Static Analysis of Monadic Datalog on Finite Labeled Trees

Dissertation
zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin

von
Herr Dipl.-Inf. André Frochaux

Präsidentin der Humboldt-Universität zu Berlin
Prof. Dr.-Ing. Dr. Sabine Kunst

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät
Prof. Dr. Elmar Kulke

Gutachter/innen:

1. Prof. Dr. Nicole Schweikardt, Humboldt-Universität zu Berlin
2. Prof. Dr. Isolde Adler, University of Leeds
3. Prof. Dr. Johannes Köbler, Humboldt-Universität zu Berlin

Tag der mündlichen Prüfung: 16.12.2016

Abstract

This thesis provides a comprehensive investigation into the decidability and complexity of the fundamental problems entailed by static analysis of monadic datalog on finite labeled trees.

Static analysis is used for optimizing queries without considering concrete database instances but exploiting information about the represented structure. Static analysis relies on three basic decision problems. First, the emptiness problem whose task is to decide whether a query returns the empty result on every database. Second, the equivalence problem asking if the result of two given queries always coincides on every database. And finally, the query containment problem where it is to decide whether on every database a given query produces a subset of the results of a second given query. We are interested in finding out whether these problems are decidable and, if so, what their complexity is.

We consider the aforementioned problems for monadic datalog on different representations of finite labeled trees. Mainly, tree structures shape the basis for models of semistructured data, probably best known in form of XML databases. We distinguish unordered and ordered trees which use the axis *child*, as well as the axes *firstchild* and *nextsibling*, respectively. An extension of the schemas by the *descendant*-axis is also considered. Furthermore, we distinguish ranked and unranked labeling alphabets.

Depending on the schema, the query language monadic datalog can reach the expressive power of monadic second order logic but remains efficiently evaluable. The query containment problem of monadic datalog on finite structures is known to be complete for 2Exptime. As we cannot express in monadic datalog that a finite structure is a tree, this result implies neither decidability nor a certain complexity for the query containment problem of monadic datalog on finite trees.

By using monadic second order logic, we show in a first step the decidability of the aforementioned problems on all considered representations of trees. Furthermore, we carve out the differences in the expressive power of monadic datalog on the different schemas. Usually, a wide range of expressive power suggests a wide range of complexity for the problems on the various schemas. But surprisingly, this is not confirmed by this thesis. The extensions by the predicates *root*, *leaf*, and *last sibling* gain the expressiveness of monadic datalog but not the complexity of the problems entailed by static analysis. In contrast, allowing the *descendant*-axis does not increase the expressive power but the complexity of our problems.

Then, the Exptime-hardness for all used schemas is shown by a reduction from the *two person corridor tiling problem* to the emptiness problem for monadic datalog on unordered unranked trees. This result is extended to all problems using one of the introduced schema.

After that, we demonstrate the membership to Exptime for the considered problems if the *descendant*-axis is omitted. To this aim, we translate the programs into two-way alternating tree automata whose principle of operation is very similar to monadic datalog evaluation. Those automata are transformed into nondeterministic bottom-up automata. This leads to an algorithm deciding the problems within exponential time. Consequently, these problems are complete for Exptime.

If the *descendant*-axis is involved, we present an algorithm that computes for a given query an equivalent query without *descendant* predicate in exponential time. Therefore,

the considered problems belong to 2EXPTIME in that case. A matching lower bound is proven by a reduction from the word problem of exponential space bounded alternating Turing machines.

Finally, we change the point of view from the set semantics to the bag semantics, reflecting the fact that in implemented database systems multiple occurrences of tuples are not eliminated unless elimination is explicitly requested. We propose a semantics for monadic datalog in the bag theoretic context. After that, we prove that the complexity of the emptiness problem of monadic datalog on finite trees under bag semantics is the same as under set semantics. Furthermore, we show by a reduction from a variant of Hilbert's tenth problem that the query containment problem of monadic datalog under bag semantics is undecidable in general.

Zusammenfassung

Die vorliegende Dissertation beinhaltet eine umfassende Untersuchung der Entscheidbarkeit und Komplexität der Probleme, welche sich durch eine statische Analyse für monadisches Datalog auf endlichen gefärbten Bäumen stellen.

Statische Analyse bedeutet hierbei Anfrageoptimierung ohne Blick auf konkrete Instanzen, aber mit Rücksicht auf deren zugrundeliegende Struktur. Im Kern beinhaltet dies die Lösung der drei folgenden Probleme: das Leerheitsproblem (die Frage, ob eine Anfrage auf jeder Instanz ein leeres Ergebnis liefert), das Äquivalenzproblem (die Frage, ob zwei Anfragen auf jeder Instanz das gleiche Ergebnis liefern) und das Query-Containment-Problem (die Frage, ob das Ergebnis der einen Anfrage auf jeder Datenbank im Ergebnis der anderen Anfrage enthalten ist). Von Interesse ist dabei, ob die Fragen für eine gegebene Anfragesprache entscheidbar sind und wenn ja, welche Komplexität ihnen inne wohnt.

Wir betrachten diese Probleme für monadisches Datalog auf unterschiedlichen Repräsentationen für endliche gefärbte Bäume. Baumstrukturen sind oft Grundlage für Modelle von semistrukturierten Daten (in ihrer bekanntesten Form als XML-Datenbanken). Hierbei unterscheiden wir zwischen ungeordneten und geordneten Bäumen, welche die Achsen *child* bzw. *firstchild* und *nextsibling* nutzen. Die verwendeten Schema können auch zusätzlich durch die *descendant*-Achse erweitert werden. Außerdem wird zwischen Färbungen mittels Alphabeten mit und ohne Rang unterschieden.

Monadisches Datalog ist eine Anfragesprache, die in Abhängigkeit vom gewählten Schema die Ausdrucksstärke der monadischen Logik zweiter Stufe erreicht und dennoch mit einem effizienten Algorithmus ausgewertet werden kann. Für monadisches Datalog auf endlichen Strukturen ist bereits bekannt, dass das Query-Containment-Problem vollständig für 2EXPTIME ist. Dies impliziert allerdings weder die Entscheidbarkeit noch eine bestimmte Komplexität für das Problem auf Baumstrukturen, da in monadischem Datalog nicht ausdrückbar ist, dass eine endliche Struktur einen Baum repräsentiert.

Mit Hilfe der monadischen Logik zweiter Stufe wird zunächst die Entscheidbarkeit der drei oben genannten Probleme auf allen betrachteten Baumrepräsentationen gezeigt. Dabei wird auch die in Abhängikeit von der gewählten Baumrepräsentation (und damit dem Schema) stark unterschiedliche Ausdrucksstärke herausgearbeitet. Diese suggeriert eine starke Diversität bei der Komplexität der Probleme statischer Analyse, welche sich jedoch in späteren Kapiteln nicht bestätigt. In Gegensatz hierzu, erhöht das Hinzunehmen

der *descendant*-Achse die Ausdrucksstärke nicht, aber die Komplexität der betrachteten Probleme.

Nachfolgend wird die EXPTIME-Härte für alle verwendeten Schema gezeigt. Grundlage der Beweise ist eine Reduktion vom *Two Person Corridor Tiling Problem* auf das Leerheitsproblem von monadischem Datalog auf ungeordneten Bäumen mit Beschriftungen aus einem Alphabet ohne Rang. Dieses Ergebnis wird auf alle verwendeten Schema und Repräsentationen übertragen.

Für den Nachweis der Zugehörigkeit werden anschließend mDatalog-Programme ohne *descendant*-Achse in Zwei-Wege-Alternierende-Baumautomaten übersetzt. Deren Arbeitsweise ist sehr mit der Auswertung von Programmen in monadischem Datalog mittels der oben genannten Achsen auf Bäumen verwandt. Diese Automaten werden in nichtdeterministische Baumautomaten übersetzt, welche bereits in diversen Arbeiten sehr gut untersucht wurden. Insgesamt werden die Probleme mit diesem Vorgehen der Klasse EXPTIME zugeordnet, weshalb sie insgesamt vollständig für EXPTIME sind.

Für den Fall, dass die *descendant*-Achse involviert ist, wird ein Algorithmus vorgestellt, welcher äquivalente Anfragen ohne Verwendung der *descendant*-Achse in exponentieller Zeit berechnet, wodurch die Probleme insgesamt in zweifach exponentieller Zeit gelöst werden können. Eine passende untere Schranke wird für fast alle Schema durch eine Reduktion vom Wortproblem für alternierende, exponentiell platzbeschränkte Turingmaschinen gezeigt.

Im abschließenden Kapitel wechseln wir die Sichtweise von der Mengen-Semantik hin zur Multimengen-Semantik. Dies trägt der Tatsache Rechnung, dass in implementierten Datenbanksystemen Tupel in Relationen – wie in Ergebnissen – mehrfach vorkommen können. Wir schlagen eine Semantik für Datalog unter Multimengen-Semantik vor und zeigen, dass die Komplexität des Leerheitsproblems für monadisches Datalog auf endlichen Bäumen unter Multimengen-Semantik der Komplexität des Problems unter Mengen-Semantik entspricht. Außerdem zeigen wir mittels einer Reduktion von einer Variante von Hilberts zehntem Problem, dass das Query-Containment-Problem für monadisches Datalog auf endlichen, gefärbten Bäumen unter Multimengen-Semantik im Allgemeinen unentscheidbar ist.

Acknowledgments

This thesis would not have been possible without the support of many people throughout the last years.

First and foremost, I want to thank my advisor Prof. Dr. Nicole Schweikardt for offering a place in her group, for giving me freedom in my work and research. I greatly benefited from her deep scientific insights and her talent to put complex ideas into simple terms for solving seemingly intractable problems. I am grateful for the excellent guidance and the continuous support, for the patience and the encouragement throughout the last years of research and thesis writing.

I thank Prof. Dr. Isolde Adler and Prof. Dr. Johannes Köbler very much for serving as a reviewer of this thesis.

A special thanks goes to Prof. Dr. Leser for bringing the bag semantics to my attention.

Of course, I would like to thank all my former and current colleagues forming an agreeable atmosphere. Particularly, this is due to Jutta Nadland and Eva Sandig assisting (and sometimes saving) us so often in many different ways as well as Petra Kämpfer managing the technical support.

I would like to double out André Hernich and Mariano Zelke. They introduced me into a researcher's life and working at a university. Thanks for ongoing interest into my ideas and for countless inspiring discussions on and off researcher's topics.

I grateful thank Mariano for carefully proofreading this thesis. Moreover, I thank Maria Tammik, Christoph Berkholz, Joachim Bremer, and Lucas Heimberg for reading parts of this thesis and their helpful comments.

Thanks to Lucas for preparing several awesome suppers in Frankfurt and to Noemi helping us out to eat all that.

Ein besonderer Dank geht an meine Freunde und meine Familie hier in Berlin. Unterstützung während der Frankfurter Zeit hieß vor allem Verzicht! Ganz besonders traf das natürlich Etienne, dem immer wieder noch viel spannendere Projekte als das Arbeiten an einer Dissertation einfallen und Violette, die alle Tiefen und Höhen, die so eine Zeit mit sich bringt, ganz tapfer ertragen und mich immer wieder bestärkt hat.

pour toi,
mon soleil, mes étoiles,
mon roc et ma tempête,
mon tout, ma compagne.

ma violette.

Contents

Chapter 1

Introduction

In basically every query language there are countless equivalent database queries to a given query. Those queries produce on an arbitrary database instance the same result but executing them may lead to different query plans and highly diverse evaluation procedures after the compilation. Therefore, the evaluation time of equivalent queries can differ widely. Finding to a given query an equivalent query with minimal evaluation time is the main task of query optimizing. An optimizer can address to different points. In this thesis we focus on optimization without taking care of concrete database instances.

In particular, this approach is of interest in a context where the same query is evaluated many times on databases with rapidly changing data. Following this approach, it is fundamental to consider the three following questions (cf. [SSS09]).

(1.) Does query Q produce the empty result on every database D?

A query Q producing on every database the empty result is called unsatisfiable.

(2.) Does query Q_1 produce the same result as query Q_2 on every database D?

Two queries Q_1 and Q_2 producing on every database the same result are called equivalent.

(3.) Is the result of Q_1 on every database D a subset of the result that Q_2 produces on D?

A query Q_1 producing on every database a result that is a subset of the result of Q_2 is called contained in Q_2.

The decision problems corresponding to these questions are called the emptiness problem, the equivalence problem, and the query containment problem, respectively.

Of course, there is a strong connection between these three questions. So, Q_1 and Q_2 are equivalent if and only if Q_1 is contained in Q_2 and Q_2 is contained in Q_1. Furthermore, if a query Q_1 is equivalent to a query that always produces the empty result, then Q_1 is unsatisfiable.

Clearly, the equivalence problem reduces to the query containment problem. Moreover, if a language is closed under complementation and conjunction, then the containment problem reduces to the emptiness problem: Q_1 is contained in Q_2 if and only if the

result of a query saying 'Q_1 and not Q_2' is always empty. Finally, the containment problem reduces to the equivalence problem if the query language is closed under disjunction: Q_1 is contained in Q_2 if and only if the query 'Q_1 or Q_2' is equivalent to Q_2.

The whole area of reasoning about the semantic properties of queries is called static analysis.

We consider the above-mentioned problems for monadic datalog on finite labeled trees. Practically all models of semistructured data base on labeled trees. The probably best known form of semistructured data is XML. Example 1.1 illustrates how an XML file can be modeled through a labeled tree. Usually, the children of the same node in such a tree model are sorted by a sibling ordering. If such an order is present, we speak of ordered trees. If otherwise the ordering is irrelevant or omitted, then the trees are called unordered.

Example 1.1. *Figure 1.1 shows how to obtain a tree model from a document with semistructured data, in this case an XML document of an ordering. The document uses the following labeling alphabet*

$$\Sigma = \{\text{order}, \text{head}, \text{date}, \text{name}, \text{address}, \text{item}, \text{orderno}, \text{price}, \ldots\}.$$

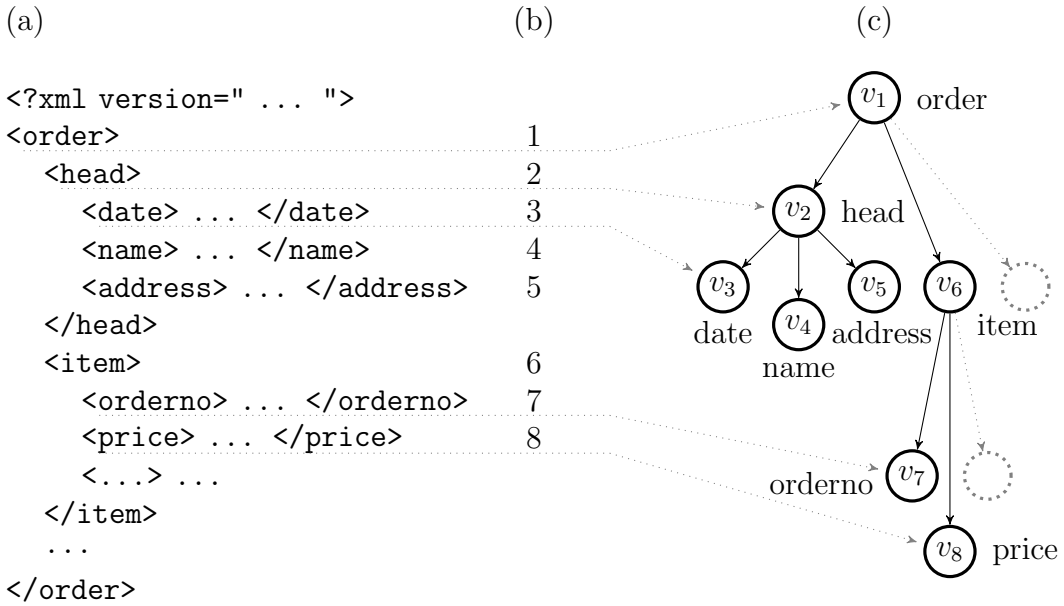

Figure 1.1: (a) An XML document, (b) consecutive numbering of the opening tags and (c) the resulting tree model where the labels are placed beside the nodes.

The type definitions for such documents are given in (extended) context-free grammars, for XML in the form of DTDs. Derivation trees of context-free grammars are ranked, that means that the number of children of a node is determined by the label of that node and therefore the number of children of a node is bounded by some constant. If extended context-free grammars are used, which allow arbitrary regular expressions

over grammar symbols on the right-hand side of productions, the derivation trees will be unranked. Thus, by contrast to ranked trees, unranked trees have no restrictions on the number of children a node can have.

As such semistructured documents are one of the standards for information exchange in the world wide web, it is of interest to query, transform, or validate them. Certainly, (query) automata, logics, and query languages for trees have been studied extensively. As an overview, also in the context of XML, we refer to surveys like [Abi01], [Lib06], [Nev02], [NS02], [Sch07], and [Via01]. It turns out that query languages and schema formalisms for XML tend to use monadic second order logic (MSO) as the benchmark. It can be seen that type definitions, for example, are essentially equivalent to MSO sentences and various query languages have the expressive power of unary MSO queries.

MSO over trees is well-understood (cf. [TW68], [Don70], [Cou90]). The combined complexity, that is with respect to both the input and the query size, for the evaluation problem of MSO over trees is known to be complete for PSPACE. Adherent to the assumption that the same query is posed many times against a changing database, it is reasonable to consider evaluation complexity in terms of the size of the database only. This complexity is called data complexity. Considering data complexity, unary MSO queries on trees can be evaluated in linear time (cf. [FFG01] and [FG02]) by using an approach having nonelementary complexity in terms of the size of the query.

As a consequence of Trahtenbrot's Theorem (cf. [EF95]), the emptiness problem, the equivalence problem, and the query containment problem for MSO on finite structures are not decidable in general. However, on trees, these problems are decidable [TW68] by using translations into tree automata with nonelementary size [FG02].

We consider the query language monadic datalog. Datalog itself can be seen as a standard tool for expressing queries with recursion. The data complexity of the evaluation problem is P-complete (implicit in [Var82], [Imm86]) whereas the combined complexity is complete for Exptime (implicit in [Var82]). The emptiness problem of datalog is decidable (cf. [AHV95]), but the equivalence problem and the query containment problem are undecidable [Shm87].

Depending on the schema and the structures under consideration, the monadic version of datalog can reach the expressive power of MSO. In [GK04], Gottlob and Koch proved that the combined complexity of the evaluation problem over arbitrary finite structures is complete for NP. Already in 1988, Cosmadakis et al. [CGKV88] showed that the query containment problem for monadic datalog on finite structures is decidable within 2-fold exponential time and hard for Exptime. This complexity gap was closed by Benedikt et al. [BBS12] in 2012 by proving 2Exptime-completeness.

Gottlob and Koch [GK04] established the first results for monadic datalog concerning trees. They showed that the combined complexity of the evaluation problem for monadic datalog for ordered unranked labeled trees is solvable in polynomial time, yet in linear time if the query is in Tree-Marking Normal Form, a normal form introduced in the same article. Furthermore, they showed that the query containment problem for ordered unranked labeled trees is hard for Exptime and decidable. For achieving these results, a schema is utilized providing the firstchild axis, the nextchild axis, and unary relations for representing the root, the leaves, the last siblings, and the labels of the node. The hardness result is obtained by a reduction from query automata introduced by Neven and Schwentick in [NS02]. A tight upper bound was left open.

In this thesis, we close this complexity gap by presenting an algorithm that decides the query containment problem for monadic datalog on ordered unranked trees within exponential time. A translation of Boolean queries within polynomial time into two-way alternating tree automata plays a key role in this approach. Two-way alternating automata were introduced in [Slu85].

Moreover, by a reduction from the two person corridor tiling problem, we prove that the problem is hard for EXPTIME even if only the binary predicates firstchild and nextsibling together with the labeling predicates are used. These EXPTIME-completeness results are extended to the emptiness problem and the equivalence problem. At once, we establish EXPTIME-completeness for the mentioned problems for monadic datalog on unordered unranked labeled trees where the schema uses the relation child and the unary labeling predicates. These completeness results hold even for extensions of the schema by the root and the leaf predicate.

Finally, we extend the former schema by the relation descendant. In contrast to the former extensions by the unary predicates, the descendant predicate does not increase the expressive power of the queries. We usually assume that increasing the expressive power goes hand in hand with a higher complexity of the considered problems. However, if we extend the schema by root, leaf, or last sibling relations this is not the case, as noted above. But even more surprisingly, the allowance of the descendant predicate enhances the complexity of the considered problems without increasing the expressiveness.

During the last part of this thesis, for practically all schemas, a 2EXPTIME lower bound is proven. By a reduction from the validity problem for Boolean conjunctive queries, we achieve this result for the query containment problem of monadic datalog on ordered unranked labeled trees. After that, we obtain 2EXPTIME-hardness for the emptiness problem for Boolean monadic datalog on unordered ranked labeled trees by a reduction from the word problem for exponential space bounded alternating Turing machines. This result is adopted to the remaining problems and tree representations which allow the descendant axis. A matching upper bound is given by presenting an algorithm solving the problems for all considered schemas by extending rewriting methods of conjunctive queries introduced in [GKS06]. For unordered unranked labeled trees, this answers a question posed by Abiteboul et al. in [ABMW13] asking whether the query containment problem of monadic datalog is decidable for unordered unranked labeled trees if the descendant axis is or is not involved.

Besides, we provide evidence for all afore considered problems for all mentioned representations of trees labeled by a ranked alphabet.

With real world databases in mind, we finally investigate the emptiness problem and the query containment problem for monadic datalog on trees under bag semantics. To this end, we propose a semantics for monadic datalog under bag semantics that extends the known proof tree semantics of the set theoretic context.

Structure of this Thesis

Chapter 2 fixes the basic notations used throughout this thesis. It defines the various representations of finite labeled trees including the underlying relational structures. Furthermore, Chapter 2 provides definitions of syntax and semantics of monadic datalog and point out monotonicity and preservation under homomorphisms, two important proper-

ties of monadic datalog. Finally, we formalize the emptiness problem, the equivalence problem, as well as the query containment problem.

Chapter 3 introduces monadic second order logic (MSO). We analyze the expressive power of monadic datalog by utilizing its monotonicity and preservation properties and compare them with the expressive power of MSO. Translating monadic datalog queries into unary queries in MSO leads to decidability results for the three aforementioned problems on all introduced tree structures and their different schemas. The content of Chapter 3 for unranked finite trees was published in [FS13].

In Chapter 4, we focus on the lower bounds. Therefore, we give a reduction from the two person corridor tiling problem to the emptiness problem of monadic datalog on unordered unranked labeled trees using a schema extended by unary predicates denoting the root and the leaves of the tree. In a second step, we show that this extension can be skipped. Later, the result is transferred to the emptiness problem for monadic datalog on ranked unordered trees. This is done by modifying the former reduction to use a ranked alphabet making the reduction more intuitive in some steps.

In Chapter 5, we concentrate on the upper bound for schemas not using the descendant axis. To this end, we solve the query containment problem for unary monadic datalog queries on unranked ordered labeled trees using a maximal extended schema for ordered trees. As on this schema it is expressible that a finite labeled tree respects a certain rank, this result implies the membership even for ranked trees. The former result is obtained by using the automata-theoretic method. In particular, we translate the two given unary monadic datalog queries into appropriate Boolean queries which in turn will be translated into Boolean queries in monadic datalog on binary trees. This enables a further translation into nondeterministic bottom-up tree automata. One query can be translated directly. For the second query, a (repeated) negation would prevent the 2-fold exponential time bound and instead a detour via two-way alternating tree automata is demonstrated. After the two corresponding nondeterministic bottom-up tree automata are obtained, the problem is reduced to standard methods of automata theory. Finally, the algorithm solving the query containment problem for monadic datalog on finite labeled trees can be used to solve the emptiness problem and the equivalence problem within the same effort.

The content of Chapter 4 and Chapter 5, concerning the query containment problem for monadic datalog on unranked labeled trees, was published in [FGS14].

In Chapter 6, we consider schemas allowing the use of the descendant axis. At first, we present an algorithm to rewrite, within exponential time, such queries into monadic datalog queries which do not make use of the descendant predicate. Together, with the results of Chapter 5, this leads to an algorithm solving the problems within 2-fold exponential time. For the lower bound, we first offer an intuitive and short reduction from the validity problem for Boolean conjunctive queries with respect to a tree automaton to the query containment problem for monadic datalog on ordered unranked trees using predicates to denote first child, next sibling, descendant, root, leaf, last sibling, as well as the labeling predicates. As the reduction makes heavily use of the order, an adaption to unranked trees seems to be pointless. Thus, an alternative, more technically, reduction from the word problem of exponential space bounded alternating Turing machines to the emptiness problem for Boolean monadic datalog on unordered ranked labeled trees is presented. This reduction can be extended to a proof for unranked labeled trees.

Finally, the obtained result is carried over to all considered problems on basically all used representations.

The membership result of the query containment problem for monadic datalog on unranked labeled trees using the descendant axis was published in [FGS14]. The corresponding hardness result was published in [FS16].

Chapter 7 gives a brief introduction to bag semantics. We propose a semantics for datalog under bag semantics and investigate the emptiness problem and the query containment problem of monadic datalog on finite labeled trees. We prove that the complexity of the emptiness problem of monadic datalog under set semantics carries over to the emptiness problem under bag semantics. Considering the query containment problem for monadic datalog on finite labeled trees under bag semantics, we show that the problem in general is undecidable. Starting point for the proof is the m-variable Diophantine equation problem over the natural numbers that is a variant of Hilbert's tenth problem.

Finally, Chapter 8 concludes this thesis by briefly summarizing the results.

Chapter 2

Preliminaries

In this chapter we fix the basic notations used throughout this thesis. We define the various representations of finite labeled trees including the underlying relational structures. Furthermore, we give definitions of syntax and semantics of monadic datalog and point out monotonicity and preservation under homomorphisms, two important properties of monadic datalog. Finally, we formalize the emptiness problem, the equivalence problem, as well as the query containment problem.

Basic Notations

We write $\mathbb{N}$ for the set of non-negative integers, and we let $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$. For a set S we write 2^S to denote the power set of S that is the set $\{X : X \subseteq S\}$.

Let M be a set. The set $\mathcal{B}^+(M)$ of *positive Boolean formulas over* M contains all elements in M, and is closed under $\wedge$ and $\vee$.[1] For a set $M' \subseteq M$ and a formula $\theta \in \mathcal{B}^+(M)$, we say that M' *satisfies* θ if and only if by assigning **true** to elements in M' and **false** to elements in $M \setminus M'$ the formula θ of propositional logic (without variables) evaluates to true.

2.1 Alphabets

We distinguish between two kinds of alphