 is a strictly more general explanation than E_1 w.r.t. $\mathcal{O}_I$). Thus, the first direction of (ii) holds. For the other direction of (ii), consider E_8 which is a most-general explanation w.r.t. $\mathcal{O}_I$. But it holds that $E_7 >_{\mathcal{O}_{\mathbf{S}}} E_8$ and E_7 is an explanation. Note that E_7 and E_8 are equivalent w.r.t. $\mathcal{O}_I$. □

6.5 Algorithms for computing most-general explanations

Next, we formally introduce the ontology-based why-not problem, which was informally described in Section 6.3, and we define algorithms for computing most-general explanations. We start by defining the notion of a why-not instance (or why-not question).

Definition 6.5.1 (Why-not instance)**.** *Let* $\mathbf{S}$ *be a schema,* I *an instance of* $\mathbf{S}$*,* q *an* m*-ary query over* I *and* $\overline{a} = \langle a_1, \ldots, a_m \rangle$ *a tuple of constants such that* $\overline{a} \notin q(I)$*. We call the quintuple* $(\mathbf{S}, I, q, Ans, \overline{a})$*, where* $Ans = q(I)$*, a* why-not instance *or a* why-not question.

In a why-not instance, the answer set Ans of q over I is assumed to have been computed already. This corresponds closely to the scenario under which why-not questions are posed where the user requests explanations for why a certain tuple is missing in the output of a query, which is computed a priori. Note that since $Ans = q(I)$ is part of a why-not instance, the complexity of evaluating q over I does not affect the complexity analysis of the problems we study in this chapter. In addition, observe that although a query q is part of a why-not instance, the query is not directly used in our derivation of explanations for why-not questions with ontologies. However, the general setup accommodates the possibility to consider q directly in the derivation of explanations and this is part of our future work.

We will study the following algorithmic problems concerning most-general explanations for a why-not instance.

Definition 6.5.2. *The* EXISTENCE-OF-EXPLANATION *problem is the following decision problem: given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an* $\mathbf{S}$*-ontology* $\mathcal{O}$ *consistent with* I*, does there exist an explanation for* $\overline{a} \notin Ans$ *w.r.t.* $\mathcal{O}$*?*

Definition 6.5.3. *The* CHECK-MGE *problem is the following decision problem: given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$*, an* $\mathbf{S}$*-ontology* $\mathcal{O}$ *consistent with* I*, and a tuple of concepts* $(C_1, \ldots, C_n)$*, is the given tuple of concepts a most-general explanation w.r.t.* $\mathcal{O}$ *for* $\overline{a} \notin Ans$*?*

Definition 6.5.4. *The* COMPUTE-ONE-MGE *problem is the following computational problem: given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an* $\mathbf{S}$*-ontology* $\mathcal{O}$ *consistent with* I*, find a most-general explanation w.r.t.* $\mathcal{O}$ *for* $\overline{a} \notin Ans$*, if one exists.*

Note that deciding the existence of an explanation w.r.t. a finite $\mathbf{S}$-ontology is equivalent to deciding existence of a most-general explanation w.r.t. the same $\mathbf{S}$-ontology.

Thus, our approach to the why-not problem makes use of $\mathbf{S}$-ontologies. In particular, our notion of a "best explanation" is a *most-general explanation*, which is defined with respect to an $\mathbf{S}$-ontology. We study the problem in three flavors: one in which the $\mathbf{S}$-ontology is obtained from an external source (Section 6.5.1), and thus it is part of the input, and two in which the $\mathbf{S}$-ontology is not part of the input, and is derived, respectively, from the instance I (Section 6.5.2), or from the schema $\mathbf{S}$ (Section 6.5.3).

6.5.1 Case 1: External ontology

We start by studying the case of computing ontology-based why-not explanations w.r.t. an external $\mathbf{S}$-ontology. We first study the complexity of deciding whether or not there exists an explanation w.r.t. an external $\mathbf{S}$-ontology.

Theorem 6.5.1.

(i) The problem CHECK-MGE *is solvable in* PTIME.

(ii) The problem EXISTENCE-OF-EXPLANATION *is* NP*-complete. It remains* NP*-complete even for bounded schema arity.*

Intuitively, to check if a tuple of concepts is a most-general explanation, we can first check in PTIME if it is an explanation. Then, for each concept in the explanation, we can check in PTIME if it is subsumed by some other concept in $\mathcal{O}$ such that by replacing it with this more general concept, the tuple of concepts remains an explanation. The membership in NP is due to the fact that we can guess a tuple of concepts of polynomial size and verify in PTIME that it is an explanation. The lower bound is by a reduction from the SET COVER problem. Our reduction uses a query of unbounded arity and a schema of bounded arity. As we will show in Theorem 6.5.2, the problem is in PTIME if the arity of the query is fixed.

In light of the above result, we define an algorithm, called the EXHAUSTIVE SEARCH ALGORITHM, which is an EXPTIME algorithm for solving the COMPUTE-ONE-MGE problem.

This algorithm first generates the set of all possible explanations, and then iteratively reduces the set by removing the tuples of concepts that are less general than some tuple of concepts in the set. In the end, only most-general explanations are returned. At first, in line 1, for each element of the tuple $\overline{a} = \langle a_1, \dots, a_m \rangle$, we build the set $\mathcal{C}(a_i)$ containing all the concepts in $\mathcal{C}$ whose extension contains a_i. Then, in line 2, we build the set of all possible explanations by picking a concept in $\mathcal{C}(a_i)$ for each position in $\overline{a}$, and by discarding the ones that have a non empty intersection with the answer set Ans. Finally, in lines 3–5, we remove from the set those explanations that have a strictly more general explanation in the set.

We now show that EXHAUSTIVE SEARCH ALGORITHM is correct (i.e., it outputs the set of all most-general explanations for the given why-not instance w.r.t. the given $\mathbf{S}$-ontology), and runs in exponential time in the size of the input.

Algorithm 1: EXHAUSTIVE SEARCH ALGORITHM

```
Input: a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$, where $\overline{a} = (a_1, \ldots, a_m)$, a finite
       S-ontology $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$
Output: the set of most-general explanations for $\overline{a} \notin Ans$ wrt $\mathcal{O}$
1 Let $\mathcal{C}(a_i) = \{C \in \mathcal{C} \mid a_i \in ext(C, I)\}$ for all $i, 1 \leq i \leq m$
2 Let
  $\mathcal{X} = \{(C_1, \ldots, C_m) \mid C_i \in \mathcal{C}(a_i)$ and $(ext(C_1, I) \times \ldots \times ext(C_m, I)) \cap Ans = \emptyset\}$
3 foreach pair of explanations $E_1, E_2 \in \mathcal{X}$, $E_1 \neq E_2$ do
4     if $E_1 >_{\mathcal{O}} E_2$ then
5         remove $E_2$ from $\mathcal{X}$
6 return $\mathcal{X}$
```

Theorem 6.5.2. *Let a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and an $\mathbf{S}$-ontology $\mathcal{O}$ be an input to* EXHAUSTIVE SEARCH ALGORITHM *and let $\mathcal{X}$ be the corresponding output. The following hold:*

(i) $\mathcal{X}$ is the set of all most-general explanations for $\overline{a} \notin Ans$ (modulo equivalence);

(ii) EXHAUSTIVE SEARCH ALGORITHM *runs in* EXPTIME *in the size of the input (in* PTIME *if we fix the arity of the input query).*

Theorem 6.5.2, together with Theorem 6.4.2, yields the following corollary (recall that, by construction of $\mathcal{O}_{\mathcal{B}}$, it holds that every input instance I is consistent with $\mathcal{O}_{\mathcal{B}}$).

Corollary 6.5.1. *There is an algorithm that takes as input a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and an OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$, where $\mathcal{T}$ is a DL-Lite$_{\mathcal{R}}$ TBox and $\mathcal{M}$ is a set of GAV mappings, and computes all the most-general explanations for $\overline{a} \notin Ans$ w.r.t. the $\mathbf{S}$-ontology $\mathcal{O}_{\mathcal{B}}$ in* EXPTIME *in the size of the input (in* PTIME *if the arity of q is fixed) .*

6.5.2 Case 2: Ontologies from an instance

We now study the why-not problem w.r.t. an $\mathbf{S}$-ontology O_I that is derived from an instance. First, note that the presence of nominals in the concept language guarantees a trivial answer for the EXISTENCE-OF-EXPLANATION W.R.T. $\mathcal{O}_I$ problem. An explanation always exists, namely the explanation with nominals corresponding to the constants of the tuple $\overline{a}$. In fact, a *most-general explanation* always exists, as follows from the results below.

Definition 6.5.5. *The* COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_I$ *is the following computational problem: given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$, find a most-general explanation w.r.t. $\mathcal{O}_I$ for $\overline{a} \notin Ans$, where $\mathcal{O}_I$ is the $\mathbf{S}$-ontology that is derived from I, as defined in Section 6.4.2.*

First, we state an important proposition, that underlies the correctness of the algorithms that we will present. The following proposition shows that, when we search for explanations w.r.t. $\mathcal{O}_I$, we can always restrict our attention to a particular finite restriction of this ontology.

Proposition 6.5.1. *Let $(\mathbf{S}, I, q, Ans, \overline{a})$ be a why-not instance. If E is an explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I$ (resp. $\mathcal{O}_\mathbf{S}$), then there exists an explanation E' for $\overline{a} \notin Ans$ such that $E <_{\mathcal{O}_I[\mathcal{K}]} E'$ (resp. $E <_{\mathcal{O}_\mathbf{S}[\mathcal{K}]} E'$), where $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$ and each constant in E' belongs to $\mathcal{K}$.*

In our proof, we iteratively reduce the number of constants occurring in the explanation. That is, for every explanation E with concepts containing constants outside of $\mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$, we produce a new explanation E' which is more general than E and which contains less constants outside of $\mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$.

Notice that since, in principle, it is possible to materialize the ontology $\mathcal{O}_I[\mathcal{K}]$ (i.e., to explicitly compute all the concepts $\mathcal{C}$ in the ontology, the subsumption relation $\sqsubseteq_I$, and the extension ext), the EXHAUSTIVE SEARCH ALGORITHM, together with Proposition 6.5.1, give us a method for solving COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_I$. In particular, given a schema, EXHAUSTIVE SEARCH ALGORITHM solves COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_I$ in 2EXPTIME (in EXPTIME if the arity of q is fixed). This is because to find a most-general explanation w.r.t $\mathcal{O}_I$, it is sufficient to restrict to the concept language $L_\mathbf{S}[\mathcal{K}]$ and its fragments, where $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$. Then COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_I$ is solvable in 2EXPTIME follows from the fact that the $\mathbf{S}$-ontology $\mathcal{O}_I[\mathcal{K}]$ is computable in at most 2EXPTIME.

We now present a more effective algorithm for solving COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_I$. (See Algorithm 2.) We start by introducing the notion of a *least upper bound* of a set of constants X w.r.t. an instance I, denoted by $\mathsf{lub}_I(X)$. This, intuitively, corresponds to the most-specific concept whose extension contains all constants of X. We first consider the case in which $\mathsf{lub}_I(X)$ is expressed using selection-free $L_\mathbf{S}$ concepts. The following lemma states two important properties of $\mathsf{lub}_I(X)$ that are crucial for the correctness of Algorithm 2.

Lemma 6.5.1. *Given an instance I of schema $\mathbf{S}$ and a set of constants X, we can compute in polynomial time a selection-free $L_\mathbf{S}$ concept, denoted $\mathsf{lub}_I(X)$, that is the smallest concept whose extension contains all the elements in X definable in the language. In particular, the following hold:*

(i) $X \subseteq ext(\mathsf{lub}_I(X), I)$,

(ii) there is no concept C' in selection-free $L_\mathbf{S}$ such that $C' \sqsubset_I \mathsf{lub}_I(X)$ and $X \subseteq ext(C', I)$.

We are now ready to introduce Algorithm 2. We will start with a high-level description of the idea behind it. The algorithm navigates through the search space of possible explanations using an incremental search strategy and makes use of the above defined notion of lub. We start with an explanation that has, in each position, the lub of the constant (i.e., nominal) that occurs in that position. Then, we try to construct a more general explanation by expanding the set of constants considered by each lub.

Notice that INCREMENTAL SEARCH ALGORITHM produces explanations which are tuples of conjunctions of concepts. Therefore it produces an explanation whose concepts are concept expressions in the language $L_\mathbf{S}$ or selection-free $L_\mathbf{S}$. We will study the behavior of the algorithm for each of these languages separately.

First, we focus on the case in which INCREMENTAL SEARCH ALGORITHM produces most-general explanations using selection-free $L_\mathbf{S}$ concepts. We show that the algorithm

Algorithm 2: INCREMENTAL SEARCH ALGORITHM

```
   Input: a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$
   Output: a most-general explanation for $\overline{a} \notin Ans$ wrt $\mathcal{O}_I$
1  Let $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$
2  Let $\mathcal{X} = (X_1, \ldots, X_m)$ s.t. each $X_j = \{a_j\}$. // support set
3  Let $E = (C_1, \ldots, C_m)$ s.t. each $C_j = \mathsf{lub}_I(X_j)$. // first candidate explanation
4  foreach $1 \leq j \leq m$ do
5      foreach $b \in \mathcal{K} \setminus ext(E_j, I)$ do
6          $X'_j = X_j \cup \{b\}$
7          Let $C'_j = \mathsf{lub}_I(X'_j)$ // a more general concept in position j
8          Let $E' := (C_1, \ldots, C'_j, \ldots C_m)$ // a more general explanation
9          if $E' \cap Ans = \emptyset$ then
10             $E := E'$
11             $\mathcal{X} := (X_1, \ldots, X'_j, \ldots X_m)$
12 return $E$
```

is correct, i.e., that it outputs an explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I$, and that it runs in polynomial time with selection-free $L_{\mathbf{S}}$.

Theorem 6.5.3 (Correctness and running time of INCREMENTAL SEARCH ALGORITHM)**.** *Let the why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *be an input to* INCREMENTAL SEARCH ALGORITHM *and E the corresponding output. The following holds:*

(i) E is a most-general explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I = (\mathcal{C}, \sqsubseteq_I, ext)$, where $\mathcal{C}$ is selection-free $L_{\mathbf{S}}$;

(ii) INCREMENTAL SEARCH ALGORITHM *runs in* PTIME *in the size of the input.*

Now we extend our analysis of INCREMENTAL SEARCH ALGORITHM to the general case in which it works with $L_{\mathbf{S}}$. First, we state an analogue of Lemma 6.5.1 for $L_{\mathbf{S}}$.

Lemma 6.5.2. *Given an instance I of* $\mathbf{S}$ *and a set of constants X, we can compute in exponential time a $L_{\mathbf{S}}$ concept, denoted $\mathsf{lub}^{\sigma}_I(X)$, that is the smallest concept whose extension contains all the elements in X definable in the language. Such a concept is polynomial-time computable for bounded schema arity. In particular, the following hold:*

(i) $X \subseteq ext(\mathsf{lub}^{\sigma}_I(X), I)$,

(ii) there is no concept C' in $L_{\mathbf{S}}$ such that $C' \sqsubset_I \mathsf{lub}^{\sigma}_I(X)$ and $X \subseteq ext(C', I)$.

By INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS we will refer to the algorithm obtained from INCREMENTAL SEARCH ALGORITHM by replacing $\mathsf{lub}_I(X)$ with $\mathsf{lub}^{\sigma}_I(X)$ in line 3 and line 7.

The following theorem shows that INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS is correct, i.e., that it outputs an explanation for $\overline{a} \notin Ans$ w.r.t. the $\mathbf{S}$-ontology $\mathcal{O}_I$, and that it runs in exponential time (in polynomial time for bounded schema arity).

Theorem 6.5.4 (Correctness and running time of INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS)**.** *Let the why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *be an input to* INCRE-

MENTAL SEARCH ALGORITHM WITH SELECTIONS *and E the corresponding output. The following hold:*

(i) E is a most-general explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I = (\mathcal{C}, \sqsubseteq_I, ext)$, where $\mathcal{C}$ is $L_{\mathbf{S}}$;

(ii) INCREMENTAL SEARCH ALGORITHM *runs in* EXPTIME *in the size of the input (in* PTIME *for bounded schema arity).*

We close this section with the study of the following problem.

Definition 6.5.6. *The* CHECK-MGE W.R.T. $\mathcal{O}_I$ *problem is the following decision problem: given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and a tuple of concepts $E = (C_1, \ldots, C_n)$, is E a most-general explanation w.r.t. $\mathcal{O}_I$ for $\overline{a} \notin Ans$?*

Our next proposition states the running time of our algorithm for the CHECK-MGE W.R.T. $\mathcal{O}_I$ for various fragments of our concept language. The algorithm operates very similarly to lines 4–11 of INCREMENTAL SEARCH ALGORITHM. Given a tuple of concepts, we check whether that tuple of concepts can be extended to a more general tuple of concepts through ideas similar to lines 4–11 of INCREMENTAL SEARCH ALGORITHM. If the answer is "no", then we return "yes". Otherwise, we return "no".

Proposition 6.5.2. *There is an algorithm that solves* CHECK-MGE W.R.T. $\mathcal{O}_I$ *in:*

- PTIME *for selection-free $L_{\mathbf{S}}$, or for $L_{\mathbf{S}}$ with bounded schema arity;*
- EXPTIME *for $L_{\mathbf{S}}$ in the general case.*

6.5.3 Case 3: Ontologies from schema

We now study the case of solving the why-not problem w.r.t. to an $\mathbf{S}$-ontology $O_{\mathbf{S}}$ that is derived from a schema. As in the previous case, the presence of nominals in the concept language guarantees that the trivial explanation always exists. Therefore we do not consider the decision problem EXISTENCE-OF-EXPLANATION W.R.T. $\mathcal{O}_{\mathbf{S}}$.

Definition 6.5.7 (COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$)**.** COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ *is the following computational problem: given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$, find a most-general explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$ for $\overline{a} \notin Ans$, where $\mathcal{O}_{\mathbf{S}}$ is the $\mathbf{S}$-ontology that is derived from $\mathbf{S}$, as defined in Section 6.4.2.*

The complexity of COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ depends on the complexity of subsumption checking for $L_{\mathbf{S}}$. As can be seen in Table 6.1, subsumption checking with respect to arbitrary integrity constraints is undecidable. Therefore, for the general case in which no restriction is imposed on the integrity constraints, COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ is unlikely to be decidable. The restrictions on the integrity constraints of $\mathbf{S}$ allow for the definition of several variants of the problem that, under some restrictions, are decidable.

We restrict now to the cases in which we are able to materialize the $\mathbf{S}$-ontology $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$, with $\mathcal{K} = \text{adom}(I) \cup \{a_1, \ldots, a_m\}$. EXHAUSTIVE SEARCH ALGORITHM gives us a method for solving COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$. The following proposition gives us a double exponential upper bound for COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ in the general case, and a polynomial case under specific assumptions (cf. Table 6.1).

Proposition 6.5.3. *There is an algorithm that solves* COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$

- *in* 2EXPTIME *for* $L_{\mathbf{S}}$*, provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* EXPTIME *for selection-free* $L_{\mathbf{S}}$*, and projection-free* $L_{\mathbf{S}}$*, provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* PTIME *for* $L_{\mathbf{S}}^{\min}$*, if the arity of* q *is fixed and provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* PTIME.

We end with the definition of CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$.

Definition 6.5.8. *The* CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ *problem is the following decision problem: given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and a tuple of concepts* $E = (C_1, \ldots, C_n)$*, is* E *a most-general explanation w.r.t.* $\mathcal{O}_{\mathbf{S}}$ *for* $\overline{a} \notin Ans$*?*

As for COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$, the undecidability of concept subsumption in the general case suggests that it is unlikely for CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ to be decidable without imposing any restriction on Π and Σ. However, also this problem allows for the characterization of several decidable variants.

In particular, since CHECK-MGE is solvable in PTIME (see Theorem 6.5.1), by materializing $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ we can derive some upper bounds for CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ too.

Proposition 6.5.4. *There is an algorithm that solves* CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$

- *in* 2EXPTIME *for* $L_{\mathbf{S}}$ *concepts, provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* EXPTIME *for selection-free* $L_{\mathbf{S}}$*, and projection-free* $L_{\mathbf{S}}$*, provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* PTIME *for* $L_{\mathbf{S}}^{\min}$*, provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* PTIME.

The proof is analogous to the one for Proposition 6.5.3.

We expect that the upper bounds for COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ and CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ can be improved. Pinpointing the complexity of these problems is left for future work.

6.6 Variations of the framework

We consider several refinements and variations to our framework involving finding short explanations, and providing alternative definitions of *explanations* and of what it means to be *most general*.

Producing a Short Explanation. A most general explanation may contain redundant concepts as conjuncts. An explanation that is equivalent but *shorter* may be more helpful to the user. To simplify our discussion, we restrict our attention to ontologies that are derived from an instance and show that the problem of finding a most-general explanation of minimal length is NP-hard in general, where the *length* of an explanation $E = (C_1, \ldots, C_k)$ is measured by the total number of symbols needed to write out C_1, $\ldots$, C_k.

Proposition 6.6.1. *Given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$*, the problem of finding a most-general explanation to* $\bar{a} \notin Ans$ *of minimal length is* NP*-hard.*

Given that computing a shortest most-general explanation is intractable in general, we may consider the task of shortening a given most-general explanation. The INCREMENTAL SEARCH ALGORITHM produces concepts that may contain superfluous conjuncts. It is thus natural to ask whether the algorithm can be modified to produce a most-general explanation of a shorter length. This question can be formalized in at least two ways.

Let I be an instance of a schema $\mathbf{S}$, and let $C = \sqcap\{C_1, \ldots, C_n\}$ be any $L_{\mathbf{S}}$ concept expression. We may assume that each C_i is intersection-free. We say that C is *irredundant* if there is a no strict subset $X \subsetneq \{C_1, \ldots, C_n\}$ such that $C \equiv_{\mathcal{O}_I} \sqcap X$. We say that an explanation (with respect to $\mathcal{O}_I$) is irredundant if it consists of irredundant concept expressions. We say that explanations E_1 and E_2 are *equivalent* w.r.t. an ontology $\mathcal{O}$, denoted as $E_1 \equiv_{\mathcal{O}} E_2$, if $E_1 \leq_{\mathcal{O}} E_2$ and $E_2 \leq_{\mathcal{O}} E_1$.

Proposition 6.6.2. *There is a polynomial-time algorithm that takes as input an instance* I *of a schema* $\mathbf{S}$*, as well as an* $L_{\mathbf{S}}$ *concept expression* C*, and produces an irredundant concept expression* C' *such that* $C \equiv_{\mathcal{O}_I} C'$*.*

Hence, by combining Proposition 6.6.2 with INCREMENTAL SEARCH ALGORITHM, we can compute an irredundant most-general explanation w.r.t. $\mathcal{O}_I$ in polynomial time.

We say that an explanation $E = (C_1, \ldots, C_k)$ is *minimized* w.r.t. $\mathcal{O}_I$ if there does not exist an explanation $E' = (C_1, \ldots, C_k)$ such that $E \equiv_{\mathcal{O}_I} E'$ and E' is shorter than E. Every minimized explanation is irredundant, but the converse may not be true. For instance, let O be an ontology with three atomic concepts C_1, C_2, C_3 such that $C_1 \sqsubseteq_O C_2 \sqcap C_3$ and $C_2 \sqcap C_3 \sqsubseteq_O C_1$. Then the concept $C_2 \sqcap C_3$ is irredundant with respect to O. However, C_1 is an equivalent concept of strictly shorter length.

Proposition 6.6.3. *Given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an explanation* E *to why* $\bar{a} \notin Ans$*, the problem of finding a minimized explanation equivalent to* E *is* NP*-hard.*

Cardinality based preference. We have currently defined a *most-general explanation* to be an explanation E such that there is no explanation E' with $E' >_{\mathcal{O}} E$. A natural alternative is to define "most general" in terms of the cardinality of the extensions of the concepts in an explanation. Formally, let $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ be an $\mathbf{S}$-ontology, and I an instance. We define the *degree of generality* of an explanation $E = (C_1, \ldots, C_m)$ with respect to $\mathcal{O}$ and I to be the (possibly infinite) sum $|ext(C_1, I)| + \cdots + |ext(C_m, I)|$. For two explanations, E_1, E_2, we write $E_1 >^{card}_{\mathcal{O},I} E_2$, if E_1 has a strictly higher degree of generality than E_2 with respect to $\mathcal{O}$ and I. We say that an explanation E is $>^{card}$-maximal (with respect to $\mathcal{O}$ and I) if there is no explanation E' such that $E' >^{card}_{\mathcal{O},I} E$.

Proposition 6.6.4. *Assuming P≠NP, there is no* PTIME *algorithm that takes as input a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an* $\mathbf{S}$*-ontology* $\mathcal{O}$*, and produces a* $>^{card}$*-maximal explanation for* $\overline{a} \notin Ans$*. This holds even for unary queries.*

In particular, this shows (assuming P≠NP) that computing $>^{card}$-maximal explanations is harder than computing most-general explanations. The proof of Proposition 6.6.4 goes by reduction from a suitable variant of SET COVER. Our reduction is in fact an L-reduction, which implies that there is no PTIME constant-factor approximation algorithm for the problem of finding a $>^{card}$-maximal explanation.

Strong explanations. We now examine an alternative notion of an explanation that is essentially independent to the instance of a why-not question. Recall that the second condition of our current definition of an explanation $E = (C_1, \ldots, C_m)$ requires that $ext(C_1, I) \times \cdots \times ext(C_1, I)$ does not intersect with Ans, where I is the given instance. We could replace this condition by a stronger condition, namely that $ext(C_1, I') \times \cdots \times ext(C_1, I')$ does not intersect with $q(I')$, for *any* instance I' of the given schema that is consistent with the ontology $\mathcal{O}$. If this holds, we say that E is a *strong explanation*.

A strong explanation is also an explanation but not necessarily the other way round. When a strong explanation E for $\overline{a} \notin Ans$ exists, then, intuitively, the reason why $\overline{a}$ does not belong to Ans, is essentially independent from the specific instance I, and has to do with the ontology $\mathcal{O}$ and the query q. In the case where the ontology $\mathcal{O}$ is derived from a schema $\mathbf{S}$, a strong explanation may help one discover possible errors in the integrity constraints of $\mathbf{S}$, or in the query q. We leave the study of strong why-not explanations for future work.

6.7 Conclusion

We have presented a new framework for why-not explanations, which leverages concepts from an ontology to provide high-level and meaningful reasons for why a tuple is missing from the result of a query. Our focus in this chapter was on developing a principled framework, and on identifying the key algorithmic problems. The exact complexity of some problems raised in this chapter remains open. In addition, there are several directions for future work.

Recall that, in general, there may be multiple most-general explanations for $\overline{a} \notin q(I)$. While we have presented a polynomial time algorithm for computing a most-general explanation to a why-not question w.r.t. $\mathcal{O}_I$ for the case of selection-free $L_{\mathbf{S}}$, the most-general explanation that is returned by the algorithm may not always be the most helpful explanation. In future work, we plan to investigate whether there is a polynomial delay algorithm for enumerating all most-general explanations.

Although we only looked at *why-not explanations*, it will be natural to consider *why explanations* in the context of an ontology, and in particular, understand whether the notion of most-general explanations, suitably adapted, applies in this setting. In addition, Roy and Suciu (2014) recently initiated the study of what one could call "why so high" and "why so low" explanations for numerical queries (such as aggregate queries). Again, it would be interesting to see if our approach can help in identifying high-level such explanations.

We have focused on providing why-not explanations to missing tuples of queries that are posed against a database schema. However, our framework for answering the why-not question is general and could, in principle, be applied also to queries posed against the ontology in an OBDA setting.

Finally, we plan to explore ways whereby our high-level explanations can be used to complement and enhance existing data-centric and/or query-centric approaches. We illustrate this with an example. Suppose a certain publication X is missing from the answers to query over some publication database. A most-general explanation may be that X was published by Springer (supposing all Springer publications are missing from the answers to the query). This explanation provides insight on potential high-level issues that may exist in the database and/or query. For example, it may be that all Springer publications are missing from the database (perhaps due to errors in the integration/curation process) or the query has inadvertently omitted the retrieval of all Springer publications. This is in contrast with existing data-centric (resp. query-centric) approaches, which only suggest fixes to the database instance (resp. query) so that the specific publication X appears in the query result.

6.A Missing proofs for Section 4

6.A.1 Proofs for Section 6.4.1

Theorem 6.4.1. *[Calvanese et al. (2007); Poggi et al. (2008)] Let $\mathcal{T}$ be a DL-Lite$_\mathcal{R}$ TBox.*

(i) There is a PTIME*-algorithm for deciding subsumption. That is, given $\mathcal{T}$ and two concepts C_1, C_2, decide if $\mathcal{T} \models C_1 \sqsubseteq C_2$.*

(ii) There is an algorithm that, given an OBDA specification $\mathcal{B}$, an instance I over $\mathbf{S}$ *and a concept C, computes $certain(C, I, \mathcal{B}) = \bigcap\{\mathcal{I}(C) \mid \mathcal{I}$ is a solution for I w.r.t. $\mathcal{B}\}$. For a fixed OBDA specification, the algorithm runs in* PTIME *(*AC^0 *in data complexity).*

Proof. Item (i). The subsumption problem in *DL-Lite*$_\mathcal{R}$ is known to be in PTIME in the size of the TBox (Calvanese et al., 2007).

Item (ii). First, observe that *DL-Lite*$_\mathcal{R}$ is FO-rewritable (Calvanese et al., 2007), i.e., given a TBox $\mathcal{T}$ and an Abox $\mathcal{A}$, for every CQ q asked against $\mathcal{T} \cup \mathcal{A}$, there exists a query q' such that $certain(q, I, \mathcal{T} \cup \mathcal{A}) = q'(\mathcal{A})$, where q' is a UCQ called the perfect rewriting of q w.r.t. $\mathcal{T}$. Such q' is PTIME computable.

Then, let $\mathcal{B}$ be an OBDA specification. If we replace the ABox with $\mathcal{M} \cup I$, we have that for every query CQ q asked against $\mathcal{B}$, there exists a query q' such that $certain(q, I, \mathcal{B}) = q'(\mathcal{M} \cup I)$, where q' is as defined above. Then, let us denote by $unf_\mathcal{M}(q')$ the *unfolding* of q' w.r.t. $\mathcal{M}$, i.e., the UCQ query obtained from q' by substituting each atom from $\Phi_C \cup \Phi_R$ with the union of bodies of the mappings from $\mathcal{M}$ that have the atom as the head. Unfolding is also PTIME computable (Poggi et al., 2008). Then, computing $certain(q, I, \mathcal{B})$ amounts to evaluate $unf_\mathcal{M}(q')$ on I, which is in PTIME.

Finally, notice that computing $ext(C, I)$ requires one to evaluate the unary query $q(x) = C(x)$ over $\mathcal{B}$. This task, also known as *instance retrieval*, takes polynomial time in light of the above results. □

Theorem 6.4.2. *The* $\mathbf{S}$*-ontology $\mathcal{O}_\mathcal{B} = (\mathcal{C}_{\mathcal{O}_\mathcal{B}}, \sqsubseteq_{\mathcal{O}_\mathcal{B}}, ext_{\mathcal{O}_\mathcal{B}})$ can be computed from a given OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ in* PTIME *if $\mathcal{T}$ is a DL-Lite$_\mathcal{R}$ TBox and $\mathcal{M}$ is a set of GAV mappings.*

Proof. The size of $\mathcal{C}_{\mathcal{O}_\mathcal{B}}$ is polynomial in the size of $\mathcal{T}$. From Theorem 6.4.1, it follows that concept subsumption can be decided in PTIME for *DL-Lite*$_\mathcal{R}$ TBoxes, and also that computing the extension $ext(C, I)$ of each concept $C \in \mathcal{C}_{\mathcal{O}_\mathcal{B}}$ requires PTIME (AC^0 in data complexity). □

6.A.2 Proof of Theorem 6.4.3

In this section we give proofs for the results stated in Table 6.1.

Before turning to the proofs, we note that we only consider the subsumption problem for input concepts that do not contain nominals. Indeed, we can show that in this case the problem can be easily solved. First, however, we prove the following

Proposition 6.A.1. *Let* $\mathbf{S}$ *be a schema,* C_1 *a concept in* $L_\mathbf{S}$ *and* $C_2^i, 1 \leq i \leq n$ *concepts in intersection-free* $L_\mathbf{S}$*. Then*

$$C_1 \sqsubseteq_\mathbf{S} C_2^1 \sqcap \ldots \sqcap C_2^n \textit{ iff } C_1 \sqsubseteq_\mathbf{S} C_2^i \textit{ for every } i \leq n.$$

Proof. ($\Leftarrow$). Let I be an instance of $\mathbf{S}$, and $a \in [\![C_1]\!]^I$. Then for every $i, 1 \leq i \leq n$ it holds $a \in [\![C_2^i]\!]^I$. This means that $a \in [\![C_2^1 \sqcap \ldots \sqcap C_2^n]\!]^I$, as required.

($\Rightarrow$). Let I be an instance of $\mathbf{S}$, and $a \in [\![C_1]\!]^I$. Then it holds $a \in [\![C_2^1 \sqcap \ldots \sqcap C_2^n]\!]^I$, which means $a \in [\![C_2^i]\!]^I$ for every $i, 1 \leq i \leq n$. $\square$

Henceforth, this proposition allows restrict ourselves to the case of subsumption in an intersection-free concept $L_\mathbf{S}$. Let $C_1 \sqsubseteq_\mathbf{S} C_2$ be an instance of a subsumption problem, where $C_1 \in L_\mathbf{S}$ and C_2 is from intersection-free $L_\mathbf{S}$ and C_1 or C_2 contain a nominal. First, if C_1 contains a nominal as a conjunct, then in fact C_1 is equivalent to this nominal or inconsistent. In the latter case subsumption trivially holds. The following proposition shows how we deal with the remaining cases.

Proposition 6.A.2. *Let* $\mathbf{S}$ *be a schema. Then for every* $a, b \in \mathbf{Const}$ *the following hold*

(i) $\{a\} \not\sqsubseteq_\mathbf{S} \{b\}$, $\{a\} \sqsubseteq_\mathbf{S} \{a\}$,

(ii) $\{a\} \not\sqsubseteq_\mathbf{S} \pi_A(\sigma_\theta(R))$,

(iii) $\sqcap_{i=1}^n \pi_{A_i}(\sigma_{\theta_i}(R_i)) \sqsubseteq_\mathbf{S} \{a\}$ *iff for every* $i, 1 \leq i \leq n$ *it holds* $\theta_i \models A_i = a$.

Proof. (i) follows from the definitions. For (ii), we can pick the empty instance in which $R^I = \emptyset$, thus giving a counter example for subsumption. We show (iii). From the definition it follows that $\sqcap_{i=1}^n \pi_{A_i}(\sigma_{\theta_i}(R_i)) \sqsubseteq_\mathbf{S} \{a\}$ if and only if $[\![\pi_{A_i}(\sigma_{\theta_i}(R_i))]\!]^I = a$ for every $i \in \{1, \ldots, n\}$ and every instance I of $\mathbf{S}$. For every $i \in \{1, \ldots, n\}$, if it holds that $\theta_i \models A_i = a$, then clearly $[\![\pi_{A_i}(\sigma_{\theta_i}(R_i))]\!]^I = a$ for every instance I of $\mathbf{S}$. For the opposite direction, suppose there is a j such that $\theta_j \not\models A_j = a$. Then let J be an instance of $\mathbf{S}$ such that $\bar{b} \in R_j^J$, $\bar{b}$ satisfies θ_j and $\pi_{A_j}(\bar{b}) \neq a$. Then $[\![\pi_{A_j}(\sigma_{\theta_j}(R_j))]\!]^J$ contains $\pi_{A_j}(\bar{b})$ which is not equal to a, a contradiction. $\square$

Thus from now on, we only deal with the subsumption problem for concepts that do not contain nominals.

Schema with UCQ-view definitions

In this section we assume a schema $(\mathbf{S}, \Sigma)$, where Σ is a collection of (nested) UCQ-view definitions. We first consider the simplest fragments of $L_\mathbf{S}$.

Selection-free fragments of $L_\mathbf{S}$

The complexity results for subsumption come from the known results for the containment problem of Datalog queries. For the semantics of Datalog we refer to Chapter 2. Let Π_1 and Π_2 be Datalog queries defined over the same set of extensional predicates EDB. We say that a Datalog query Π_1 is *contained* in a Datalog query Π_2, denoted as $\Pi_1 \subseteq \Pi_2$, if it holds $G_1^{\Pi_1(I)} \subseteq G_2^{\Pi_2(I)}$ for every instance I over EDB, where G_i is the goal predicate of Π_i.

Claim 6.A.1. *The containment problem for Datalog and the subsumption problem in $L_{\mathbf{S}}^{\min}$ or selection-free $L_{\mathbf{S}}$ with respect to* $\mathbf{S}$ *are* PTIME*-equivalent.*

Proof. (sketch) Let Σ be a collection of nested UCQ-view definitions without comparisons. We may see it as a non-recursive Datalog program as was noted in Preliminaries.

- Reduction from the subsumption problem to the containment problem. We show the case of selection-free $L_{\mathbf{S}}$, then the case of $L_{\mathbf{S}}^{\min}$ follow. Let $C_1 = \sqcap_{i=1}^{n_1} \pi_{A_i^1}(R_i^1)$ and $C_2 = \sqcap_{j=1}^{n_2} \pi_{A_j^2}(R_j^2)$ be from selection-free $L_{\mathbf{S}}$, and $C_1 \sqsubseteq_{\mathbf{S}} C_2$ an instance of the subsumption problem. We then define the Datalog query $\Pi_i, i = 1, 2$, as the program Σ together with the additional rule $C_i(x)$:- $R_1^i(\bar{x}_1), \ldots, R_{n_i}^i(\bar{x}_{n_i})$, where x is in the attribute A_j^i of $\bar{x}_j$, and C_i as the goal predicate. Then it can be shown $C_1 \sqsubseteq_{\mathbf{S}} C_2$ if and only if $\Pi_1 \subseteq \Pi_2$.

- Reduction from the containment problem to the subsumption problem. We show the reduction for the case $L_{\mathbf{S}}^{\min}$, then the case of selection-free $L_{\mathbf{S}}$ would follow. Let $\Pi_1 \subseteq \Pi_2$ be an instance of Datalog containment problem, and EDB the common set of extensional predicates. Let G_1 and G_2 be the corresponding goal predicates. W.l.o.g. we can assume that both G_1 and G_2 are unary. Indeed, it can be shown that an n-ary ($n \geq 2$) containment can be reduced to a unary containment of Datalog queries. We define $C_i := \pi_1(G_i) = G_i$ and a schema $(\mathbf{S}, \Sigma)$, where $\mathbf{S} = EDB \cup IDB(\Pi_1) \cup IDB(\Pi_2)$ and $\Sigma = \Pi_1 \cup \Pi_2$ (written as nested UCQ-view definitions). Then it holds $\Pi_1 \subseteq \Pi_2$ if and only if $C_1 \sqsubseteq_{\mathbf{S}} C_2$. □

We make use of the following results that can be found in (Benedikt and Gottlob, 2010; Nutt, 2013; Shmueli, 1993).

Theorem 6.A.1. *The containment problem $\Pi_1 \subseteq \Pi_2$, where Π_1 and Π_2 are Datalog queries, is undecidable. It is decidable and* CONEXPTIME*-complete if both Π_1 and Π_2 are non-recursive. The hardness result holds even for a fixed input schema. Additionally it is*

1. *Π_2^P-complete if both Π_1 and Π_2 linear,*
2. NP*-complete if Π_1 and Π_2 are collections of* UCQ*-view definitions, and*
3. *Π_2^P-complete if Π_1 and Π_2 are collections of* UCQ*-view definitions with comparisons.*

In particular, item 3. follows from the results on containment of CQs with comparisons. In (Nutt, 2013) a Π_2^P lower bound was shown where CQs have comparisons of type $x \mathsf{op}\, c$, $\mathsf{op} \in \{\leq, >\}$. Then using the reduction in Claim 6.A.1 and Theorem 6.A.1 we have the following.

Corollary 6.A.1. *Let* $\mathbf{S}$ *be a schema with a collection of view definitions Σ as integrity constraints. The subsumption problem $C_1 \sqsubseteq_{\mathbf{S}} C_2$, where C_1 and C_2 are from $L_{\mathbf{S}}^{\min}$ or selection-free $L_{\mathbf{S}}$, is*

a) CONEXPTIME*-complete if Σ is a collection of nested* UCQ*-view definitions without comparisons,* *b)* Π_2^P*-complete if* Σ *is a collection of linearly nested* UCQ*-view definitions without comparisons,*

c) Π_2^P*-complete if* Σ *is a collection of* UCQ*-view definitions.*

d) NP*-complete if* Σ *is a collection of* UCQ*-view definitions without comparisons.*

In the next section we show that the upper bounds still hold even for the case of nested UCQ-view definitions *with* comparisons.

Full $L_{\mathbf{S}}$

In this section we consider the subsumption problem for the concept language $L_{\mathbf{S}}$ w.r.t. a schema with a collection of view definitions. For an upper bound, we reduce the subsumption problem to Datalog containment. Let $C_1 \sqsubseteq_{\mathbf{S}} C_2$ be an instance of the subsumption problem for $L_{\mathbf{S}}$.

We prove the following.

Proposition 6.A.3. *Let* $(\mathbf{S}, \Sigma)$ *be a schema with* Σ *a collection of nested* UCQ*-view definitions,* C_1 *and* C_2 *concepts in* $L_{\mathbf{S}}$*. Then there exist* PTIME *computable non-recursive Datalog queries* Π_1 *and* Π_2 *such that*

$$C_1 \sqsubseteq_{\mathbf{S}} C_2 \textit{ iff } \Pi_1 \subseteq \Pi_2.$$

Moreover, if Σ *is linearly nested, then* Π_1 *and* Π_2 *are linear non-recursive Datalog queries.*

Proof. Let $\mathbf{S}$, C_1 and C_2 be as in the statement. It is convenient to represent C_1 and C_2 as conjunctive queries. This is done via the standard translation of relational algebra expressions to first order logic. Assume that $C_1(x)$ and $C_2(x)$ are (unary) conjunctive queries with arithmetic comparisons corresponding to C_1 and C_2 respectively. First we note that the subsumption problem $C_1 \sqsubseteq_{\mathbf{S}} C_2$ is equivalent to the containment problem of two Datalog queries where the goal predicates are defined as $C_1(x)$ and $C_2(x)$ respectively. The latter containment can be reduced to Boolean containment of Datalog queries (in PTIME). Thus assume we have reduced our problem to Datalog containment $\Pi_1' \subseteq \Pi_2'$, where Π_i' has the form $\Pi \cup \{G_i\text{:-}Q_i\}$ with the Boolean goal predicates G_i. Note this reduction preserves the syntactic restrictions (i.e., non-recursiveness and linearity) on the datalog programs. Let $\mathbf{D}$ denote EDB predicates.

Let $\mathcal{K} \subseteq \mathbf{Const}$ be the set of constants appearing in Π_1' or Π_2'. For every constant $c \in \mathcal{K}$ we introduce new unary EDB predicates $P_{\mathsf{op}\ c}$, $\mathsf{op} \in \{<, =, >\}$. Additionally, we introduce a new unary EDB predicate $Dom(\cdot)$. Let $\mathbf{D}^{\mathsf{op}}$ be the set of the newly introduced unary predicates. Let $\widetilde{\Pi}_i'$, $i = 1, 2$, denote the result of the following operations on Π_i':

(i) Replace each occurrence of $x \mathsf{op}\, c$ with $P_{\mathsf{op}\ c}(x)$, $\mathsf{op} \in \{=, \leq, \geq, <, >\}$. The predicates $P_{\leq c}$ and $P_{\geq c}$ are IDB predicates defined below.

(ii) For every rule of Π_i' and every variable x in the rule, add $Dom(x)$ to the antecedent of the rule.

Then we define Π_i as follows.

$\Pi_1 = \tilde{\Pi}'_1 \cup \Pi^{\leq,\geq} \cup \Pi^{Dom}$ and $\Pi_2 = \tilde{\Pi}'_2 \cup \Pi^{\leq,\geq} \cup \Pi^{Dom} \cup \{G_2 \text{ :- } Bad\} \cup \Pi^{Bad}$. The program $\Pi^{\leq,\geq}$ is defined by the following rules, for every $c \in \mathcal{C}$:

$$\begin{aligned}
&P_{\leq c}(x) \text{ :- } P_{<c}(x),\\
&P_{\leq c}(x) \text{ :- } P_{=c}(x),\\
&P_{\geq c}(x) \text{ :- } P_{>c}(x),\\
&P_{\geq c}(x) \text{ :- } P_{=c}(x).
\end{aligned}$$

The program Π^{Dom} is defined by the following rules, for every $c \in \mathcal{C}$:

$$\begin{aligned}
&Dom(x) \text{ :- } P_{=c}(x),\\
&Dom(x) \text{ :- } P_{<c}(x),\\
&Dom(x) \text{ :- } P_{>c}(x).
\end{aligned}$$

Bad is a 0-ary IDB predicate defined by the program Π^{Bad}, which lists all possible inconsistencies that we try to avoid. For every $c, c_1, c_2 \in \mathcal{K}$ such that $c_1 < c_2$ and $\mathtt{op}_1, \mathtt{op}_2 \in \{<, =, >\}$ with $\mathtt{op}_1 \neq \mathtt{op}_2$, Π^{Bad} contains:

$$\begin{aligned}
&Bad \text{ :- } P_{\mathtt{op}_1 c}(x), P_{\mathtt{op}_2 c}(x),\\
&Bad \text{ :- } P_{=c_1}(x), P_{=c_2}(x),\\
&Bad \text{ :- } P_{<c_1}(x), P_{>c_2}(x),\\
&Bad \text{ :- } P_{=c_1}(x), P_{>c_2}(x),\\
&Bad \text{ :- } P_{<c_1}(x), P_{=c_2}(x).
\end{aligned}$$

Additionally, if $(\mathbf{Const}, <)$ has the left endpoint c_l and/or the right end point c_r , then Π^{Bad} also contains the rules

$$\begin{aligned}
&Bad \text{ :- } P_{<c_l}(x),\\
&Bad \text{ :- } P_{>c_r}(x).
\end{aligned}$$

Moreover, for every c_1 and c_2 from $\mathcal{K}$ such that $c_1 < c_2$ and there is no $c' \in \mathbf{Const}$ with $c_1 < c' < c_2$, Π^{Bad} contains:

$$Bad \text{ :- } P_{>c_1}(x), P_{<c_2}(x).$$

Clearly Π_1 and Π_2 are PTIME computable and satisfy the syntactic restrictions (i.e., non-recursiveness and linearity). We claim that $\Pi'_1 \subseteq \Pi'_2$ iff $\Pi_1 \subseteq \Pi_2$.

($\Leftarrow$). Suppose there is an instance I' over $\mathbf{D}$ such that $\Pi'_1(I') \models G_1$ and $\Pi'_2(I') \not\models G_2$. We then define an instance I over $\mathbf{D} \cup \mathbf{D}^{\mathsf{op}}$ as the instance with the same active domain as I. Moreover, the interpretation of EDB predicates is defined as follows.

- $R(\bar{a}) \in I$ iff $R(\bar{a}) \in I'$, for every $R \in \mathbf{D}$,
- $P_{\mathtt{op}\ c}(a) \in I$ iff $a \in \mathrm{adom}(I)$ and $(\mathbf{Const}, <) \models a \mathtt{op}\, c$, where $\mathtt{op} \in \{=, <, >\}$,
- $Dom(a) \in I$ iff $a \in \mathrm{adom}(I)$.

It is straightforward to show that $\Pi_1(I) \models G_1$ and $\Pi_2(I) \not\models G_2$. In particular, to show the latter we use the fact that Bad is not satisfied in $\Pi_2(I)$ by definition of the interpretations of the predicates $P_{\mathsf{op}c}$. Thus, we have provided a counterexample for $\Pi_1 \subseteq \Pi_2$.

($\Rightarrow$). Suppose there exists an instance $\tilde{I}$ over $\mathbf{D} \cup \mathbf{D}^{\mathrm{op}}$ such that $\Pi_1(\tilde{I}) \models G_1$ and $\Pi_2(\tilde{I}) \not\models G_2$. We define a new instance I as the restriction of $\tilde{I}$ to Dom, i.e., the domain is defined as $\mathrm{adom}(I) = \{a \mid Dom(a) \in \tilde{I}\}$. Note that the extension of Dom in $\Pi_1(I)$ and $\Pi_2(I)$ are the same and not empty. Note also that I is still a counterexample for the containment $\Pi_1 \subseteq \Pi_2$, i.e., $\Pi_1(I) \models G_1$ and $\Pi_2(I) \not\models G_2$. We prove the following.

Claim 6.A.2. *For every $a \in \mathrm{adom}(I)$ there exists $c \in \mathbf{Const}$ such that for every $c' \in \mathcal{K}$ and $\mathsf{op} \in \{<, =, >\}$ it holds*

$$I \models P_{\mathsf{op}\ c'}(a) \ \textit{ iff } \ (\mathbf{Const}, <) \models c\mathsf{op}\ c'.$$

Proof. By definition of I, it holds $Dom(a) \in I$ for every $a \in \mathrm{adom}(I)$. This means that either $P_{=c}(a)$ or $P_{>c}(a)$ or $P_{<c}(a)$ holds in I for every $c \in \mathcal{K}$. In case $P_{=c}(a)$ holds in I, there is no other fact $P_{=c'}(a)$ in I for some $c' \neq c$ because of Bad. Then using Bad we can show that $I \models P_{\mathsf{op}\ c'}(a)$ iff $(\mathbf{Const}, <) \models c\mathsf{op}\ c'$. Thus, c is as needed. Suppose there is no c such that $I \models P_{=c}(a)$. Let c_1 be the maximal element of $\mathcal{K}$ such that $P_{>c_1}(a) \in I$ and c_2 the minimal element of $\mathcal{K}$ such that $P_{<c_2}(a) \in I$. In case $(\mathbf{Const}, <)$ has endpoints, we have that $c_l \leq c_1$ and $c_2 \leq c_r$ due to Bad. Furthermore, we have that c_2 is not the immediate successor of c_1 by Bad. Thus there exists $c \in \mathbf{Const} \setminus \mathcal{K}$ such that $c_1 < c < c_2$. Moreover, by the choice of c for every $c' \in \mathcal{K}$ it holds $I \models P_{\mathsf{op}\ c'}(a)$ iff $(\mathbf{Const}, <) \models c\mathsf{op}\ c'$. □

We then define an instance I' as the homomorphc image $f(I)$, where f is such that for every $a \in \mathrm{adom}(I)$ we define $f(a) = c$, where c is from Claim 6.A.2.

We need to show that (A) $\Pi'_1(I') \models G_1$ and (B) $\Pi'_2(I') \not\models G_2$.

This follows from the next claim. For a CQ with arithmetic comparisons $Q(\overline{x})$, by $\tilde{Q}(\overline{x})$ we denote the result of replacing each comparison $x\mathsf{op}\ c$ with the atom $P_{\mathsf{op}\ c}(x)$ and adding the conjunct $Dom(x)$.

Claim 6.A.3. *Let $Q(\overline{x})$ be a CQ with arithmetic comparisons over $\mathbf{D} \cup IDB(\Pi)$. Then*

$$\tilde{\Pi} \cup \Pi^{\leq,\geq} \cup \Pi^{Dom}(I) \models \tilde{Q}(\bar{a}) \textit{ iff } \Pi(I') \models Q(f(\overline{a})).$$

Proof. The proof is by the induction on the depth of the datalog program. Base of induction: EDB predicates and arithmetic comparisons.

- Let $Q(x) = x\mathsf{op}\ c$. Then $\tilde{\Pi} \cup \Pi^{\leq,\geq} \cup \Pi^{Dom}(I) \models P_{\mathsf{op}\ c}(a) \wedge Dom(a)$ iff $I \models P_{\mathsf{op}\ c}(a)$ iff (by definition of I') $I' \models f(a)\mathsf{op}\ c$ iff $\Pi(I') \models f(a)\mathsf{op}\ c$. The cases $\mathsf{op} \in \{\leq, \geq\}$ can be treated similarly, though the predicates $P_{\leq c}$ and $P_{\geq c}$ are IDBs.

- $Q(\overline{x}) = R(\overline{x})$ for an EDB predicate R. Then $\tilde{\Pi} \cup \Pi^{\leq,\geq} \cup \Pi^{Dom}(I) \models R(\overline{a}) \wedge Dom(a)$ iff $I \models R(\overline{a})$ iff (by definition of I') $I' \models R(f(\overline{a}))$ iff $\Pi(I') \models R(f(\overline{a}))$.

Step of induction. Let $R(\overline{x})$:- $R_1^i(\overline{x},\overline{y}),\ldots,R_k^i(\overline{x},\overline{y}),Dom(\bar{x},\bar{y})$ be the rules in $\tilde{\Pi}$ defining $R(\overline{x})$, where each R_i is a predicate over $\mathbf{D}\cup\mathbf{D}^{\text{op}}\cup\text{IDB}(\Pi)$. Then $\tilde{\Pi}\cup\Pi^{\leq,\geq}\cup\Pi^{Dom}(I)\models R(\overline{a})$ iff $\tilde{\Pi}\cup\Pi^{\leq,\geq}\cup\Pi^{Dom}(I)\models R_1^i(\overline{a},\overline{b})\wedge\ldots\wedge R_k^i(\overline{a},\overline{b})$ for some i and tuple $\overline{b}$ from $\text{adom}(I)$. By the induction hypothesis, the latter is equivalent to $\Pi(I')\models R_1^i(f(\overline{a}),f(\overline{b}))\wedge\ldots\wedge R_k^i(f(\overline{a}),f(\overline{b}))$ (note if R_j^i was from $\mathbf{D}^{\text{op}}$, then it is replaced by the corresponding arithmetic comparison), which is equivalent to $\Pi(I')\models R(f(\overline{a}))$. □

We show how (A) and (B) follow from this claim. Recall $\Pi_1(I)\models G_1$, which is the same as $\tilde{\Pi}\cup\Pi^{\leq,\geq}\cup\Pi^{Dom}(I)\models\tilde{Q}_1$ for Q_1 being the body of the rule defining G_1. By Claim 6.A.3 it follows $\Pi(I')\models Q_1$, and thus $\Pi_1'(I')\models G_1$. Similarly we can show (B). Indeed, suppose the opposite, i.e. $\Pi_2'(I')\models G_2$. This is equivalent to $\Pi(I')\models Q_2$ for Q_2 being the definition of G_2. By Claim 6.A.3 it follows that $\tilde{\Pi}\cup\Pi^{\leq,\geq}\cup\Pi^{Dom}(I)\models\tilde{Q}_2$. This implies that $\tilde{\Pi}_2'\cup\Pi^{\leq,\geq}\cup\Pi^{Dom}(I)\models G_2$, which in turn implies $\Pi_2(I)\models G_2$, which is a contradiction. □

Thus from Theorem 6.A.1 and Proposition 6.A.3 we obtain:

Corollary 6.A.2. *Let* $\mathbf{S}$ *be a schema with a collection of view definitions* Σ *as integrity constraints. The subsumption problem* $C_1\sqsubseteq_{\mathbf{S}}C_2$*, where* C_1 *and* C_2 *are from* $L_{\mathbf{S}}$*, is*

a) In CONEXPTIME *if* Σ *is a collection of nested* UCQ*-view definitions,*

b) In Π_2^P *if* Σ *is a collection of linearly nested* UCQ*-view definitions.*

Schema with functional and inclusion dependencies

In this section we assume that the schema $\mathbf{S}$ has functional and inclusion dependencies as integrity constraints. We first show that the subsumption problem w.r.t. $\mathbf{S}$ is undecidable. We prove it by reduction from the implication problem for constraints.

Definition 6.A.2. *Let* IMPL$(\mathcal{C},\mathcal{D})$ *be the following problem: given a set of constraints* Σ *belonging to the class* $\mathcal{C}$ *and a single constraint* σ *belonging to the class* $\mathcal{D}$*, decide whether* $\Sigma\models\sigma$.

Theorem 6.A.3 (Mitchell (1983), Chandra and Vardi (1985))**.** IMPL$(FDs+IDs,IDs)$ *is undecidable.*

Note that the undecidability result already holds in case the implied constraint σ is *unary* inclusion dependency. We show the following.

Claim 6.A.4. *The concept subsumption problem for* $L_{\mathbf{S}}^{\min}$ *w.r.t.* $\mathbf{S}$ *is a special case of* IMPL$(\mathcal{C},UIDs)$ *under the same set of constraints* Σ *(where the set* Σ *belongs to the class* $\mathcal{C}$*).*

Proof. Since $C_1,C_2\in L_{\mathbf{S}}^{\min}$, they are of the form $\pi_A(R)$, i.e., unary projections over the relations of $\mathbf{S}$. Let $C_1=\pi_A(R)$ and $C_2=\pi_B(S)$, where R,S are two relation names and A and B are, respectively, attributes defined over R and S in $\mathbf{S}$. Let σ_C be the unary inclusion dependency (UID) $R[A]\subseteq S[B]$. Then $C_1\sqsubseteq_{\mathbf{S}}C_2$ iff $\Sigma\models\sigma_c$. □

Corollary 6.A.3. *The concept subsumption problem for* $L_{\mathbf{S}}^{\min}$ *w.r.t.* $(\mathbf{S},\Sigma)$*, where* Σ *is a set of inclusion and functional dependencies is undecidable.*

Subsumption for Σ a set of Inclusion Dependencies

We begin with the concept subsumption problem for $L_{\mathbf{S}}^{\min}$ w.r.t. schema with a set of inclusion dependencies Σ . We assume familiarity with the chase procedure for inclusion dependencies (Abiteboul et al., 1995).

Let Σ be a set of inclusion dependencies. We define a *position graph* G_Σ as follows.

Definition 6.A.4. *Let Σ be a set of inclusion dependencies. The position graph G_Σ of Σ is a tuple (V, E) such that $V = \{R[i] \mid R \in \mathbf{S}, i \text{ an attribute of } R\}$ and $E = \{\langle R[i], Q[j]\rangle \mid (R[\dots,i,\dots] \subseteq Q[\dots,j,\dots]) \in \Sigma\}$.*

Proposition 6.A.4. *Let Σ be a set of inclusion dependencies, σ a unary inclusion dependency $R[i] \subseteq Q[j]$. Then*

$$\Sigma \models \sigma \textit{ iff } Q[j] \textit{ is reachable from } R[i] \textit{ in } G_\Sigma.$$

Proof. ($\Leftarrow$). Suppose there is a path from $R[i]$ to $Q[j]$ in G_Σ. Each edge $\langle R_1[A], R_2[B]\rangle$ of the path corresponds to the inclusion dependency $R_1[\dots,A,\dots] \subseteq R_2[\dots,B,\dots]$ in Σ. Thus, there is a sequence of inclusion dependencies in Σ which derives σ.

($\Rightarrow$). Suppose $\Sigma \models R[i] \subseteq Q[j]$. Let $R(\overline{a})$ be a fact. Then it must hold that a fact $Q(\overline{b})$ with $b_j = a_i$ is contained in the result of chasing $\{R(\overline{a})\}$ with Σ. By induction on the length L of the chase on $R(\overline{a})$ we show that a_i is passed to reachable nodes only, i.e. if $Q(\dots,a_i,\dots) \in chase_\Sigma(\{R(\overline{a})\})$, then there is a path from $R[i]$ to $Q[j]$, where j is the position in Q where the value a_i is.

$L = 0$. Trivially true since $R[i]$ is reachable from itself.

$L = k + 1$. Assume $Q(\overline{b})$, where $b_l = a_i$ for some l, is obtained by application of $\alpha = (P[\overline{A}] \subseteq Q[\overline{B}])$ to $chase^k_\Sigma(R(\overline{a}))$. This means that there is a fact $P(\overline{c})$ in $chase^k_\Sigma(R(\overline{a}))$ such that $c_j = a_i$ for some $j \in \overline{A}$. Hence, $P[j]$ is reachable from $R[i]$, by the induction hypothesis. Since also there is an edge from $P[j]$ to $Q[l]$, there is a path from $R[i]$ to $Q[l]$ as required. $\square$

Thus we can reduce the subsumption problem for $L_{\mathbf{S}}^{\min}$ w.r.t. $(\mathbf{S}, \Sigma)$, where Σ is a set of IDs, to graph reachability. Recall that graph reachability can be decided in PTIME.

Corollary 6.A.4. *The concept subsumption problem for $L_{\mathbf{S}}^{\min}$ with respect to $(\mathbf{S}, \Sigma)$, where Σ is a set of IDs, can be solved in* PTIME.

We then show that the subsumption problem for selection-free $L_{\mathbf{S}}$ w.r.t. $(\mathbf{S}, \Sigma)$, where Σ is a set of IDs, can be reduced (in PTIME) to the one for $L_{\mathbf{S}}^{\min}$. This follows from the next two lemmas.

Lemma 6.A.1. *Let $\mathbf{S}$ be a schema, C_1 a concept in selection-free $L_{\mathbf{S}}$, $C_2^j, j \leq n$ concepts in $L_{\mathbf{S}}^{\min}$. Then*

$$C_1 \sqsubseteq_{\mathbf{S}} C_2^1 \sqcap \dots \sqcap C_2^n \textit{ iff } C_1 \sqsubseteq_{\mathbf{S}} C_2^j \textit{ for every } j \leq n.$$

Proof. ($\Rightarrow$). Let I be an instance over $\mathbf{S}$ and $a \in [\![C_1]\!]^I$. Then by the assumption $a \in [\![C_2^1 \sqcap \dots \sqcap C_2^n]\!]^I$. The latter means that $a \in [\![C_2^j]\!]^I$ for every $j, 1 \leq j \leq n$, as needed.

($\Leftarrow$). Let I be an instance over $\mathbf{S}$ and $a \in [\![C_1]\!]^I$. By the assumption, $a \in [\![C_2^j]\!]^I$ for every $j, 1 \leq j \leq n$, i.e., $a \in [\![C_2^1 \sqcap \dots \sqcap C_2^n]\!]^I$, as needed. $\square$

Lemma 6.A.2. *Let Σ be a set of IDs, and $(\mathbf{S}, \Sigma)$ a schema. Then for concepts C, $C_1, \ldots, C_n \in L^{\min}_{\mathbf{S}}$ it holds,*

$$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq_{\mathbf{S}} C \textit{ iff } C_i \sqsubseteq_{\mathbf{S}} C \textit{ for some } i \leq n.$$

Proof. One direction ($\Leftarrow$) is trivial. We show ($\Rightarrow$). Suppose $C_i \not\sqsubseteq_{\mathbf{S}} C$ for every $i \leq n$. There exist instances $I_i, i \leq n$ over $\mathbf{S}$ such that each of I_i is a counter-example for subsumption $C_i \not\sqsubseteq_{\mathbf{S}} C$. We can assume that a constant a witnesses every non-subsumption, i.e., $a \in [\![C_i]\!]^{I_i}$ and $a \notin [\![C]\!]^{I_i}$. Let I be the union of instances I_i. Note that I satisfies Σ. Indeed, let $\sigma = (R_1[\bar{A}] \subseteq R_2[\bar{B}])$ be an inclusion dependency from Σ. Let for $\bar{a}$ it holds $R_1(\bar{a}) \in I$. This means that this fact holds in one of I_i. Since each I_i satisfies σ, the fact $R_2(\bar{b})$ with $\bar{a}|_{\bar{A}} = \bar{b}|_{\bar{B}}$ holds in I_i. But this also means that this fact holds in the union I. Thus, σ is satisfied in I. Moreover, I is a counter-example for the subsumption $C_1 \sqcap \ldots \sqcap C_n \sqsubseteq_{\mathbf{S}} C$, since $a \in [\![C_1 \sqcap \ldots \sqcap C_n]\!]^I$ and $a \notin [\![C]\!]^I$. □

Subsumption for Σ a set of Functional Dependencies

We now consider the concept subsumption problem w.r.t. a schema $(\mathbf{S}, \Sigma)$, where Σ is a set of function dependencies.

We show that the subsumption problem is in PTIME.

Proposition 6.A.5. *The concept subsumption problem for $L_{\mathbf{S}}$ w.r.t $(\mathbf{S}, \Sigma)$, where Σ is a set of functional dependencies, is solvable in* PTIME.

Proof. Recall that an order $(V, <)$ is dense if for every x and y in V with $x < y$, there exists $z \in V$ such that $x < z$ and $z < y$.

Let C_1 and C_2 be $L_{\mathbf{S}}$ concepts. It is convenient to consider them as unary CQ queries with comparisons. For a CQ Q by $Var(Q)$ we denote the set of variables of Q.

Without loss of generality we may assume that CQs with comparisons are in normal form. We say that a CQ with a set of comparisons θ is in *normal form* if for every variable x it holds that

- Either θ contains at most one equality of x with a constant. Moreover, if such an equality occurs in θ, then θ contains no other comparisons of x.
- Or θ contains at most one interval or semi-interval for x. That is, a semi-interval $x \leq c$, $x \geq c$, $x < c$, or $x > c$, or (open or closed) interval $c_1 \mathtt{op}_1\ x\ \mathtt{op}_2\ c_2$, where $\mathtt{op}_1, \mathtt{op}_2 \in \{<, \leq\}$. In case $(\mathbf{Const}, <)$ contains a left endpoint, then θ contains at most one right semi-interval $x \geq c$ or $x > c$, or an interval for x. Similarly, if $(\mathbf{Const}, <)$ contains a right endpoint, then θ contains at most one left semi-interval $x \geq c$ or $x > c$, or an interval for x.

Note that each CQ with comparisons can be converted (in PTIME) into an equivalent CQ with comparisons in normal form.

A *generalized instance* over $\mathbf{S}$ and a set of variables Var ($Var \cap \mathbf{Const} = \emptyset$) is a set of relational facts of the form $R(x_1, \ldots, x_n)$ and comparisons $x \mathtt{op} c$, where $R \in \mathbf{S}, x_1, \ldots, x_n, x \in Var$, $\mathtt{op} \in \{=, <, >, \leq, \geq\}$ and $c \in \mathbf{Const}$, with the condition that x must occur in some relational fact of I. We may assume that every consistent generalized instance is in normal form, as defined above for queries with comparisons.

For a generalized instance I and a conjunctive query Q, by $I \models Q$ we denote the fact that there is a homomorphism h from $Var(Q)$, variables of Q, to Var that preserves relational facts and arithmetic comparisons, that is, for a variable $x \in Var(Q)$, the comparisons of $h(x)$ must imply the comparisons $\theta(x)$ of x. By *the canonical generalized instance* I_Q of Q we mean the generalized instance consisting of the facts of Q, i.e., the set $\{R(\bar{x}) \mid R(\bar{x}) \text{ is a conjunct of } Q\} \cup \{x\mathsf{op}c \mid x\mathsf{op}c \text{ is a conjunct of } Q\}$. We may assume without loss of generality that relational facts do not contain constants since we can replace each constant by a fresh variable and equating it with the constant. For the algorithm, we use the chase technique adapted to deal with comparisons. Without loss of generality we may assume that every FD has one attribute as the consequent.
Chase step. Let I be a generalized instance over Var. We assume there is a total order $\leq_v$ on Var. Let σ be an FD $R:\ X \to B$. Let $R(\bar{x})$ and $R(\bar{y})$ be facts in I such that $\pi_A(\bar{x}) = \pi_A(\bar{y})$ for every $A \in X$ and $\pi_B(\bar{x}) \neq \pi_B(\bar{y})$. By x and y we denote the variables $\pi_B(\bar{x})$ and $\pi_B(\bar{y})$, respectively. Let $\theta(x)$ and $\theta(y)$ be the comparisons of x and y in I. Then the *result of applying the FD to I* is defined as follows, depending on the type of comparisons of x and y. We consider all the possible cases for the shape of comparisons of x and y and depending on the cases either obtain "failure" or a variable z and the constraint $\theta(z)$. Here z is $\min_{\leq_v}\{x, y\}$ and $w = \max_{\leq_v}\{x, y\}$. Then the result of applying FD is the instance I' obtained from I by substituting every occurrence of w with z. Additionally, the constraint $\theta(z)$ is added to I' and the old comparisons $\theta(x)$ and $\theta(y)$ are discarded.

- $\theta(x)$ is $x = c_1$ and $\theta(y)$ is $y = c_2$. Then "failure".
- $\theta(x)$ is $x = c_1$ and $\theta(y)$ is constraints involving inequality only. If c_1 does not satisfy $\theta(y)$, then "failure". Otherwise, let $\theta(z)$ be $z = c_1$.
- $\theta(x)$ is $c_x^1 \leq x \leq c_x^2$ and $\theta(y)$ is $c_y^1 \leq y \leq c_y^2$. If $[c_x^1, c_x^2] \cap [c_y^1, c_y^2] = \emptyset$, then "failure". Otherwise, let $\theta(z)$ be $c_1 \leq z \leq c_2$, where $c_1 = \max\{c_1^x, c_1^y\}$ and $c_2 = \min\{c_2^x, c_2^y\}$.
- $\theta(x)$ is $c_x^1 \leq x < c_x^2$ and $\theta(y)$ is $c_y^1 < y \leq c_y^2$. If $[c_x^1, c_x^2) \cap (c_y^1, c_y^2] = \emptyset$, then the result is "failure". Otherwise, let $\theta(z)$ be $c_1\mathsf{op}_1\ z\mathsf{op}_2\ c_2$, where $c_1 = \max\{c_1^x, c_1^y\}$, $c_2 = \min\{c_2^x, c_2^y\}$, and op_1 is "$\leq$" if c_1 is c_1^x and "$<$" otherwise, and op_2 is "$<$" if c_2 is c_2^x and "$\leq$" otherwise.
- $\theta(x)$ is $x\mathsf{op}_1 c_2^x$, where $\mathsf{op}_1 \in \{\leq, <\}$ and $\theta(y)$ is $y\mathsf{op}_2 c_1^y$, where $\mathsf{op}_2 \in \{\geq, >\}$. If $c_2^x < c_1^y$, then "failure". Otherwise, let $\theta(z)$ be $c_1^y\mathsf{op}_2^{-1}\ z\mathsf{op}_1\ c_2^x$.
- Other cases are similar.

By $I \to_\sigma I'$ we denote a chase step, where I' is the result of applying σ to I.
Chase sequence. Let Σ be a set of FDs, and let I be a generalized instance over Var. A *chase sequence* of I with Σ is a sequence of chase steps $I_i \to_\sigma I_{i+1}$, with $I_0 = I$ and $\sigma \in \Sigma$. A *finite chase of I with* Σ is a finite chase sequence of length m with the requirement that either $I_m =$ "failure" or there is no $\sigma \in \Sigma$ that can be applied to I_m. Then I_m is *the result of the finite chase of I with* Σ, denoted as $chase_\Sigma(I)$. The following facts can be proved similarly as for the classical chase for FDs.

Lemma 6.A.3. *Let I be a generalized instance and Σ a set of FDs. Then every chase sequence of I with Σ is a finite chase of I with Σ. Moreover, the length of the sequence is bounded by a polynomial in the size of I and Σ.*

Lemma 6.A.4. *Let I be a generalized instance and Σ a set of FDs. Let I_m be the result of a finite chase of I with Σ such that $I_m \neq$ "failure". Then I_m satisfies Σ. If there exists a finite chase of I with Σ with the result "failure", then there is no instance that satisfies Σ.*

Lemma 6.A.5. *Let $I_1 \rightarrow_\sigma I_2$ be a chase step, where $I_2 \neq$ "failure". Let I be a generalized instance such that (i) I satisfies σ and (ii) there exists a homomorphism $h_1 : I_1 \rightarrow I$. Then there exists a homomorphism $h_2 : I_2 \rightarrow I$.*

Our algorithm would proceed as follows. Let Q_i be the CQ with comparisons corresponding to C_i. Clearly, $C_1 \sqsubseteq_{\mathbf{S}} C_2$ iff Q_1 is contained in Q_2 w.r.t. Σ. The latter can be reduced to containment of Boolean queries and thus w.l.o.g. we can assume that Q_1 and Q_2 are Boolean. Note if Q_1 or Q_2 are inconsistent (which can be checked in PTIME), then the subsumption problem is trivial. Let I_{Q_1} be the canonical generalized instance of Q_1. Let $chase_\Sigma(I_{Q_1})$ be the result of the finite chase of I_{Q_1} with Σ. It is computable in PTIME by Lemma 6.A.3. Then we claim that Q_1 is contained in Q_2 if and only if $chase_\Sigma(I_{Q_1}) \models Q_2$.

Claim 6.A.5. *Let Q_1 and Q_2 be CQs with comparisons, Σ a set of FDs. Let $chase_\Sigma(I_{Q_1})$ be the result of the finite chase of I_{Q_1} with Σ. Then $Q_1 \subseteq_\Sigma Q_2$ iff $chase_\Sigma(I_{Q_1}) \models Q_2$.*

Proof. ($\Leftarrow$). Assume $chase_\Sigma(I_{Q_1}) \models Q_2$. If $chase_\Sigma(I_{Q_1}) =$ "failure", then by Lemma 6.A.4 there is no instance that satisfies Σ. Thus the containment is vacuously true.

Assume $chase_\Sigma(I_{Q_1}) \neq$"failure". Let I be an arbitrary instance such that $I \models \Sigma$ and $I \models Q_1$. This means there exists a homomorphism $h : Q_1 \rightarrow I$, which can be considered as a homomorphism from I_{Q_1} to I. Applying Lemma 6.A.5 to each chase step, we obtain that there exists a homomorphism $h' : chase_\Sigma(I_{Q_1}) \rightarrow I$. Since $chase_\Sigma(I_{Q_1}) \models Q_2$, there exists a homomorphism $g : Q_2 \rightarrow chase_\Sigma(I_{Q_1})$. Then it can be verified that $h' \circ g : Q_1 \rightarrow I$ is a homomorphism, thus $I \models Q_2$ as needed.

($\Rightarrow$). Assume $Q_1 \subseteq_\Sigma Q_2$. We have that $chase_\Sigma(I_{Q_1}) \models Q_1$ and $chase_\Sigma(I_{Q_1})$ satisfies Σ by Lemma 6.A.4. Since Q_1, and thus I_{Q_1}, is consistent, and the chase step preserves consistency, we have that $chase_\Sigma(I_{Q_1})$ is consistent too. Since $(\mathbf{Const}, <)$ is an infinite dense order, for every variable in $chase_\Sigma(I_{Q_1})$ we can assign a distinct constant such that every comparison in $chase_\Sigma(I_{Q_1})$ is satisfied. Let I be the instance obtained from $chase_\Sigma(I_{Q_1})$ by this assignment. We have that $I \models Q_1$ and I satisfies Σ. By assumption, it holds that $I \models Q_2$, i.e., there is a homomorphism $h : Q_2 \rightarrow I$. Since the assignment of the variables in $chase_\Sigma(I_{Q_1})$ is 1-1, we have that there is a homomorphism $g : Q_2 \rightarrow chase_\Sigma(I_{Q_1})$, i.e., $chase_\Sigma(I_{Q_1}) \models Q_2$, as needed. □

Note that the conditions of Claim 6.A.5 can be checked in PTIME according to Lemma 6.A.6 below. □

Note that for a concept from $L_{\mathbf{S}}$, the corresponding translation to first order logic is a CQ with comparisons of the form $\Phi(x, \bar{y}), \theta(x), \theta'(\bar{y})$, where Φ is a conjunction of atoms R_i such that for $i \neq j$ it holds $Var(R_i) \cap Var(R_j) = \{x\}$, $\theta(x)$, and $\theta'(\bar{y})$

are comparisons. In the above proof we reduced the containment of such queries to Boolean containment. The corresponding Boolean CQ with comparisons has the form $P(z, x), \Phi(x, \bar{y}), \theta(x), \theta'(\bar{y})$, where P is a fresh relational symbol. We show that such queries can be evaluated over a generalized instance in PTIME.

Lemma 6.A.6. *Let I be a generalized instance over* $\mathbf{S}$ *and Var, Q the Boolean* CQ *with comparisons corresponding to a $L_{\mathbf{S}}$ concept. Then deciding if $I \models Q$ can be done in* PTIME.

Proof. Checking $I \models Q$ amounts to checking existence of a homomorphism $h : Var(Q) \to Var$ such that

- if $R(\bar{x}) \in Q$ then $R(h(\bar{x})) \in I$,
- the comparisons of $h(x)$ must imply the comparisons of x.

From the above discussion we have that Q has the form $\Phi(x, \bar{y}), \theta(x), \theta'(\bar{y})$, where Φ is a conjunction of atoms such that for $R_i, R_j \in \Phi, i \neq j$, it holds $Var(R_i) \cap Var(R_j) = \{x\}$, $\theta(x)$ and $\theta'(\bar{y})$ are comparisons. We provide the following algorithm. For each atom $R_i(x, \bar{y}^i)$ (w.l.o.g. we can assume that x is in the first attribute) in Q, we compute the set $Var_i = \{z \in Var \mid R_i(z, \bar{w}^i) \in I, \theta(z) \models \theta(x), \theta'(\bar{w}^i) \models \theta'(\bar{y}^i)\}$. Such set can be computed in PTIME. Then $I \models Q$ if and only if $\cap_i Var_i \neq \emptyset$. □

6.B Missing proofs of Section 5

6.B.1 Proofs for Section 6.5.1

Theorem 6.5.1.

(i) The problem CHECK-MGE *is solvable in* PTIME.

(ii) The problem EXISTENCE-OF-EXPLANATION *is* NP*-complete. It remains* NP*-complete even for bounded schema arity.*

Proof. (i) First, given a tuple of concepts $E = (C_1, \ldots, C_m)$, we can check in PTIME if E is an explanation. Then, to check if E is a most-general explanation, we need to check if for each i, the concept C_i cannot be replaced by a concept C'_i such that $C_i \sqsubseteq_{\mathcal{O}} C'_i$ by keeping an empty intersection with Ans. Thus, for each position i in E, we try to replace C_i with all the possible concepts in $\mathcal{C} \setminus \{C_i\}$, which can be done in PTIME.

(ii) First, for membership in NP, we can guess a tuple of concepts of polynomial size, and check in PTIME if it is an explanation.

Secondly, we prove hardness via a reduction from SET COVER. Let us first recall the problem: given a triple (S, X, k), where S is a set, X is a set containing subsets of S, and k is an integer, SET COVER is the problem of deciding if there exists a subset of X of size k such that the union of the elements in the subset covers S. Given an instance (S, X, k) of SET COVER, we build an instance of EXISTENCE-OF-EXPLANATION with bounded schema arity.

Let the why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ be defined as:

$$\begin{aligned}
&\mathbf{S} = \{R(A)\}\\
&I = \{R(s) \mid s \in S\}\\
&q(x, \ldots, x) \text{ :- } R(x) \text{ where the arity of } q \text{ is } k\\
&\overline{a} = (a, \ldots, a) \text{ for some arbitrary } a \notin S
\end{aligned}$$

Let $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ be the $\mathbf{S}$-ontology defined as:

$$\begin{aligned}
&\mathcal{C} = \{C_Y | Y \in X\}\\
&\sqsubseteq = \{(C_Y, C_K) \mid K \subseteq Y\}\\
&ext(C_Y, I) = S \cup \{a\} \setminus Y, \text{ for each } Y \in X
\end{aligned}$$

Note that I is consistent with $\mathcal{O}$. Indeed, $C_Y \sqsubseteq C_K$ is equivalent to $K \subseteq Y$ which in turn implies $S \cup \{a\} \setminus Y \subseteq S \cup \{a\} \setminus K$, i.e., $ext(C_Y, I) \subseteq ext(C_K, I)$. Then we show that there exists a set cover of size k for S iff the answer for EXISTENCE-OF-EXPLANATION is "yes".

($\Rightarrow$). Let $Y_1, \ldots, Y_k$ be a solution for SET COVER. We show that $E = (C_1, \ldots, C_k)$, where $C_i = C_{Y_i}$, is an explanation for $\overline{a} \notin Ans$.

Assume, towards a contradiction, that E is not an explanation for $\overline{a} \notin Ans$. Then one of the following conditions must hold: (i) there exists C_i such that $a \notin ext(C_i, I)$; (ii) $ext(C_1, I) \times \ldots \times ext(C_k, I) \cap Ans \neq \emptyset$. Both these conditions lead to contradiction. In particular, by construction, for each i, $a \in C_i$, which refutes (i). Suppose that (ii) holds. Then there exists an element $s \in S$ such that for each i, $s \in ext(C_i, I)$ and thus $s \notin Y_i$ (since $ext(C_i, I)$ is the complement of Y_i w.r.t. $S \cup \{a\}$). But this contradicts the assumption that $Y_1, \ldots, Y_k$ is a set cover.

($\Leftarrow$). Let $E = (C_1, \ldots, C_k)$, where $C_i = C_{Y_i}$, be an explanation for $\overline{a} \notin Ans$. We show that $Y_1, \ldots, Y_k$ is a solution for SET COVER.

Suppose, towards a contradiction, that $Y_1, \ldots, Y_k$ is not a set cover. Thus there exists at least one element $s \in S$ such that $s \notin Y_i$ for each i, which, in turn, implies that $s \in ext(C_i, I)$ for each i. But then $ext(C_1, I) \times \ldots \times ext(C_k, I) \cap Ans \neq \emptyset$, which contradicts the assumption that E is an explanation. □

Theorem 6.5.2. *Let a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and an $\mathbf{S}$-ontology $\mathcal{O}$ be an input to* EXHAUSTIVE SEARCH ALGORITHM *and let $\mathcal{X}$ be the corresponding output. The following hold:*

(i) $\mathcal{X}$ is the set of all most-general explanations for $\overline{a} \notin Ans$ (modulo equivalence);

(ii) EXHAUSTIVE SEARCH ALGORITHM *runs in* EXPTIME *in the size of the input (in* PTIME *if we fix the arity of the input query).*

Proof. (i) *(Correctness)* We show the correctness of EXHAUSTIVE SEARCH ALGORITHM by proving the following claims: (*a*) every most-general explanation belongs to $\mathcal{X}$; (*b*) every $E \in \mathcal{X}$ is a most-general explanation.

Suppose E is a most-general explanation for $\overline{a} \notin Ans$. Since E is an explanation, by definition its extension contains $\overline{a}$ and does not intersect Ans, thus it is part of $\mathcal{X}$ at the beginning (line 2). Moreover, since it is a most-general explanation, there is no explanation $E' \in \mathcal{X}$ such that $E' >_{\mathcal{O}} E$, thus the test in line 4 fails and E is never removed from $\mathcal{X}$. This proves claim (*a*). Suppose there exists an element $E \in \mathcal{X}$ that is not a most-general explanation for $\overline{a} \notin Ans$. First, observe that E must be an explanation, otherwise it could not be in $\mathcal{X}$, according to line 2. Then it must be the case that there exists an explanation E' such that $E' >_{\mathcal{O}} E$. We may assume w.l.o.g. that E' is a most-general explanation. Then, as proved in point (*a*), $E' \in \mathcal{X}$, therefore the pair E, E' would have been considered for the comparison in line 4, and E would have been removed from $\mathcal{X}$, which leads to a contradiction and proves point (*b*).

(ii) *(Running time)* First, notice that the extension $ext(C, I)$ of a concept is polynomial-time computable by definition. Let us call $p(k + |\mathrm{adom}(I)|)$ this polynomial, where k be the maximal size of concepts from $\mathcal{C}$. Then, building each set $C(a_i)$ at line 1 requires $O(|\mathcal{C}| \cdot |p(k + |\mathrm{adom}(I)|)|)$. Building the set $\mathcal{X}$ at line 2 requires $O((m \cdot |\mathcal{C}| \cdot |p(k + |\mathrm{adom}(I)|)|)^m \cdot |Ans|)$, where m is the arity of q. Finally, the steps from line 3 to 5 require $O(|\mathcal{C}|^{2m})$ since the size of X is bounded by $|\mathcal{C}|^m$. Therefore EXHAUSTIVE SEARCH ALGORITHM runs in $O(|\mathcal{C}| \cdot |p(k + |\mathrm{adom}(I)|)| + (m \cdot |\mathcal{C}| \cdot |p(k + |\mathrm{adom}(I)|)|)^m \cdot |Ans| + |\mathcal{C}|^{2m})$, which is exponential in the size of the input. For a fixed m, the size of the query, the algorithm becomes polynomial. □

6.B.2 Proofs for Section 6.5.2

Proposition 6.5.1. *Let* $(\mathbf{S}, I, q, Ans, \overline{a})$ *be a why-not instance. If E is an explanation for* $\overline{a} \notin Ans$ *w.r.t.* $\mathcal{O}_I$ *(resp.* $\mathcal{O}_{\mathbf{S}}$*), then there exists an explanation* E' *for* $\overline{a} \notin Ans$ *such that* $E <_{\mathcal{O}_I[\mathcal{K}]} E'$ *(resp.* $E <_{\mathcal{O}_{\mathbf{S}}[\mathcal{K}]} E'$*), where* $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$ *and each constant in* E' *belongs to* $\mathcal{K}$.

Proof. Before turning to the proof we note that each (consistent) selection expression is equivalent to a selection expression of the form $\sigma_C(R)$, where C is a list of comparisons such that for every attribute name A, C has at most one occurrence of $A < c_1$ or $A \leq c_1$ for arbitrary $c_1 \in \mathbf{Const}$, and at most one occurrence of $A > c_2$ or $A \geq c_2$ for arbitrary $c_2 \in \mathbf{Const}$. Additionally, if C contains an expression $A = c$, then it is the only expression in C mentioning the name A. From now on we assume that the selection expressions have this normalized form.

Let I be an instance, $\overline{a} = (a_1, \ldots, a_m)$ a tuple of elements from $\mathbf{Const}$, and q a query such that $\overline{a} \notin Ans$. By $\mathcal{K}$ we denote the set $\mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$. Let E be of the form $(E_1, \ldots, E_n)$, where each E_i is a conjunction $E^i_1 \sqcap \ldots \sqcap E^i_{l(i)}$ of atomic concepts of the form $\pi_A(\sigma_C(R))$ or nominals $\{c\}$. If E contains only concepts that use values from $\mathcal{K}$ then there is nothing to prove. Suppose otherwise. First we note that all the nominal concepts in the conjunction E_i must be exactly $\{a_i\}$, since otherwise $a_i \notin [\![E^i]\!]^I$ which is a contradiction. Thus we only consider the case when a constant outside of $\mathcal{K}$ appears in a selection expression. For this, we iteratively apply the following procedure producing a more general explanation containing less concepts which contain values outside of $\mathcal{K}$. Suppose there are indices i and j such that E^i_j is a conjunct of E_i of the form $\pi_A(\sigma_C(R))$ and C contains an expression $B\mathsf{op}c$ where $c \notin \mathcal{K}$. We show that we can replace C with C' that does not contain the expression anymore, resulting in an explanation E' that is more general than E.

We distinguish the following cases.

- $\mathsf{op} \in \{=\}$. This case is impossible. Since $a_i \in ext(E^i_j, I)$, there is a tuple $\bar{b} \in R^I$ such that $\pi_A(\bar{b}) = a_i$ and $\pi_B(\bar{b}) = c$. But $\pi_B(\bar{b}) \in \mathcal{K}$ and $c \notin \mathcal{K}$, a contradiction.

- $\mathsf{op} \in \{\leq, <\}$. Note C contains only one expression of the form $B\mathsf{op}c$. We distinguish two cases: when there exists or not $c'' \in \mathcal{K}$ such that $c < c''$

  - There exists $c'' \in \mathcal{K}$ such that $c < c''$. Let c' be an element of $\mathcal{K}$ which is closest to c from above, i.e. $c' = \min\{c_1 \mid c < c_1 \text{ and } c_1 \in \mathcal{K}\}$. Note such c' exists due to the assumption $c'' \in \mathcal{K}$. Let E' be the tuple of concepts obtained from E by replacing $E^i_j = \pi_A(\sigma_C(R))$ with the concept $E'^i_j = \pi_A(\sigma_{C'}(R))$, where $C' = (C \setminus \{B\mathsf{op}c\}) \cup \{B < c'\}$. We claim that E' is an explanation that is more general than E. First, it is easy to see that for every $k \leq m$ it holds $a_k \in ext(E_k, I)$. Indeed, we only need to check that $a_i \in ext(E'^i_j, I)$. Since $a_i \in ext(E^i_j, I)$, there is a tuple $\bar{b} \in R^I$ such that $\pi_A(\bar{b}) = a_i$ and $\pi_B(\bar{b})\mathsf{op}c$. By the choice of c', we also have $\pi_B(\bar{b}) < c'$. Thus, the tuple $\bar{b}$ is a witness for $a_i \in ext(E'^i_j, I)$.

    Now we show the extension of E' does not intersect with the query answer. Suppose the opposite, there exists a tuple $\bar{b} \in (ext(E_1, I) \times \ldots \times ext(E'_i, I) \times \ldots \times ext(E_n, I)) \cap Ans$. This implies $b_i \in ext(E'^i_j, I)$ and thus there exists a tuple $\bar{d} \in R^I$ such that $\pi_A(\bar{d}) = b_i$ and $\pi_B(\bar{d}) < c'$ Note that each element of $\bar{b}$ is in $\mathcal{K}$. Note also that by construction, for every tuple $\bar{e}$ with $\pi_A(\bar{e}) = b_i$ and $\pi_B(\bar{e}) < c'$ it holds $\pi_B(\bar{e}) > c$. Indeed, otherwise $\bar{b} \in (ext(E_1, I) \times \ldots \times ext(E_i, I) \times \ldots \times ext(E_n, I)) \cap Ans$, which is a contradiction. Thus we have that $c < \pi_B(\bar{d}) < c'$ and $\pi_B(\bar{d}) \in \mathcal{K}$, which is a contradiction with the choice of c'.

    We show that E' is more general than E w.r.t. $\mathcal{O}_\mathbf{S}$ (w.r.t. $\mathcal{O}_I$ would then follow). To this purpose, it is enough to show that $E^i_j \sqsubseteq_\mathbf{S} E'^i_j$. Let I' be an arbitrary instance of $\mathbf{S}$, and $b \in ext(E^i_j, I')$. This means there exists a tuple $\bar{d}$ such that $\pi_A(\bar{d}) = b$ and $\pi_B(\bar{d}) < c$. Since $c < c'$, we also have $\pi_B(\bar{d}) < c'$ and thus $b \in ext(E'^i_j, I')$ as needed.

  - There is no $c'' \in \mathcal{K}$ with $c < c''$. In this case let E' be obtained from E by replacing $E^i_j = \pi_A(\sigma_C(R))$ with the concept $E'^i_j = \pi_A(\sigma_{C'}(R))$, where $C' = C \setminus \{B\mathsf{op}c\}$. If C' is empty, we take $E'^i_j = \pi_A(R)$. Similarly to the previous case, it is easy to show that for every $k \leq m$, it holds $a_i \in ext(E_k, I)$. Suppose that there exists a tuple $\bar{b} \in (ext(E_1, I) \times \ldots \times ext(E'_i, I) \times \ldots \times ext(E_n, I)) \cap Ans$. This implies $b_i \in ext(E'^i_j, I)$ and thus there exists a tuple $\bar{d} \in R^I$ such that $\pi_A(\bar{d}) = b_i$ and $\bar{d}$ satisfies the constraints C'. Note that $\pi_B(\bar{d}) < c$ since $\pi_B(\bar{d}) \in \mathcal{K}$, $c \notin \mathcal{K}$ and there is no $c'' \in \mathcal{K}$ with $c < c''$. But this implies that $\overline{b} \in (ext(E_1, I) \times \ldots \times ext(E_i, I) \times \ldots \times ext(E_n, I)) \cap Ans$ which is a contradiction.

Similarly to the previous case we can show that E' is more general than E w.r.t. $\mathcal{O}_{\mathbf{S}}$. This holds essentially because we make the selection expression weaker, thus making E' more general than E.

- $\mathsf{op} \in \{\geq, >\}$. The argument is similar to the previous case. We either take the closest c' from $\mathcal{K}$ with $c' < c$ or remove the constraint $B\mathsf{op}c$. $\square$

Lemma 6.5.1. *Given an instance I of schema $\mathbf{S}$ and a set of constants X, we can compute in polynomial time a selection-free $L_{\mathbf{S}}$ concept, denoted $\mathsf{lub}_I(X)$, that is the smallest concept whose extension contains all the elements in X definable in the language. In particular, the following hold:*

(i) $X \subseteq ext(\mathsf{lub}_I(X), I)$,

(ii) there is no concept C' in selection-free $L_{\mathbf{S}}$ such that $C' \sqsubset_I \mathsf{lub}_I(X)$ and $X \subseteq ext(C', I)$.

Proof. Let $\mathsf{lub}_I(X) = \bigsqcap\{C \mid C \in L^{\min}_{\mathbf{S}}[\mathcal{K}] \text{ and } X \subseteq ext(C, I)\}$.

The first item holds by construction: for each conjunct C of $\mathsf{lub}_I(X)$, $ext(C, I)$ contains X. For the second item, suppose there exists a concept C' such that $C' \sqsubset_I \mathsf{lub}_I(X)$ and $X \subseteq ext(C', I)$. But since $X \subseteq ext(C', I)$, C' must be in the conjunction defining $\mathsf{lub}_I(X)$, which contradicts the fact that $C' \sqsubset_I \mathsf{lub}_I(X)$. $\square$

Theorem 6.5.3 (Correctness and running time of INCREMENTAL SEARCH ALGORITHM)**.** *Let the why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ be an input to* INCREMENTAL SEARCH ALGORITHM *and E the corresponding output. The following holds:*

(i) E is a most-general explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I = (\mathcal{C}, \sqsubseteq_I, ext)$, where $\mathcal{C}$ is selection-free $L_{\mathbf{S}}$;

(ii) INCREMENTAL SEARCH ALGORITHM *runs in* PTIME *in the size of the input.*

Proof. (i) *(Correctness)* First, observe that E is an explanation. In particular, in the worst case E is the trivial explanation that has in each position j the nominal corresponding to the constant a_j in the input tuple $\vec{a}$. Then, the proof builds on Lemma 6.5.1.

Now suppose that $E = (C_1, \ldots, C_m)$ is not a most-general explanation for $\overline{a} \notin Ans$. That is, there exists an explanation $E' = (C'_1, \ldots, C_m)$ for $\overline{a} \notin Ans$ that is strictly more general than E, i.e., $E' >_{\mathcal{O}_I} E$. Thus, by definition, $C_j \sqsubseteq C'_j$ for every j, $1 \leq j \leq m$. Let $\mathcal{X} = (X_1, \ldots, X_m)$ the support set used by the algorithm for computing E. Since $E' >_{\mathcal{O}_I} E$, there exists a position j for which $C_j \sqsubset C'_j$, implying that $ext(C_j, I) \subset ext(C'_j, I)$. Let b be an element in $ext(C'_j, I) \setminus ext(C_j, I)$. Then, since b would have been considered in line 5, if it is not in X_j it must be the case that adding it would cause the resulting explanation to intersect Ans. But $b \in ext(C'_j, I)$, thus $E' \cap Ans \neq \emptyset$, i.e., E' is not an explanation, which contradicts our assumption. Finally, observe that INCREMENTAL SEARCH ALGORITHM outputs a most-general explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I[\mathcal{K}]$, but Proposition 6.5.1 guarantees that we can restrict to $\mathcal{O}_I[\mathcal{K}]$, with $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$, without losing any most-general explanation.

(ii) *(Running time)* Let $|L_{\mathbf{S}}^{\min}|$ the number of distinct concept expressions, up to logical equivalence, that can be defined in $L_{\mathbf{S}}^{\min}$ and let m be the arity of q. First, observe that initializing the support set (line 2) takes m steps, and that building the first candidate explanation (line 3) takes $O(m \cdot |L_{\mathbf{S}}^{\min}| \cdot p(|\mathrm{adom}(I)|)$, since each $\mathsf{lub}_I(X_j)$ takes $|L_{\mathbf{S}}^{\min}| \cdot p(|\mathrm{adom}(I)|)$ steps. Moreover, the nested for-loop (lines 4–11) takes $O(m \cdot |\mathrm{adom}(I)| \cdot |L_{\mathbf{S}}^{\min}| \cdot p(|\mathrm{adom}(I)|)$. Thus the running time of the algorithm is $O(m + (m \cdot |L_{\mathbf{S}}^{\min}| \cdot p(|\mathrm{adom}(I)|)) \cdot (1 + |\mathrm{adom}(I)|))$, and since $|L_{\mathbf{S}}^{\min}|$ is polynomial in the size of $\mathbf{S}$ (see Proposition 6.4.2), we can conclude that INCREMENTAL SEARCH ALGORITHM runs in PTIME. □

Lemma 6.5.2. *Given an instance I of $\mathbf{S}$ and a set of constants X, we can compute in exponential time a $L_{\mathbf{S}}$ concept, denoted $\mathsf{lub}_I^{\sigma}(X)$, that is the smallest concept whose extension contains all the elements in X definable in the language. Such a concept is polynomial-time computable for bounded schema arity. In particular, the following hold:*

(i) $X \subseteq ext(\mathsf{lub}_I^{\sigma}(X), I)$,

(ii) there is no concept C' in $L_{\mathbf{S}}$ such that $C' \sqsubset_I \mathsf{lub}_I^{\sigma}(X)$ and $X \subseteq ext(C', I)$.

Proof. Let $\mathsf{lub}_I^{\sigma}(X) = \bigsqcap\{C \mid C =$ intersection-free $L_{\mathbf{S}}$ and $X \subseteq ext(C, I)\}$.

The proof is analogous to the proof of Lemma 6.5.1. In this case, however, the bound on the number of distinct concept definable via intersection-free $L_{\mathbf{S}}$ is single exponential (see Proposition 6.4.2), therefore the set can be computed in EXPTIME. If we bound the arity of the schema, the bound on the number of concepts in intersection-free $L_{\mathbf{S}}$ becomes polynomial, thus in this case $\mathsf{lub}_I^{\sigma}(X)$ can be computed in PTIME. □

Theorem 6.5.4 (Correctness and running time of INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS)**.** *Let the why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ be an input to* INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS *and E the corresponding output. The following hold:*

(i) E is a most-general explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I = (\mathcal{C}, \sqsubseteq_I, ext)$, where $\mathcal{C}$ is $L_{\mathbf{S}}$;

(ii) INCREMENTAL SEARCH ALGORITHM *runs in* EXPTIME *in the size of the input (in* PTIME *for bounded schema arity).*

Proof. Both *correctness* (1) and *running time* (2) follow from the analysis in Theorem 6.5.3, and from the fact that in this case Lemma 6.5.2 guarantee that there is no explanation E' using concepts from $L_{\mathbf{S}}$ such that $E' >_{\mathcal{O}} E$, and also that $\mathsf{lub}_I^{\sigma}(X)$ is exponential-time computable (polynomial-time computable for bounded schema arity). □

6.B.3 Proofs for Section 6.5.3

Proposition 6.5.3. *There is an algorithm that solves* COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$

- *in* 2EXPTIME *for $L_{\mathbf{S}}$, provided that the input schema $\mathbf{S}$ is from a class for which concept subsumption can be checked in* EXPTIME*,*

- *in* EXPTIME *for selection-free $L_{\mathbf{S}}$, and projection-free $L_{\mathbf{S}}$, provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* PTIME *for $L_{\mathbf{S}}^{\min}$, if the arity of q is fixed and provided that the input schema* $\mathbf{S}$ *is from a class for which concept subsumption can be checked in* PTIME*.*

Proof. First, from Proposition 6.4.2, it follows that the number of distinct concept expressions that one can build up to logical equivalence is at most single exponential for selection-free or intersection-free $L_{\mathbf{S}}[\mathcal{K}]$, and double exponential for $L_{\mathbf{S}}[\mathcal{K}]$. Next, the computation of $\sqsubseteq_{\mathbf{S}}$ would require testing concept subsumption for exponentially many pairs of concepts. Moreover, since the ext function is polynomial-time computable, we can therefore build $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ in EXPTIME for selection-free or intersection-free $L_{\mathbf{S}}[\mathcal{K}]$, and in 2EXPTIME for the concept language $L_{\mathbf{S}}[\mathcal{K}]$.

This gives us bounds on materializing $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ depending on the concept language. Then we can use $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ as input for EXHAUSTIVE SEARCH ALGORITHM, and solve COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$.

We know, from Theorem 6.5.2, that EXHAUSTIVE SEARCH ALGORITHM runs in EXPTIME in the size of the input (PTIME for bounded query arity). Moreover, the size of the materialized $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ is in the order of $2^{2^{|\mathbf{S}|}}$. Therefore the main factor in solving COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ is $O(|\mathcal{O}_{\mathbf{S}}|^{2^m})$, which is double exponential for $L_{\mathbf{S}}$, and single exponential for the other languages, even for bounded query arity, if concept subsumption can be checked in EXPTIME. Moreover, solving COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ is in PTIME for $L_{\mathbf{S}}^{\min}$ with bounded query arity, if concept subsumption can be checked in PTIME. □

6.C Missing proofs for Section 6

Proposition 6.6.2. *There is a polynomial-time algorithm that takes as input an instance I of a schema* $\mathbf{S}$*, as well as an $L_{\mathbf{S}}$ concept expression C, and produces an irredundant concept expression C' such that $C \equiv_{\mathcal{O}_I} C'$.*

Proof. We provide an easy algorithm.

Algorithm 3: Irredundant concept

```
Input: a concept $C = \sqcap\mathbf{C}$, an instance $I$
Output: an equivalent irredundant concept
1 foreach $C_i \in \mathbf{C}$ do
2     if $\exists C_j \in \mathbf{C}$ s.t. $C_j \neq C_i$ and $C_i \sqsubseteq_I C_j$ then
3         $\mathbf{C} = \mathbf{C} \setminus C_j$
4 return $\sqcap\mathbf{C}$
```

The algorithm consists of one for-loops of length of C. The inner if-statement can be checked in PTIME since the subsumption $C \sqsubseteq_I C'$ can be checked in PTIME. Thus overall, Irredundant concept runs in PTIME.

We now prove that the algorithm produces an $\mathcal{O}_I$-equivalent concept. For this it is enough to make sure that the step on line 3 preserves equivalence. Suppose C' was obtained from C by removing the concept C_j which satisfied the if-condition on line 2. Since $C' \subset C$, it holds $C \sqsubseteq_I C'$. Conversely, since $C = C' \sqcap C_j$ and $C'' \sqsubseteq_I C_j$ for some $C'' \in \mathbf{C}$, it holds $C' \sqsubseteq_I C$.

There is no proper irredundant subset since we exhaustively remove concepts that are implied by other concepts in the conjunction. $\square$

Proposition 6.6.1. *Given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$, the problem of finding a most-general explanation to $\bar{a} \notin Ans$ of minimal length is* NP*-hard.*

Proof. We reduce from set cover problem. Recall, given a triple (U, S), where U is a set, S is a set containing subsets of U, SET COVER is the problem of finding a minimal size subset of S such that the union of the elements in the subset covers U. Given an instance (U, S) of SET COVER, we build a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ as follows.

$$\begin{array}{ll}
\mathbf{S} & = \{R_C(A) \mid C \in S\} \\
I & = \{R_C(c) \mid C \in S, c \notin C\} \cup \{R_C(a), R_C(b) \mid C \in S\}, \\
 & \text{ where } a, b \notin U \text{ are fresh elements} \\
q(x) & \text{:- } \cup_{C \in S} R_C(x) \\
Ans & = U \\
\overline{a} & = (a)
\end{array}$$

Let $\mathcal{O}_I = (\mathcal{C}, \sqsubseteq_I, ext)$, where $\mathcal{C}$ is selection-free $L_{\mathbf{S}}$, be the $\mathbf{S}$-ontology derived from I. Without loss of generality we can assume that the length of each atomic concept $\pi_1(R_C)$ is equal 1 (since each of the atomic concept is of the same size). Note that there are finitely many concepts in $\mathcal{C}$ and the extension of each atomic concept $\pi_A(R_C)$ is equal to $(U \setminus C) \cup \{a, b\}$. Thus, in order for a concept C in selection-free $L_{\mathbf{S}}$ to be an explanation, its extension must be equivalent to either $\{a\}$ or $\{a, b\}$. Indeed, $ext(C, I)$ must contain a and not intersect with $Ans = U$, which is only possible when $ext(C, I) = \{a\}$ or $ext(C, I) = \{a, b\}$. Thus, in order for a concept C to be a most-general explanation w.r.t. $\mathcal{O}_I$, its extension must be equivalent to $\{a, b\}$. We prove that $ext(\sqcap\{\pi_1(R_{C_1}), \ldots, \pi_1(R_{C_n})\}, I) = \{a, b\}$ if and only if $\bigcup_{i=1}^n C_i = U$, i.e., it covers U. Indeed, $ext(\sqcap\{\pi_1(R_{C_1}), \ldots, \pi_1(R_{C_n})\}, I) = \bigcap_{i=1}^n (U \setminus C_i) \cup \{a, b\}$ and it is equal $\{a, b\}$ iff $\bigcap_{i=1}^n (U \setminus C_i) = \emptyset$, i.e., $U = \bigcup_{i=1}^n C_i$. Note that the length of a most-general explanation precisely corresponds to the size of a set cover. Thus, finding a cover of U of minimal is equivalent to finding a most-general explanation for $\bar{a} \notin Ans$ of minimal length. $\square$

Proposition 6.6.3. *Given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and an explanation E to why $\bar{a} \notin Ans$, the problem of finding a minimized explanation equivalent to E is* NP*-hard.*

Proof. (Sketch) Again we reduce from SET COVER. The construction is exactly the same as in the previous proposition. Now as the input explanation E we take $\sqcap\{\pi_1(R_C) \mid C \in S\}$. The extension of this concept in I is equal to $\{a, b\}$. Thus arguing as in the previous Proposition, finding a minimal size explanation equivalent to E amounts to finding a minimal size set cover for U. $\square$

6.C.1 Cardinality based preference

Proposition 6.6.4. *Assuming P≠NP, there is no* PTIME *algorithm that takes as input a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an* $\mathbf{S}$*-ontology* $\mathcal{O}$*, and produces a* $>^{card}$*-maximal explanation for* $\overline{a} \notin Ans$*. This holds even for unary queries.*

To show hardness we reduce from the following modified version of set cover problem.

Problem 6.C.1. *(Subset cover problem)*

- *Given: a universe* U*, its proper subset* U_1 *and a class of sets* $\mathcal{S}$ *such that* $\cup\mathcal{S} \subseteq U$ *and which covers* U_1*, i.e.* $U_1 \subseteq \cup\mathcal{S}$*.*
- *Find: a class of sets* $\mathcal{C} \subseteq \mathcal{S}$ *such that it covers* U_1 *and* $|\cup\mathcal{C} \setminus U_1|$ *is minimal.*

We prove the following.

Proposition 6.C.1. *The subset cover problem is* NP*-hard.*

Proof. We first recall the definition of L-reduction.

Definition 6.C.1. *Let A and B be optimization problems and* c_A *and* c_B *their respective cost functions. A pair of functions* f *and* g *is an L-reduction of* A *to* B *if all of the following conditions are met:*

- *functions* f *and* g *are computable in polynomial time,*
- *if* x *is an instance of problem A, then* $f(x)$ *is an instance of problem B,*
- *if* y *is a solution to* $f(x)$*, then* $g(y)$ *is a solution to* x*,*
- *there exists a positive constant* α *such that* $\mathrm{OPT_B}(f(x)) \leq \alpha\mathrm{OPT_A}(x)$*,*
- *there exists a positive constant* β *such that for every solution y to f(x)* $|\mathrm{OPT_A}(x) - c_A(g(y))| \leq \beta|\mathrm{OPT_B}(f(x)) - c_B(y)|$.

In our case the cost functions are the following:

- Set cover problem: $|\mathcal{C}|$ for a solution $\mathcal{C}$,
- Subset cover problem: $|\cup\mathcal{C} \setminus U_1|$ for a solution $\mathcal{C}$.

We L-reduce the set cover problem to the subset cover problem. To this purpose we provide the functions f and g such that all the requirements are met. Let $(U, \mathcal{S})$ be an arbitrary instance of set cover problem. Then we define $f(U, \mathcal{S}) = (U', U_1', \mathcal{S}')$, where

- $U' = U \cup \{a_1, \ldots, a_m\}$, where $m = |\mathcal{S}|$ and each a_i is a fresh element,
- $U_1' = U$,
- $\mathcal{S}' = \{C_i \cup \{a_i\} \mid C_i \in \mathcal{S}\}$.

For a family of sets $\mathcal{C}' \subseteq \mathcal{S}'$ we define $g(\mathcal{C}') = \{C \setminus \{a_1, \ldots, a_m\} \mid C \in \mathcal{C}'\}$. It is clear from the definitions that both f and g are computable in PTIME. Next it can also be seen that if $\mathcal{C}' \subseteq \mathcal{S}'$ is a subset cover for $f(U, \mathcal{S}) = (U', U_1', \mathcal{S}')$, then $g(\mathcal{C}')$ is a set cover for U. The needed constants α and β are equal to 1. □

Proof. (of Proposition 6.6.4) We construct an L-reduction from the subset cover problem to the problem of finding a $<^{card}$-maximal explanation.

Let A be the subset cover problem and B the problem of finding a $<^{card}$-maximal explanation. As before, the cost function for A for the input $x = (U, U_1, \mathcal{S})$ and the solution $s = \mathcal{C}$ is defined as $c_A(x, s) = | \cup \mathcal{C} \setminus U_1|$. For the problem B, the cost function is defined as follows: let $x = (\mathbf{S}, I, q, Ans, \overline{a}, \mathcal{O})$ be an instance of B and $s = (C_1, \ldots, C_k)$ an explanation to $\bar{a} \notin Ans = (q_1(I), \ldots, q_k(I))$, then $c_B(x, s) = |I| - \Sigma_{i=1}^{k} |ext(C_i, I)| - \Sigma_{i=1}^{k} |q_i(I)|$.

For an instance $x = (U, U_1, \mathcal{S})$ of the subset cover problem we define $f(x) := (\mathbf{S}, I, q, Ans, \overline{a}, \mathcal{O})$, where

- $\mathbf{S} = \{R_C(A) \mid C \in \mathcal{S}\}$,
- $\bar{a} = (a)$, where $a \notin U$ is a fresh element,
- $I = \{R_C(c) \mid C \in \mathcal{S}, c \in C\} \cup \{R_C(a) \mid C \in \mathcal{C}\}$ is an instance, which means that $adom(I) = U \cup \{a\}$,
- q is such that $q(I) = Ans = U_1$.
- $\mathcal{O} = (\hat{\mathcal{C}}, \sqsubseteq, ext)$ such that $\hat{\mathcal{C}}$ consists of atomic concepts $\hat{C}$ for each set $C \in \mathcal{S}$, their extensions in I are defined as $ext((\hat{C}, I) = (U \setminus C) \cup \{a\}$, and $\hat{C}_1 \sqsubseteq \hat{C}_2$ iff $C_2 \subseteq C_1$,

Next, let $\hat{C} = \hat{C}_1 \sqcap \ldots \sqcap \hat{C}_k$ be a concept with $\hat{C}_i \in \hat{\mathcal{C}}$ for the input $f(U, U_1, \mathcal{S}) = (\mathbf{S}, I, q, Ans, \overline{a}, \mathcal{O})$. We define $g(\hat{C}) := \{C_1, \ldots, C_k\} \subseteq \mathcal{S}$. It is clear that both f and g are computable in PTIME. Also

- If $(U, U_1, \mathcal{S})$ is an instance of the subset cover problem, then $f(U, U_1, \mathcal{S})$ is an instance of t the problem of finding a $<^{card}$-maximal explanation.
- Let $\hat{C} = \hat{C}_1 \sqcap \ldots \sqcap \hat{C}_k$ be an explanation to $a \notin Ans$ with the input $f(U, U_1, \mathcal{S}) = (\mathbf{S}, I, q, Ans, \overline{a}, \mathcal{O})$. Then we claim that $g(\hat{C})$ is a subset cover for $(U, U_1, \mathcal{S})$. First, it holds that $(ext(\hat{C}_1 \sqcap \ldots \sqcap \hat{C}_k, I)) \cap Ans = \emptyset$. By definition we have $Ans = U_1$ and $ext(\hat{C}_1 \sqcap \ldots \sqcap \hat{C}_k, I) = \bigcap_i^k ((U \setminus C_i) \cup \{a\}) = (U \setminus \bigcup_i^k C_i) \cup \{a\}$. Since the intersection of those two sets is empty and also $U_1 \subset U$, we have that $U_1 \subseteq \bigcup_i^k C_i$, i.e. $g(\hat{C}) = \{C_1, \ldots, C_k\}$ covers U_1.
- Let $\mathcal{C}$ be an optimal subset cover for $(U, U_1, \mathcal{S})$. Then the cost $| \cup \mathcal{C} \setminus U_1|$ is minimal. Let also $\hat{C} = \hat{C}_1 \sqcap \ldots \sqcap \hat{C}_k$ be a $<^{card}$-maximal explanation to $a \notin Ans$ for the input $f(U, U_1, \mathcal{S}) = (\mathbf{S}, I, q, Ans, \overline{a}, \mathcal{O})$. The cost of this explanation is $|I| - |ext(\hat{C}, I)| - |Ans| = | \cup_i^k C_i| - |U_1|$ which is minimal. Then it holds $| \cup \mathcal{C} \setminus U_1| \leq \alpha(| \cup_i^k C_i| - |U_1|)$ for $\alpha = 1$.

- Let $\mathcal{C}$ be an optimal subset cover for $x = (U, U_1, \mathcal{S})$ and $\hat{C} = \hat{C}_1 \sqcap \ldots \sqcap \hat{C}_k$ a $<^{card}$-maximal explanation to $a \notin Ans$ for the input $f(x) = (\mathbf{S}, I, q, Ans, \overline{a}, \mathcal{O})$. Let also $y = \hat{C}' = \hat{C}'_1 \sqcap \ldots \hat{C}'_l$ be an arbitrary explanation to $a \notin Ans$ for the input $f(x)$, and $g(y) = \{C'_1, \ldots, C'_l\}$ the corresponding subset cover for $(U, U_1, \mathcal{S})$. Then $|OPT_A(x) - c_A(g(y))| = || \cup \mathcal{C} \setminus U_1| - | \cup_i^l C'_i \setminus U_1||$. This is equal to $|| \cup_i^k C_i \setminus U_1| - | \cup_i^l C'_i \setminus U_1||$, which is equal to $|OPT_B(f(x)) - c_B(y)|$. Thus we can take $\beta = 1$. $\square$

7 Conclusion

In this thesis we contributed to the study of the containment problem for various query languages over trees and relational databases (Part I), and introduced a new framework for why-not explanations (Part II). Next we summarize the obtained results and end with future work.

7.1 Main findings

The first part of the thesis is devoted to the containment problem for expansions of XPath and conjunctive queries over trees, and conjunctive queries over relational databases. Firstly, we considered query languages over unranked trees and tried to answer the following question for a given query language $\mathcal{L}$.

RQ 1 What is the complexity of the containment problem for $\mathcal{L}$ over unranked trees?

Concretely, $\mathcal{L}$ was one of the following query languages over unranked trees.

- Positive XPath with attibute value comparisons (Chapter 3),
- Conjunctive Queries with attribute value comparisons (Chapter 3),
- Conditional Tree Patterns (Chapter 5)

Table 7.1 summarizes the results of Chapter 3, together with previously known results. We can see that adding attribute value comparisons of the form $@_a \mathsf{op}\ c$ does not increase the complexity of containment. Moreover, when the underlying ordered domain for attributes is restricted to be either finite, discrete, or with endpoints, the complexity of

	$\mathsf{PosXPath}^{@}$	$\mathsf{CQ}^{@}$
no attributes	coNP (Björklund et al., 2011)	Π_2^P (Björklund et al., 2011)
optional attributes	coNP (Thm. 3.3.2)	Π_2^P (Thm. 3.3.2)
required attributes	PSpace-hard (Thm. 3.3.3)	PSpace-hard (Thm. 3.3.3)

Table 7.1: Complexity results for containment over trees for Positive XPath and CQ with attribute value comparisons (Chapter 3).

containment remains the same as well. However, when trees are required to have at least one attribute defined in every node, the containment becomes harder (PSPACE-hard). The latter hardness result is rather strong: it already holds for tree patterns with attribute value comparisons.

In Chapter 5 we expanded tree patterns with the conditional descendant axis, resulting in Conditional Tree Patterns (CTP). The conditional axis is a natural expansion for XPath, providing expressive completeness for first-order logic on ordered unranked trees (Marx, 2005). In Chapter 5 we proved that the containment problem for CTP is PSPACE-complete. Interestingly, the lower bound is proved via establishing the fact that containment is PSPACE-hard for tree patterns expanded with unrestricted *label* negation. The lower bound in case of CTP then follows from the fact that CTP is able to express this type of negation, as far as the containment problem is concerned. More precisely, there is a polynomial reduction from the containment problem for tree patterns with label negation to the containment problem for conditional tree patterns. Notably, when negation in tree patterns is restricted to be *safe* (that is, a negated label in a node must co-occur with a positive label), then containment drops to CONP, as Table 7.1 shows.

Each CTP formula can be conveniently represented as a tree in which each descendant edge is (recursively) labeled with the tree corresponding to a CTP subformula. Thus, tree patterns form a particular case when each descendant edge is labeled with $\top$. In fact, this edge labeling is the only topological addition to tree patterns offered by conditional tree patterns. The lower bound in Chapter 5 is quite strong: the PSPACE lower bound already holds for CTP with such nested edge labelling of depth 1 (or in other words, the edge labels are tree patterns).

In Chapter 4 we considered acyclic conjunctive queries with atomic negation and arithmetic comparisons, interpreted over relational databases. It is known that the acyclicity restriction on conjunctive queries allows for tractable containment (Gottlob et al., 2001). We considered the following

RQ 2 Does acyclicity make the complexity of containment for conjuctive queries expanded with atomic negation or arithmetic comparison tractable? If not, what additional restrictions need to be imposed to make it tractable?

Table 7.2 summarizes the results of Chapter 4. According to Theorem 4.3.3, even the most restrictive notion of acyclicity – Berge-acyclicity – is not enough to make containment for conjunctive queries with atomic negation or arithmetic comparisons tractable. To address the second part of the research question, we have defined a special class of queries, called *pointed Berge-acyclic* queries with guarded atomic negation (arithmetic comparisons). This is the class of conjunctive queries with guarded atomic negation (or arithmetic comparisons) where each query

(i) is connected (that is, its hypergraph is connected),

(ii) every query in the class contains a fixed constant r ("pointness"),

(iii) Berge-acyclic.

Theorem 4.3.3 implies that if at least one of these conditions is omitted, containment becomes CONP-hard. It is however not known whether containment for pointed Berge-acyclic queries with negation (comparisons) is solvable in PTIME. On a positive side, we

Class	Complexity
CQ with atomic $\neg$	Π_2^P-c (Ullman, 2000), (Wei and Lausen, 2003)
CQ with comparisons	Π_2^P-c (Klug, 1988), (van der Meyden, 1997)
ACQ($\neg^{g}$) , ACQ with comparisons	coNP-c (Thm. 4.3.2, Thm. 4.3.3)
pointed Berge-ACQs with $\neg^g$ or comparisons	in coNP
child-only Tree patterns with $\neg^g$	PTime (Cor. 4.4.1)
desc.-only Tree patterns with $\neg^g$	PTime (Cor. 4.4.2)
child-only Tree patterns with comparisons	PTime (Cor. 4.4.2)
desc.-only Tree patterns with comparisons	PTime (Cor. 4.4.2)

Table 7.2: Known results and the results of Chapter 4. Here $\neg^g$ denotes guarded atomic negation.

proved that containment for child-only tree patterns with guarded negation, a subclass of pointed Berge-acyclic queries, is solvable in PTime using the homomorphism technique. Unfortunately, this technique is not extensible for pointed Berge-acylic queries with guarded atomic negation (arithmetic comparisons) involving relations of high arity.

In the second part of the thesis we have introduced a new framework for why-not explanations. This framework makes use of ontologies to provide high-level explanations to why certain data is missing from query results. An ontology is a hierarchy of concepts, or a set of subsumption relations between concepts, which formalizes the domain knowledge. In Chapter 6 we have considered two cases how to obtain an ontology: either it is already provided externally (e.g., as a description logic ontology) or it is derived from the database instance or schema with integrity constraints. The latter was formalized in the following research question.

RQ 3 How to extract an ontology from a database instance or a schema?

To answer this question, we first introduced an appropriate concept language $L_{\mathbf{S}}$ comprising nominals (that is, an element of the domain), projections of relation, projections of relations with selections, and conjunctions thereof. Having fixed this language, extracting an ontology from an instance or a schema amounts to decide which subsumptions hold with respect to the instance or schema. Thus we were interested in the following decision problems: given a database schema $\mathbf{S}$ (resp. instance I), two $L_{\mathbf{S}}$ concepts C_1 and C_2, decide whether C_1 is subsumed by C_2 with respect to $\mathbf{S}$ (resp. I), denoted as $C_1 \sqsubseteq_{\mathbf{S}} C_2$ ($C_1 \sqsubseteq_I C_2$). While subsumption with respect to an instance can be trivially solved in PTime, the complexity of subsumption w.r.t. a schema $\mathbf{S}$ depends on the type of integrity constraints in $\mathbf{S}$.

Table 7.3 summarizes the complexity results for subsumption w.r.t. the database schema $\mathbf{S}$, where $L_{\mathbf{S}}^{\min}$ is the language consisting of only the top concept, nominals and projections of relations. In Chapter 6 we were interested in "good" explanations, where by "good" we mean most general explanations (w.r.t. the ontology). Intuitively, these are concepts that cover the maximal number of missing answers and thus are high-level. We asked the following

Constraints	Complexity of subsumption for $L_{\mathbf{S}}$
UCQ-view def. (no comparisons)	NP-complete
UCQ-view def.	Π_2^P-complete
linearly nested UCQ-view def.	Π_2^P-complete
nested UCQ-view def.	coNExpTime-complete
Functional Dependencies (FDs)	in PTime
Inclusion Dependencies (IDs)	? (in PTime for selection-free $L_{\mathbf{S}}$)
IDs + FDs	Undecidable

All stated lower bounds already hold for $L_{\mathbf{S}}^{\min}$ concept expressions.

Table 7.3: Complexity of concept subsumption w.r.t schema $\mathbf{S}$.

RQ 4 How to produce "good" explanations?

Related to this research question, we considered three algorithmic problems: check if an explanation exists, check if a given concept is a most general explanation, and compute one most general explanation. These problems largely depend on how the ontology is obtained. If an ontology is provided externally, then we can check in PTime if a concept is a most general explanation. However, checking existence of an explanation is an NP-complete problem. In light of this result we provide an algorithm for computing one most general explanation, which runs in exponential time.

When the ontology is derived from the instance or schema, existence of an explanation is trivial, because of the presence of nominals in the concept language. We have provided algorithms for checking if a concept is a most general explanation, whose running times depend on the complexity of the subsumption problem (w.r.t. a schema they are provided in Table 7.3). As for computing a most general explanation, we have provided an algorithm for the case of an ontology derived from the instance, which runs in PTime. We have also provided a naive algorithm (running in exponential time) for computing a most general explanation in case the ontology is derived from the schema. Devising an optimal algorithm for this case is left for future work.

7.2 Future work

We list concrete open questions that stem from the work presented in this thesis.

In Chapter 3 we saw that for the coNP hardness proof of containment for tree patterns without the wildcard and with attribute value comparisons, both types of comparisons $@_a = c$ and $@_a \neq c$ are needed. Moreover, if we only allow attribute value comparisons of type $@_a = c$, then containment becomes solvable in PTime. It is an open problem what the exact complexity of containment is for tree patterns without the wildcard and with comparisons of the form $@_a \neq c$ only.

In Chapter 4 we showed a number of coNP hardness proofs that led us to define a class of pointed Berge-acyclic conjunctive queries with guarded negation. We showed that for a particular subclass of it, namely child-only tree patterns with label negation, containment is tractable. However, it is an open question what the precise complexity of the containment problem for pointed Berge-acyclic queries with guarded negation is. Also in that chapter we showed that containment for α-acyclic queries with guarded negation where the arity of negated atoms is bounded by a constant, is in coNP. It is an open question whether containment is still in coNP or Π_2^P-hard if there is no bound on arity of negated atoms.

In Chapter 5 we studied conditional tree patterns, which is the positive downward fragment of Conditional XPath without disjunction and union (Marx, 2005). We can also consider more expressive tree patterns – the positive downward fragment of Regular XPath without disjunction and union (ten Cate, 2006). Pictorially, these tree patterns can have regular expressions as the labels for descendant edges. The conditional descendant that is labeled with Q and ending with P can be expressed with regular tree patterns as $/(\texttt{child} :: \texttt{Q})^*/\texttt{child} :: \texttt{P}$. The complexity of containment for regular tree patterns thus must be between PSPACE and EXPTIME. It is open what the precise complexity is.

Expressivity characterization with respect to some yardstick logics (such as **FO**) is important. Such a characterization allows us to compare different fragments and derive properties from the known ones in the yardstick logics. In Chapter 5 we gave an expressivity characterization for unions of CTP with disjunction, similar to the one for the union of tree patterns in (Benedikt et al., 2005). It is interesting to determine what the exact **FO** fragment corresponding to Conditional Tree Patterns is.

In Chapter 6 we have introduced a new why-not framework, which is neither query-centric nor data-centric, in the sense that it does not suggest fixes for the query or the underlying data. It would be interesting to explore ways how high-level explanations produced by our framework can complement and enhance the existing approaches. Ultimately, it would be interesting to see usefulness of high-level explanations in practice. A concrete tool for generating these explanations should then be built.

An algorithm for generating all most general explanations (MGEs) provided in Chapter 6 is an exhaustive search algorithm and rather inefficient. It would be interesting to develop efficient algorithms for *enumeration* of all most general explanations. Typically, in such a setting with exponential number of output solutions, one is interested in algorithms with *polynomial delay*. That is, the time needed to produce the first solution, and, thereafter, the time delay between two consecutive solutions is bounded by a polynomial in the size of the input. In future work, we plan to investigate whether there is a polynomial delay algorithm for enumerating all most-general explanations.

Bibliography

S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley, 1995. (Cited on pages 4, 10, 15, 108, and 138.)

F. N. Afrati, C. Li, and P. Mitra. On containment of conjunctive queries with arithmetic comparisons. In *EDBT*, pages 459–476, 2004. (Cited on page 14.)

F. N. Afrati, S. Cohen, and G. M. Kuper. On the complexity of tree pattern containment with arithmetic comparisons. *Inf. Process. Lett.*, 111(15):754–760, 2011. (Cited on pages 19 and 28.)

A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2): 218–246, 1979. (Cited on page 15.)

S. Amer-Yahia, S. Cho, L. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *The VLDB Journal*, 11:315–331, 2002. ISSN 1066-8888. URL `http://dx.doi.org/10.1007/s00778-002-0076-7`. (Cited on pages 3, 19, 50, 73, 75, 77, 82, and 83.)

M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and implementation of the logicblox system. In *SIGMOD '15*, 2015. (Cited on page 104.)

T. Arora, R. Ramakrishnan, W. G. Roth, P. Seshadri, and D. Srivastava. Explaining program execution in deductive systems. In *DOOD*, pages 101–119, 1993. (Cited on page 104.)

A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite family and relations. *J. Artif. Intell. Res. (JAIR)*, 36:1–69, 2009. (Cited on page 117.)

S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-weight linked data publication from relational databases. In *WWW*, pages 621–630, 2009. (Cited on page 107.)

A. Baid, W. Wu, C. Sun, A. Doan, and J. F. Naughton. On debugging non-answers in keyword search systems. In *EDBT*, 2015. (Cited on page 104.)

V. Bárány, B. ten Cate, and L. Segoufin. Guarded negation. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 356–367. Springer, 2011. ISBN 978-3-642-22011-1. (Cited on pages 52 and 75.)

M. Benedikt and G. Gottlob. The impact of virtual views on containment. *PVLDB*, 3(1):297–308, 2010. (Cited on pages 12, 109, and 133.)

M. Benedikt and C. Koch. Xpath leashed. *ACM Comput. Surv.*, 41(1), 2009. (Cited on pages 3 and 19.)

M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. *Theor. Comput. Sci.*, 336(1): 3–31, 2005. (Cited on pages 14, 73, 75, 84, and 157.)

M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008. (Cited on pages 3 and 19.)

C. Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd., Oxford, UK, UK, 1985. ISBN 0720404797. (Cited on page 55.)

D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. *VLDB J.*, 14(4):373–396, 2005. (Cited on page 22.)

N. Bidoit, M. Herschel, and K. Tzompanaki. Query-based why-not provenance with nedexplain. In *EDBT*, pages 145–156, 2014a. (Cited on pages 22 and 104.)

N. Bidoit, M. Herschel, and K. Tzompanaki. Immutably answering why-not questions for equivalent conjunctive queries. In *TaPP'14*, 2014b. (Cited on page 22.)

M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. In *PODS*, pages 213–224, 2013. (Cited on pages 23 and 107.)

C. Bizer and A. Seaborne. D2rq - treating non-rdf databases as virtual rdf graphs. In *ISWC2004 (posters)*, 2004. (Cited on page 107.)

H. Björklund, W. Martens, and T. Schwentick. Optimizing conjunctive queries over trees using schema information. In *MFCS*, pages 132–143, 2008. (Cited on page 20.)

H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. *J. Comput. Syst. Sci.*, 77(3):450–472, 2011. (Cited on pages 4, 6, 20, 27, 29, 33, 34, 46, 98, and 153.)

P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001. (Cited on page 82.)

A. Borgida, D. Calvanese, and M. Rodriguez-Muro. Explanation in the DL-Lite family of description logics. In *On the Move to Meaningful Internet Systems*, pages 1440–1457, 2008. (Cited on page 107.)

P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *In ICDT*, 2001. (Cited on pages 21 and 22.)

P. Buneman, S. Khanna, and W. C. Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002. (Cited on page 22.)

D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000. (Cited on pages 77 and 98.)

D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. of Automated reasoning*, 39(3):385–429, 2007. (Cited on pages 113, 115, and 131.)

D. Calvanese, G. D. Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Transactions on Computational Logic (TOCL)*, 9(3):22, 2008. (Cited on page 15.)

D. Calvanese, M. Ortiz, M. Simkus, and G. Stefanoni. Reasoning about explanations for negative query answers in DL-Lite. *J. Artif. Intell. Res.*, 48:635–669, 2013. (Cited on pages 23 and 107.)

A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 77–90, 1977a. (Cited on pages 1, 3, 4, and 51.)

A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977b. (Cited on pages 4, 10, and 12.)

A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. on Computing*, 14(3):671–677, 1985. (Cited on page 137.)

A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009. (Cited on pages 22 and 104.)

S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. *J. Comput. Syst. Sci.*, 54(1):61–78, 1997. (Cited on page 14.)

C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000. (Cited on pages 1 and 13.)

J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009. (Cited on pages 20, 22, and 104.)

B. Chin, D. von Dincklage, V. Ercegovak, P. Hawkins, M. S. Miller, F. Och, C. Olston, and F. Pereira. Yedalog: Exploring knowledge at scale. In *SNAPL*, 2015. (Cited on page 104.)

B. S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986. (Cited on pages 48 and 92.)

E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Prog. Lang. Syst.*, 8:244–263, 1986. (Cited on page 77.)

E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6): 377–387, 1970. (Cited on page 9.)

S. Cosmadakis and P. Kanellakis. Parallel evaluation of recursive rule queries. PODS '86, 1986. (Cited on page 14.)

Y. Cui and J. Widom. Lineage tracing in a data warehousing system. In *ICDE*, pages 683–684, 2000. (Cited on page 22.)

Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 2000. (Cited on pages 20 and 22.)

W. Czerwinski, W. Martens, P. Parys, and M. Przybylko. The (almost) complete guide to tree pattern containment. In *PODS*, pages 117–130, 2015. (Cited on page 19.)

A. Deutsch and V. Tannen. Containment and integrity constraints for XPath. In M. Lenzerini, D. Nardi, W. Nutt, and D. Suciu, editors, *KRDB*, volume 45 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001. (Cited on page 19.)

A. Deutsch and V. Tannen. XML queries and constraints, containment and reformulation. *Theor. Comput. Sci.*, 336(1):57–87, 2005. (Cited on page 19.)

F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *EDBT*, pages 561–572, 2013. (Cited on pages 113 and 115.)

G. Dong and J. Su. Conjunctive query containment with respect to views and constraints. *Inf. Process. Lett.*, 57(2), Jan. 1996. (Cited on page 15.)

A. Facchini, Y. Hirai, M. Marx, and E. Sherkhonov. Containment for conditional tree patterns. *Logical Methods in Computer Science*, 11(2), 2015. (Cited on pages 8 and 48.)

R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983. (Cited on pages 13, 52, and 55.)

C. Farré, W. Nutt, E. Teniente, and T. Urpí. Containment of conjunctive queries over databases with null values. In *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings*, pages 389–403, 2007. (Cited on page 14.)

G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001. (Cited on pages 1, 3, 5, 13, 51, 52, 54, 59, and 154.)

G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002. (Cited on page 13.)

G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005a. (Cited on pages 16, 27, 39, and 80.)

G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *J. ACM*, 52(2), Mar. 2005b. (Cited on page 3.)

G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006. (Cited on pages 20 and 31.)

T. J. Green. Logiql: a declarative language for enterprise applications. In *PODS '15*, 2015. (Cited on page 104.)

T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, pages 675–686, 2007a. (Cited on page 22.)

T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007b. (Cited on pages 21, 22, and 104.)

T. J. Green, M. Aref, and G. Karvounarakis. Logicblox, platform and language: A tutorial. In *Proceedings of the Second International Conference on Datalog in Academia and Industry*, pages 1–8, 2012. (Cited on page 104.)

M. Götz, C. Koch, and W. Martens. Efficient algorithms for the tree homeomorphism problem. In *In DBPL*, pages 17–31, 2007. (Cited on page 69.)

A. Y. Halevy. Theory of answering queries using views. *SIGMOD Rec.*, 29(4), Dec. 2000. (Cited on page 1.)

T. Halpin and S. Rugaber. *LogiQL: A Query Language for Smart Databases*. CRC Press, 2014. (Cited on page 104.)

Z. He and E. Lo. Answering why-not questions on top-k queries. ICDE '12, 2012. (Cited on page 22.)

P. Hell and J. Nesetril. *Graphs and Homomorphisms*. Oxford University Press, 2004. (Cited on page 4.)

M. Herschel and M. A. Hernández. Explaining missing answers to SPJUA queries. *PVLDB*, 3(1):185–196, 2010. (Cited on page 23.)

M. Herschel, M. A. Hernández, and W. C. Tan. Artemis: A system for analyzing missing answers. *PVLDB*, 2 (2):1550–1553, 2009. (Cited on page 104.)

J. Hidders. Satisfiability of XPath expressions. In G. Lausen and D. Suciu, editors, *DBPL*, volume 2921 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2003. (Cited on pages 3 and 19.)

J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008a. (Cited on page 104.)

J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008b. (Cited on page 23.)

M. S. Islam, R. Zhou, and C. Liu. On answering why-not questions in reverse skyline queries. In *ICDE 2013*, 2013. (Cited on page 22.)

D. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of computer and system sciences*, 28(1):167–189, 1984. (Cited on page 15.)

H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968. (Cited on page 4.)

B. Kimelfeld and Y. Sagiv. Revisiting redundancy and minimization in an XPath fragment. In *EDBT'08*, pages 61–72, 2008. (Cited on page 86.)

A. C. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988. (Cited on pages 14, 51, 53, 57, and 155.)

P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000. (Cited on page 4.)

E. V. Kostylev, J. L. Reutter, and D. Vrgoc. Containment of data graph queries. In *ICDT'2014*, pages 131–142, 2014. (Cited on page 1.)

O. Kupferman and M. Y. Vardi. An automata-theoretic approach to modular model checking. *ACM Trans. Prog. Lang. Syst.*, 22(1):87–128, 2000. (Cited on pages 75, 77, 80, and 94.)

A. Y. Levy, A. O. Mendelzon, and Y. Sagiv. Answering queries using views. PODS '95, 1995. (Cited on page 1.)

L. Libkin and C. Sirangelo. Reasoning about XML with temporal logics and automata. *J. Applied Logic*, 8(2): 210–232, 2010. (Cited on page 74.)

L. Lubyte and S. Tessaris. Automatic extraction of ontologies wrapping relational data sources. In *DEXA*, pages 128–142, 2009. (Cited on page 107.)

M. Marx. Conditional XPath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005. (Cited on pages 17, 18, 49, 74, 77, 80, 154, and 157.)

M. Marx and E. Sherkhonov. Containment for queries over trees with attribute value comparisons. *Information Systems*, Forthcoming, 2015. (Cited on pages 8 and 97.)

A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010. (Cited on page 104.)

G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004. (Cited on pages 1, 3, 18, 19, 20, 27, 46, 50, 55, 60, 64, 66, 73, 75, 76, 77, 78, 81, 82, 83, 87, 88, 97, and 98.)

J. C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56(3):154 – 173, 1983. (Cited on page 137.)

F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006. (Cited on pages 19, 48, 75, 76, 77, 81, and 92.)

W. Nutt. Ontology and database systems: Foundations of database systems. 2013. Teaching material. `http://www.inf.unibz.it/~nutt/Teaching/ODBS1314/ODBSSlides/3-conjQueries.pdf`. (Cited on pages 14, 57, and 133.)

A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics X*, pages 133–173, 2008. (Cited on pages 23, 106, 107, 114, 115, and 131.)

S. Roy and D. Suciu. A formal approach to finding explanations for database queries. SIGMOD '14, 2014. (Cited on page 129.)

Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *JACM*, 27(4):633–655, 1980. (Cited on page 11.)

E. Sherkhonov and M. Marx. Containment for tree patterns with attribute value comparisons. In *Proceedings of the 16th International Workshop on the Web and Databases 2013, WebDB 2013.*, pages 7–12, 2013. (Cited on page 8.)

E. Sherkhonov and M. Marx. Containment of acyclic conjunctive queries with atomic negation and arithmetic comparisons. *Submitted to a journal*, 2015. (Cited on page 8.)

O. Shmueli. Equivalence of DATALOG queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993. (Cited on pages 14 and 133.)

O. Shmueli and S. Tsur. Logical diagnosis of ldl programs. In *Int'l Conf. on Logic Programming*, 1990. (Cited on pages 104 and 105.)

B. ten Cate. The expressivity of XPath with transitive closure. In *PODS*, pages 328–337, 2006. (Cited on pages 17, 78, 86, 98, and 157.)

B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2.0. *J. ACM*, 56(6), 2009. (Cited on page 49.)

B. ten Cate, C. Civili, E. Sherkhonov, and W. Tan. High-level why-not explanations using ontologies. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 31–43, 2015. (Cited on pages 2 and 8.)

Q. T. Tran and C. Chan. How to conquer why-not questions. In *SIGMOD*, pages 15–26, 2010. (Cited on pages 22 and 104.)

J. D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000. (Cited on pages 5, 6, 14, 51, 53, and 155.)

R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.*, 54(1):113–135, 1997. (Cited on pages 5, 14, 51, 53, 57, and 155.)

P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker Inc., 1997. (Cited on page 92.)

W3C. XML Schema recommendation. *`http://www.w3.org/XML/Schema`*. (Cited on page 3.)

W3C. XML path language (XPath) recommendation. *`http://www.w3c.org/TR/xpath/`*, 1999a. (Cited on page 3.)

W3C. XSL transformations (XSLT) recommendation. *`http://www.w3.org/TR/xslt`*, 1999b. (Cited on page 3.)

W3C. XQuery 1.0: An XML Query Language recommendation. *`http://www.w3.org/TR/xquery/`*, 2010. (Cited on page 3.)

F. Wei and G. Lausen. Containment of conjunctive queries with safe negation. In *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings*, pages 343–357, 2003. (Cited on pages 6, 14, 51, 53, 57, and 155.)

P. T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT'2003*, pages 297–311, 2003. (Cited on pages 1 and 19.)

M. Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94, 1981. (Cited on pages 13, 51, and 59.)

Samenvatting

Het eerste deel van dit proefschrift gaat over het beslissen of de ene databank-vraag (*query*) een logisch gevolg is van een andere vraag. De analyse van de complexiteit van dit probleem voor verschilende famillies van vraagtalen vormt één van de centrale problemen binnen de theorie van databanken. De ene vraag volgt logisch uit de andere als voor elke mogelijke databank, alle antwoorden op de eerste vraag ook antwoorden op de tweede vraag zijn. Dit probleem vindt onder meer zijn toepassing in het optimaliseren van databankvragen, het beantwoorden van vragen aan een databank met behulp van zogenaamde *views* en het beantwoorden van vragen onder *constraints* op de databank. Omdat dit probleem zo centraal staat, is het in het verleden uitgebreid bestudeerd voor meerdere querytalen en datamodellen.

In dit proefschrift vervolgen wij dit onderzoek voor diverse acyclische vraagtalen geïnterpreteerd op bomen en relationele databanken. Specifiek bekijken wij de volgende drie talen:

1) Conjunctieve en positieve XPath queries uitgebreid met negatie en met attribuut-waarde vergelijkingen, geïnterpreteerd op bomen;

2) *Tree patterns* uitgebreid met een conditionele *descendant* modaliteit , geïnterpreteerd op bomen; en

3) Acyclische conjunctieve queries uitgebreid met atomaire negatie en rekenkundige vergelijkingen geïnterpreteerd op relationele structuren.

We bewijzen de precieze complexiteit voor het beslissen van de logisch gevolg relatie voor deze talen.

Het tweede deel van dit proefschrift stelt een nieuw formeel raamwerk rondom zogeheten “waarom-niet verklaringen” voor. Onze waarom-niet verklaringen maken gebruik van concepten uit een ontologie om op hoog niveau en betekenisvol verklaringen te geven voor het feit dat bepaalde feiten ontbreken in de antwoorden op een query. In dit raamwerk zijn we geïnteresseerd in het vinden van de meest algemene verklaring in relatie tot een ontologie. Eerst richten we ons op het probleem van het extraheren van een ontologie uit een databaseinstantie of databaseschema. Wij tonen aan dat dit probleem gezien kan worden als het eerder genoemde probleem van logisch gevolg en verschaffen een complexiteitsanalyze. Daarna richten we ons op het probleem om de meest algemene verklaring te berekenen. We bestuderen de complexiteit van dit probleem en van vergelijkbare problemen. We geven tenslotte een algorithme voor het berekenen van waarom-niet verklaringen.

SIKS Dissertation Series

1998

1 Johan van den Akker (CWI) *DEGAS — An Active, Temporal Database of Autonomous Objects*

2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*

3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

4 Dennis Breuker (UM) *Memory versus Search in Games*

5 Eduard W. Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

1999

1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated Modelling of Quality Change of Agricultural Products*

2 Rob Potharst (EUR) *Classification using Decision Trees and Neural Nets*

3 Don Beal (UM) *The Nature of Minimax Search*

4 Jacques Penders (UM) *The Practical Art of Moving Physical Objects*

5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

6 Niek J.E. Wijngaards (VU) *Re-Design of Compositional Systems*

7 David Spelt (UT) *Verification Support for Object Database Design*

8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*

2000

1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*

2 Koen Holtman (TU/e) *Prototyping of CMS Storage Management*

3 Carolien M.T. Metselaar (UvA) *Sociaal-organisatorische Gevolgen van Kennistechnologie; een Procesbenadering en Actorperspectief*

4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*

5 Ruud van der Pol (UM) *Knowledge-Based Query Formulation in Information Retrieval*

6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*

7 Niels Peek (UU) *Decision-Theoretic Planning of Clinical Patient Management*

8 Veerle Coupé (EUR) *Sensitivity Analysis of Decision-Theoretic Networks*

9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*

10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*

11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

2001

1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*

2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*

3 Maarten van Someren (UvA) *Learning as Problem Solving*

4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*

5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*

6 Martijn van Welie (VU) *Task-Based User Interface Design*

7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*

8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*

9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice BRAHMS: a Multiagent Modeling and Simulation Language for Work Practice Analysis and Design*

11 Tom M. van Engers (VU) *Knowledge Management: The Role of Mental Models in Business Systems Design*

2002

1 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*

2 Roelof van Zwol (UT) *Modelling and Searching Web-based Document Collections*

3 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*

4 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*

5 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments Inhabited by Privacy-Concerned Agents*

6 Laurens Mommers (UL) *Applied Legal Epistemology; Building a Knowledge-based Ontology of the Legal Domain*

7 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*

8 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

9 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*

10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*

11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*

12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*

13 Hongjing Wu (TU/e) *A Reference Architecture for Adaptive Hypermedia Applications*

14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*

15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*

16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*

17 Stefan Manegold (UvA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003

1 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*

2 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*

3 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*

4 Milan Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*

5 Jos Lehmann (UvA) *Causation in Artificial Intelligence and Law – A Modelling Approach*

6 Boris van Schooten (UT) *Development and Specification of Virtual Environments*

7 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*

8 Yong-Ping Ran (UM) *Repair-Based Scheduling*

9 Rens Kortmann (UM) *The Resolution of Visually Guided Behaviour*

10 Andreas Lincke (UT) *Electronic Business Negotiation: Some Experimental Studies on the Interaction between Medium, Innovation Context and Cult*

11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*

12 Roeland Ordelman (UT) *Dutch Speech Recognition in Multimedia Information Retrieval*

13 Jeroen Donkers (UM) *Nosce Hostem – Searching with Opponent Models*

14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*

15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*

16 Menzo Windhouwer (CWI) *Feature Grammar Systems — Incremental Maintenance of Indexes to Digital Media Warehouse*

17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*

18 Levente Kocsis (UM) *Learning Search Decisions*

2004

1 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*

2 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*

3 Perry Groot (VU) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*

4 Chris van Aart (UvA) *Organizational Principles for Multi-Agent Architectures*

5 Viara Popova (EUR) *Knowledge Discovery and Monotonicity*

6 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*

7 Elise Boltjes (UM) *Voorbeeld$_{IG}$ Onderwijs; Voorbeeldgestuurd Onderwijs, een Opstap naar Abstract Denken, vooral voor Meisjes*

8 Joop Verbeek (UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale Politiële Gegevensuitwisseling en Digitale Expertise*

9 Martin Caminada (VU) *For the Sake of the Argument; Explorations into Argument-based Reasoning*

10 Suzanne Kabel (UvA) *Knowledge-rich Indexing of Learning-objects*

11 Michel Klein (VU) *Change Management for Distributed Ontologies*

12 The Duy Bui (UT) *Creating Emotions and Facial Expressions for Embodied Agents*

13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*

14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*

15 Arno Knobbe (UU) *Multi-Relational Data Mining*

16 Federico Divina (VU) *Hybrid Genetic Relational Search for Inductive Learning*

17 Mark Winands (UM) *Informed Search in Complex Games*

18 Vania Bessa Machado (UvA) *Supporting the Construction of Qualitative Knowledge Models*
19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*
20 Madelon Evers (Nyenrode) *Learning from Design: facilitating multidisciplinary design teams*

2005

1 Floor Verdenius (UvA) *Methodological Aspects of Designing Induction-Based Applications*
2 Erik van der Werf (UM) *AI techniques for the game of Go*
3 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*
4 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*
5 Gabriel Infante-Lopez (UvA) *Two-Level Probabilistic Grammars for Natural Language Parsing*
6 Pieter Spronck (UM) *Adaptive Game AI*
7 Flavius Frasincar (TU/e) *Hypermedia Presentation Generation for Semantic Web Information Systems*
8 Richard Vdovjak (TU/e) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*
9 Jeen Broekstra (VU) *Storage, Querying and Inferencing for Semantic Web Languages*
10 Anders Bouwer (UvA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*
11 Elth Ogston (VU) *Agent Based Matchmaking and Clustering — A Decentralized Approach to Search*
12 Csaba Boer (EUR) *Distributed Simulation in Industry*
13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*
14 Borys Omelayenko (VU) *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*
15 Tibor Bosse (VU) *Analysis of the Dynamics of Cognitive Processes*
16 Joris Graaumans (UU) *Usability of XML Query Languages*
17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*
18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*
19 Michel van Dartel (UM) *Situated Representation*
20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*
21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006

1 Samuil Angelov (TU/e) *Foundations of B2B Electronic Contracting*
2 Cristina Chisalita (VU) *Contextual issues in the design and use of information technology in organizations*
3 Noor Christoph (UvA) *The role of metacognitive skills in learning to solve problems*
4 Marta Sabou (VU) *Building Web Service Ontologies*
5 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*
6 Ziv Baida (VU) *Software-aided Service Bundling — Intelligent Methods & Tools for Graphical Service Modeling*
7 Marko Smiljanic (UT) *XML schema matching – balancing efficiency and effectiveness by means of clustering*
8 Eelco Herder (UT) *Forward, Back and Home Again — Analyzing User Behavior on the Web*
9 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*
10 Ronny Siebes (VU) *Semantic Routing in Peer-to-Peer Systems*
11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*
12 Bert Bongers (VU) *Interactivation — Towards an e-cology of people, our technological environment, and the arts*
13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*
14 Johan Hoorn (VU) *Software Requirements: Update, Upgrade, Redesign — towards a Theory of Requirements Change*
15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*
16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*
17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*
18 Valentin Zhizhkun (UvA) *Graph transformation for Natural Language Processing*
19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*
20 Marina Velikova (UvT) *Monotone models for prediction in data mining*
21 Bas van Gils (RUN) *Aptness on the Web*
22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*
23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*
24 Laura Hollink (VU) *Semantic Annotation for Retrieval of Visual Resources*
25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*

26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*
27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*
28 Borkur Sigurbjornsson (UvA) *Focused Information Access using XML Element Retrieval*

2007

1 Kees Leune (UvT) *Access Control and Service-Oriented Architectures*
2 Wouter Teepe (RUG) *Reconciling Information Exchange and Confidentiality: A Formal Approach*
3 Peter Mika (VU) *Social Networks and the Semantic Web*
4 Jurriaan van Diggelen (UU) *Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach*
5 Bart Schermer (UL) *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*
6 Gilad Mishne (UvA) *Applied Text Analytics for Blogs*
7 Natasa Jovanovic' (UT) *To Whom It May Concern - Addressee Identification in Face-to-Face Meetings*
8 Mark Hoogendoorn (VU) *Modeling of Change in Multi-Agent Organizations*
9 David Mobach (VU) *Agent-Based Mediated Service Negotiation*
10 Huib Aldewereld (UU) *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*
11 Natalia Stash (TU/e) *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*
12 Marcel van Gerven (RUN) *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*
13 Rutger Rienks (UT) *Meetings in Smart Environments; Implications of Progressing Technology*
14 Niek Bergboer (UM) *Context-Based Image Analysis*
15 Joyca Lacroix (UM) *NIM: a Situated Computational Memory Model*
16 Davide Grossi (UU) *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems*
17 Theodore Charitos (UU) *Reasoning with Dynamic Networks in Practice*
18 Bart Orriens (UvT) *On the development and management of adaptive business collaborations*
19 David Levy (UM) *Intimate relationships with artificial partners*
20 Slinger Jansen (UU) *Customer Configuration Updating in a Software Supply Network*
21 Karianne Vermaas (UU) *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*
22 Zlatko Zlatev (UT) *Goal-oriented design of value and process models from patterns*
23 Peter Barna (TU/e) *Specification of Application Logic in Web Information Systems*
24 Georgina Ramírez Camps (CWI) *Structural Features in XML Retrieval*
25 Joost Schalken (VU) *Empirical Investigations in Software Process Improvement*

2008

1 Katalin Boer-Sorbán (EUR) *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*
2 Alexei Sharpanskykh (VU) *On Computer-Aided Methods for Modeling and Analysis of Organizations*
3 Vera Hollink (UvA) *Optimizing hierarchical menus: a usage-based approach*
4 Ander de Keijzer (UT) *Management of Uncertain Data — towards unattended integration*
5 Bela Mutschler (UT) *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*
6 Arjen Hommersom (RUN) *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective*
7 Peter van Rosmalen (OU) *Supporting the tutor in the design and support of adaptive e-learning*
8 Janneke Bolt (UU) *Bayesian Networks: Aspects of Approximate Inference*
9 Christof van Nimwegen (UU) *The paradox of the guided user: assistance can be counter-effective*
10 Wauter Bosma (UT) *Discourse oriented Summarization*
11 Vera Kartseva (VU) *Designing Controls for Network Organizations: a Value-Based Approach*
12 Jozsef Farkas (RUN) *A Semiotically oriented Cognitive Model of Knowledge Representation*
13 Caterina Carraciolo (UvA) *Topic Driven Access to Scientific Handbooks*
14 Arthur van Bunningen (UT) *Context-Aware Querying; Better Answers with Less Effort*
15 Martijn van Otterlo (UT) *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains*
16 Henriette van Vugt (VU) *Embodied Agents from a User's Perspective*

17 Martin Op't Land (TUD) *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*
18 Guido de Croon (UM) *Adaptive Active Vision*
19 Henning Rode (UT) *From document to entity retrieval: improving precision and performance of focused text search*
20 Rex Arendsen (UvA) *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met een overheid op de administratieve lasten van bedrijven*
21 Krisztian Balog (UvA) *People search in the enterprise*
22 Henk Koning (UU) *Communication of IT-architecture*
23 Stefan Visscher (UU) *Bayesian network models for the management of ventilator-associated pneumonia*
24 Zharko Aleksovski (VU) *Using background knowledge in ontology matching*
25 Geert Jonker (UU) *Efficient and Equitable exchange in air traffic management plan repair using spender-signed currency*
26 Marijn Huijbregts (UT) *Segmentation, diarization and speech transcription: surprise data unraveled*
27 Hubert Vogten (OU) *Design and implementation strategies for IMS learning design*
28 Ildiko Flesh (RUN) *On the use of independence relations in Bayesian networks*
29 Dennis Reidsma (UT) *Annotations and subjective machines- Of annotators, embodied agents, users, and other humans*
30 Wouter van Atteveldt (VU) *Semantic network analysis: techniques for extracting, representing and querying media content*
31 Loes Braun (UM) *Pro-active medical information retrieval*
32 Trung B. Hui (UT) *Toward affective dialogue management using partially observable markov decision processes*
33 Frank Terpstra (UvA) *Scientific workflow design; theoretical and practical issues*
34 Jeroen de Knijf (UU) *Studies in Frequent Tree Mining*
35 Benjamin Torben-Nielsen (UvT) *Dendritic morphology: function shapes structure*

2009

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*
2 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*
3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*
4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*
5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks — Based on Knowledge, Cognition, and Quality*
6 Muhammad Subianto (UU) *Understanding Classification*
7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*
8 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*
9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*
10 Jan Wielemaker (UvA) *Logic programming for knowledge-intensive interactive applications*
11 Alexander Boer (UvA) *Legal Theory, Sources of Law & the Semantic Web*
12 Peter Massuthe (TU/e, Humboldt-Universtät zu Berlin) *Operating Guidelines for Services*
13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*
14 Maksym Korotkiy (VU) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*
15 Rinke Hoekstra (UvA) *Ontology Representation — Design Patterns and Ontologies that Make Sense*
16 Fritz Reul (UvT) *New Architectures in Computer Chess*
17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*
18 Fabian Groffen (CWI) *Armada, An Evolving Database System*
19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*
20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*
21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*
22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*
23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*
24 Annerieke Heuvelink (VU) *Cognitive Models for Training Simulations*
25 Alex van Ballegooij (CWI) *"RAM: Array Database Management through Relational Mapping"*
26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*
27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*
28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*
29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*

30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*
31 Sofiya Katrenko (UvA) *A Closer Look at Learning Relations from Text*
32 Rik Farenhorst and Remco de Boer (VU) *Architectural Knowledge Management: Supporting Architects and Auditors*
33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*
34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*
35 Wouter Koelewijn (UL) *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling*
36 Marco Kalz (OUN) *Placement Support for Learners in Learning Networks*
37 Hendrik Drachsler (OUN) *Navigation Support for Learners in Informal Learning Networks*
38 Riina Vuorikari (OU) *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context*
39 Christian Stahl (TUE, Humboldt-Universität zu Berlin) *Service Substitution – A Behavioral Approach Based on Petri Nets*
40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*
41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*
42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*
43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*
44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*
45 Jilles Vreeken (UU) *Making Pattern Mining Useful*
46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

2010

1 Matthijs van Leeuwen (UU) *Patterns that Matter*
2 Ingo Wassink (UT) *Work flows in Life Science*
3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*
4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*
5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*
6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*
7 Wim Fikkert (UT) *Gesture interaction at a Distance*
8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*
9 Hugo Kielman (UL) *A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging*
10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*
11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*
12 Susan van den Braak (UU) *Sensemaking software for crime analysis*
13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*
14 Sander van Splunter (VU) *Automated Web Service Reconfiguration*
15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*
16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*
17 Spyros Kotoulas (VU) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*
18 Charlotte Gerritsen (VU) *Caught in the Act: Investigating Crime by Agent-Based Simulation*
19 Henriette Cramer (UvA) *People's Responses to Autonomous and Adaptive Systems*
20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*
21 Harold van Heerde (UT) *Privacy-aware data management by means of data degradation*
22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*
23 Bas Steunebrink (UU) *The Logical Structure of Emotions*
24 Dmytro Tykhonov (TUD) *Designing Generic and Efficient Negotiation Strategies*
25 Zulfiqar Ali Memon (VU) *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*
26 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*
27 Marten Voulon (UL) *Automatisch contracteren*
28 Arne Koopman (UU) *Characteristic Relational Patterns*
29 Stratos Idreos (CWI) *Database Cracking: Towards Auto-tuning Database Kernels*
30 Marieke van Erp (UvT) *Accessing Natural History — Discoveries in data cleaning, structuring, and retrieval*
31 Victor de Boer (UvA) *Ontology Enrichment from Heterogeneous Sources on the Web*

32 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*
33 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*
34 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*
35 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*
36 Jose Janssen (OU) *Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification*
37 Niels Lohmann (TUE) *Correctness of services and their composition*
38 Dirk Fahland (TUE) *From Scenarios to components*
39 Ghazanfar Farooq Siddiqui (VU) *Integrative modeling of emotions in virtual agents*
40 Mark van Assem (VU) *Converting and Integrating Vocabularies for the Semantic Web*
41 Guillaume Chaslot (UM) *Monte-Carlo Tree Search*
42 Sybren de Kinderen (VU) *Needs-driven service bundling in a multi-supplier setting — the computational e3-service approach*
43 Peter van Kranenburg (UU) *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*
44 Pieter Bellekens (TUE) *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*
45 Vasilios Andrikopoulos (UvT) *A theory and model for the evolution of software services*
46 Vincent Pijpers (VU) *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*
47 Chen Li (UT) *Mining Process Model Variants: Challenges, Techniques, Examples*
48 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*
49 Jahn-Takeshi Saito (UM) *Solving difficult game positions*
50 Bouke Huurnink (UvA) *Search in Audiovisual Broadcast Archives*
51 Alia Khairia Amin (CWI) *Understanding and supporting information seeking tasks in multiple sources*
52 Peter-Paul van Maanen (VU) *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention*
53 Edgar J. Meij (UvA) *Combining Concepts and Language Models for Information Access*

2011

1 Botond Cseke (RUN) *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*
2 Nick Tinnemeier (UU) *Work flows in Life Science*
3 Jan Martijn van der Werf (TUE) *Compositional Design and Verification of Component-Based Information Systems*
4 Hado van Hasselt (UU) *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms*
5 Base van der Raadt (VU) *Enterprise Architecture Coming of Age — Increasing the Performance of an Emerging Discipline*
6 Yiwen Wang (TUE) *Semantically-Enhanced Recommendations in Cultural Heritage*
7 Yujia Cao (UT) *Multimodal Information Presentation for High Load Human Computer Interaction*
8 Nieske Vergunst (UU) *BDI-based Generation of Robust Task-Oriented Dialogues*
9 Tim de Jong (OU) *Contextualised Mobile Media for Learning*
10 Bart Bogaert (UvT) *Cloud Content Contention*
11 Dhaval Vyas (UT) *Designing for Awareness: An Experience-focused HCI Perspective*
12 Carmen Bratosin (TUE) *Grid Architecture for Distributed Process Mining*
13 Xiaoyu Mao (UvT) *Airport under Control. Multiagent Scheduling for Airport Ground Handling*
14 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*
15 Marijn Koolen (UvA) *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*
16 Maarten Schadd (UM) *Selective Search in Games of Different Complexity*
17 Jiyin He (UvA) *Exploring Topic Structure: Coherence, Diversity and Relatedness*
18 Mark Ponsen (UM) *Strategic Decision-Making in complex games*
19 Ellen Rusman (OU) *The Mind ' s Eye on Personal Profiles*
20 Qing Gu (VU) *Guiding service-oriented software engineering — A view-based approach*
21 Linda Terlouw (TUD) *Modularization and Specification of Service-Oriented System*
22 Junte Zhang (UvA) *System Evaluation of Archival Description and Access*
23 Wouter Weerkamp (UvA) *Finding People and their Utterances in Social Media*
24 Herwin van Welbergen (UT) *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior*
25 Syed Waqar ul Qounain Jaffry (VU) *Analysis and Validation of Models for Trust Dynamics*
26 Matthijs Aart Pontier (VU) *Virtual Agents for Human Communication — Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots*

27 Aniel Bhulai (VU) *Dynamic website optimization through autonomous management of design patterns*

28 Rianne Kaptein (UvA) *Effective Focused Retrieval by Exploiting Query Context and Document Structure*

29 Faisal Kamiran (TUE) *Discrimination-aware Classification*

30 Egon van den Broek (UT) *Affective Signal Processing (ASP): Unraveling the mystery of emotions*

31 Ludo Waltman (EUR) *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*

32 Nees-Jan van Eck (EUR) *Methodological Advances in Bibliometric Mapping of Science*

33 Tom van der Weide (UU) *Arguing to Motivate Decisions*

34 Paolo Turrini (UU) *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations*

35 Maaike Harbers (UU) *Explaining Agent Behavior in Virtual Training*

36 Erik van der Spek (UU) *Experiments in serious game design: a cognitive approach*

37 Adriana Burlutiu (RUN) *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference*

38 Nyree Lemmens (UM) *Bee-inspired Distributed Optimization*

39 Joost Westra (UU) *Organizing Adaptation using Agents in Serious Games*

40 Viktor Clerc (VU) *Architectural Knowledge Management in Global Software Development*

41 Luan Ibraimi (UT) *Cryptographically Enforced Distributed Data Access Control*

42 Michal Sindlar (UU) *Explaining Behavior through Mental State Attribution*

43 Henk van der Schuur (UU) *Process Improvement through Software Operation Knowledge*

44 Boris Reuderink (UT) *Robust Brain-Computer Interfaces*

45 Herman Stehouwer (UvT) *Statistical Language Models for Alternative Sequence Selection*

46 Beibei Hu (TUD) *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work*

47 Azizi Bin Ab Aziz (VU) *Exploring Computational Models for Intelligent Support of Persons with Depression*

48 Mark Ter Maat (UT) *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent*

49 Andreea Niculescu (UT) *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality*

2012

1 Terry Kakeeto (UvT) *Relationship Marketing for SMEs in Uganda*

2 Muhammad Umair (VU) *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models*

3 Adam Vanya (VU) *Supporting Architecture Evolution by Mining Software Repositories*

4 Jurriaan Souer (UU) *Development of Content Management System-based Web Applications*

5 Marijn Plomp (UU) *Maturing Interorganisational Information Systems*

6 Wolfgang Reinhardt (OU) *Awareness Support for Knowledge Workers in Research Networks*

7 Rianne van Lambalgen (VU) *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions*

8 Gerben de Vries (UvA) *Kernel Methods for Vessel Trajectories*

9 Ricardo Neisse (UT) *Trust and Privacy Management Support for Context-Aware Service Platforms*

10 David Smits (TUE) *Towards a Generic Distributed Adaptive Hypermedia Environment*

11 J.C.B. Rantham Prabhakara (TUE) *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*

12 Kees van der Sluijs (TUE) *Model Driven Design and Data Integration in Semantic Web Information Systems*

13 Suleman Shahid (UvT) *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions*

14 Evgeny Knutov (TUE) *Generic Adaptation Framework for Unifying Adaptive Web-based Systems*

15 Natalie van der Wal (VU) *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes*

16 Fiemke Both (VU) *Helping people by understanding them — Ambient Agents supporting task execution and depression treatment*

17 Amal Elgammal (UvT) *Towards a Comprehensive Framework for Business Process Compliance*

18 Eltjo Poort (VU) *Improving Solution Architecting Practices*

19 Helen Schonenberg (TUE) *What's Next? Operational Support for Business Process Execution*

20 Ali Bahramisharif (RUN) *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing*

21 Roberto Cornacchia (TUD) *Querying Sparse Matrices for Information Retrieval*

22 Thijs Vis (UvT) *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?*

23 Christian Muehl (UT) *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction*
24 Laurens van der Werff (UT) *Evaluation of Noisy Transcripts for Spoken Document Retrieval*
25 Silja Eckartz (UT) *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application*
26 Emile de Maat (UvA) *Making Sense of Legal Text*
27 Hayrettin Gurkok (UT) *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games*
28 Nancy Pascall (UvT) *Engendering Technology Empowering Women*
29 Almer Tigelaar (UT) *Peer-to-Peer Information Retrieval*
30 Alina Pommeranz (TUD) *Designing Human-Centered Systems for Reflective Decision Making*
31 Emily Bagarukayo (RUN) *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure*
32 Wietske Visser (TUD) *Qualitative multi-criteria preference representation and reasoning*
33 Rory Sie (OUN) *Coalitions in Cooperation Networks (COCOON)*
34 Pavol Jancura (RUN) *Evolutionary analysis in PPI networks and applications*
35 Evert Haasdijk (VU) *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics*
36 Denis Ssebugwawo (RUN) *Analysis and Evaluation of Collaborative Modeling Processes*
37 Agnes Nakakawa (RUN) *A Collaboration Process for Enterprise Architecture Creation*
38 Selmar Smit (VU) *Parameter Tuning and Scientific Testing in Evolutionary Algorithms*
39 Hassan Fatemi (UT) *Risk-aware design of value and coordination networks*
40 Agus Gunawan (UvT) *Information Access for SMEs in Indonesia*
41 Sebastian Kelle (OU) *Game Design Patterns for Learning*
42 Dominique Verpoorten (OU) *Reflection Amplifiers in self-regulated Learning*
43 (Withdrawn)
44 Anna Tordai (VU) *On Combining Alignment Techniques*
45 Benedikt Kratz (UvT) *A Model and Language for Business-aware Transactions*
46 Simon Carter (UvA) *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation*
47 Manos Tsagkias (UvA) *Mining Social Media: Tracking Content and Predicting Behavior*
48 Jorn Bakker (TUE) *Handling Abrupt Changes in Evolving Time-series Data*
49 Michael Kaisers (UM) *Learning against Learning — Evolutionary dynamics of reinforcement learning algorithms in strategic interactions*
50 Steven van Kervel (TUD) *Ontologogy driven Enterprise Information Systems Engineering*
51 Jeroen de Jong (TUD) *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching*

2013

1 Viorel Milea (EUR) *News Analytics for Financial Decision Support*
2 Erietta Liarou (CWI) *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing*
3 Szymon Klarman (VU) *Reasoning with Contexts in Description Logics*
4 Chetan Yadati (TUD) *Coordinating autonomous planning and scheduling*
5 Dulce Pumareja (UT) *Groupware Requirements Evolutions Patterns*
6 Romulo Goncalves (CWI) *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience*
7 Giel van Lankveld (UT) *Quantifying Individual Player Differences*
8 Robbert-Jan Merk (VU) *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators*
9 Fabio Gori (RUN) *Metagenomic Data Analysis: Computational Methods and Applications*
10 Jeewanie Jayasinghe Arachchige (UvT) *A Unified Modeling Framework for Service Design*
11 Evangelos Pournaras (TUD) *Multi-level Reconfigurable Self-organization in Overlay Services*
12 Maryam Razavian (VU) *Knowledge-driven Migration to Services*
13 Mohammad Zafiri (UT) *Service Tailoring: User-centric creation of integrated IT-based homecare*
14 Jafar Tanha (UvA) *Ensemble Approaches to Semi-Supervised Learning*
15 Daniel Hennes (UM) *Multiagent Learning — Dynamic Games and Applications*
16 Eric Kok (UU) *Exploring the practical benefits of argumentation in multi-agent deliberation*
17 Koen Kok (VU) *The PowerMatcher: Smart Coordination for the Smart Electricity Grid*
18 Jeroen Janssens (UvT) *Outlier Selection and One-Class Classification*
19 Renze Steenhuizen (TUD) *Coordinated Multi-Agent Planning and Scheduling*
20 Katja Hofmann (UvA) *Fast and Reliable Online Learning to Rank for Information Retrieval*
21 Sander Wubben (UvT) *Text-to-text generation by monolingual machine translation*

22 Tom Claassen (RUN) *Causal Discovery and Logic*
23 Patricio de Alencar Silva (UvT) *Value Activity Monitoring*
24 Haitham Bou Ammar (UM) *Automated Transfer in Reinforcement Learning*
25 Agnieszka Anna Latoszek-Berendsen (UM) *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System*
26 Alireza Zarghami (UT) *Architectural Support for Dynamic Homecare Service Provisioning*
27 Mohammad Huq (UT) *Inference-based Framework Managing Data Provenance*
28 Frans van der Sluis (UT) *When Complexity becomes Interesting: An Inquiry into the Information eXperience*
29 Iwan de Kok (UT) *Listening Heads*
30 Joyce Nakatumba (TUE) *Resource-Aware Business Process Management: Analysis and Support*
31 Dinh Khoa Nguyen (UvT) *Blueprint Model and Language for Engineering Cloud Applications*
32 Kamakshi Rajagopal (OUN) *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development*
33 Qi Gao (TUD) *User Modeling and Personalization in the Microblogging Sphere*
34 Kien Tjin-Kam-Jet (UT) *Distributed Deep Web Search*
35 Abdallah El Ali (UvA) *Minimal Mobile Human Computer Interaction*
36 Than Lam Hoang (TUE) *Pattern Mining in Data Streams*
37 Dirk Börner (OUN) *Ambient Learning Displays*
38 Eelco den Heijer (VU) *Autonomous Evolutionary Art*
39 Joop de Jong (TUD) *A Method for Enterprise Ontology based Design of Enterprise Information Systems*
40 Pim Nijssen (UM) *Monte-Carlo Tree Search for Multi-Player Games*
41 Jochem Liem (UvA) *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning*
42 Léon Planken (TUD) *Algorithms for Simple Temporal Reasoning*
43 Marc Bron (UvA) *Exploration and Contextualization through Interaction and Concepts*

2014

1 Nicola Barile (UU) *Studies in Learning Monotone Models from Data*
2 Fiona Tuliyano (RUN) *Combining System Dynamics with a Domain Modeling Method*
3 Sergio Raul Duarte Torres (UT) *Information Retrieval for Children: Search Behavior and Solutions*
4 Hanna Jochmann-Mannak (UT) *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation*
5 Jurriaan van Reijsen (UU) *Knowledge Perspectives on Advancing Dynamic Capability*
6 Damian Tamburri (VU) *Supporting Networked Software Development*
7 Arya Adriansyah (TUE) *Aligning Observed and Modeled Behavior*
8 Samur Araujo (TUD) *Data Integration over Distributed and Heterogeneous Data Endpoints*
9 Philip Jackson (UvT) *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language*
10 Ivan Salvador Razo Zapata (VU) *Service Value Networks*
11 Janneke van der Zwaan (TUD) *An Empathic Virtual Buddy for Social Support*
12 Willem van Willigen (VU) *Look Ma, No Hands: Aspects of Autonomous Vehicle Control*
13 Arlette van Wissen (VU) *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains*
14 Yangyang Shi (TUD) *Language Models With Meta-information*
15 Natalya Mogles (VU) *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare*
16 Krystyna Milian (VU) *Supporting trial recruitment and design by automatically interpreting eligibility criteria*
17 Kathrin Dentler (VU) *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability*
18 Mattijs Ghijsen (VU) *Methods and Models for the Design and Study of Dynamic Agent Organizations*
19 Vincius Ramos (TUE) *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support*
20 Mena Habib (UT) *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link*
21 Kassidy Clark (TUD) *Negotiation and Monitoring in Open Environments*
22 Marieke Peeters (UU) *Personalized Educational Games - Developing agent-supported scenario-based training*
23 Eleftherios Sidirourgos (UvA/CWI) *Space Efficient Indexes for the Big Data Era*

24 Davide Ceolin (VU) *Trusting Semi-structured Web Data*
25 Martijn Lappenschaar (RUN) *New network models for the analysis of disease interaction*
26 Tim Baarslag (TUD) *What to Bid and When to Stop*
27 Rui Jorge Almeida (EUR) *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty*
28 Anna Chmielowiec (VU) *Decentralized k-Clique Matching*
29 Jaap Kabbedijk (UU) *Variability in Multi-Tenant Enterprise Software*
30 Peter de Cock (UvT) *Anticipating Criminal Behaviour*
31 Leo van Moergestel (UU) *Agent Technology in Agile Multiparallel Manufacturing and Product Support*
32 Naser Ayat (UvA) *On Entity Resolution in Probabilistic Data*
33 Tesfa Tegegne (RUN) *Service Discovery in eHealth*
34 Christina Manteli (VU) *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems*
35 Joost van Ooijen (UU) *Cognitive Agents in Virtual Worlds: A Middleware Design Approach*
36 Joos Buijs (TUE) *Flexible Evolutionary Algorithms for Mining Structured Process Models*
37 Maral Dadvar (UT) *Experts and Machines United Against Cyberbullying*
38 Danny Plass-Oude Bos (UT) *Making brain-computer interfaces better: improving usability through post-processing*
39 Jasmina Maric (UvT) *Web Communities, Immigration, and Social Capital*
40 Walter Omona (RUN) *A Framework for Knowledge Management Using ICT in Higher Education*
41 Frederic Hogenboom (EUR) *Automated Detection of Financial Events in News Text*
42 Carsten Eijckhof (CWI/TUD) *Contextual Multidimensional Relevance Models*
43 Kevin Vlaanderen (UU) *Supporting Process Improvement using Method Increments*
44 Paulien Meesters (UvT) *Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden*
45 Birgit Schmitz (OUN) *Mobile Games for Learning: A Pattern-Based Approach*
46 Ke Tao (TUD) *Social Web Data Analytics: Relevance, Redundancy, Diversity*
47 Shangsong Liang (UVA) *Fusion and Diversification in Information Retrieval*

2015

1 Niels Netten (UvA) *Machine Learning for Relevance of Information in Crisis Response*
2 Faiza Bukhsh (UvT) *Smart auditing: Innovative Compliance Checking in Customs Controls*
3 Twan van Laarhoven (RUN) *Machine learning for network data*
4 Howard Spoelstra (OUN) *Collaborations in Open Learning Environments*
5 Christoph Bösch (UT) *Cryptographically Enforced Search Pattern Hiding*
6 Farideh Heidari (TUD) *Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes*
7 Maria-Hendrike Peetz (UvA) *Time-Aware Online Reputation Analysis*
8 Jie Jiang (TUD) *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions*
9 Randy Klaassen (UT) *HCI Perspectives on Behavior Change Support Systems*
10 Henry Hermans (OUN) *OpenU: design of an integrated system to support lifelong learning*
11 Yongming Luo (TUE) *Designing algorithms for big graph datasets: A study of computing bisimulation and joins*
12 Julie M. Birkholz (VU) *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks*
13 Giuseppe Procaccianti (VU) *Energy-Efficient Software*
14 Bart van Straalen (UT) *A cognitive approach to modeling bad news conversations*
15 Klaas Andries de Graaf (VU) *Ontology-based Software Architecture Documentation*
16 Changyun Wei (UT) *Cognitive Coordination for Cooperative Multi-Robot Teamwork*
17 André van Cleeff (UT) *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs*
18 Holger Pirk (CWI) *Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories*
19 Bernardo Tabuenca (OUN) *Ubiquitous Technology for Lifelong Learners*
20 Loïs Vanhée (UU) *Using Culture and Values to Support Flexible Coordination*
21 Sibren Fetter (OUN) *Using Peer-Support to Expand and Stabilize Online Learning*
22 Zhemin Zhu (UT) *Co-occurrence Rate Networks*
23 Luit Gazendam (VU) *Cataloguer Support in Cultural Heritage*
24 Richard Berendsen (UVA) *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation*
25 Steven Woudenberg (UU) *Bayesian Tools for Early Disease Detection*
26 Alexander Hogenboom (EUR) *Sentiment Analysis of Text Guided by Semantics and Structure*

27 Sándor Héman (CWI) *Updating compressed column-stores*

28 Janet Bagorogoza (TiU) *Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO*

29 Hendrik Baier (UM) *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains*

30 Kiavash Bahreini (OUN) *Real-time Multimodal Emotion Recognition in E-Learning*

31 Yakup Koç (TUD) *On Robustness of Power Grids*

32 Jerome Gard (UL) *Corporate Venture Management in SMEs*

33 Frederik Schadd (UM) *Ontology Mapping with Auxiliary Resources*

34 Victor de Graaff (UT) *Geosocial Recommender Systems*

35 Junchao Xu (TUD) *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction*

2016

1 Syed Saiden Abbas (RUN) *Recognition of Shapes by Humans and Machines*

2 Michiel Christiaan Meulendijk (UU) *Optimizing medication reviews through decision support: prescribing a better pill to swallow*

3 Maya Sappelli (RUN) *Knowledge Work in Context: User Centered Knowledge Worker Support*

4 Laurens Rietveld (VU) *Publishing and Consuming Linked Data*

5 Evgeny Sherkhonov (UVA) *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers*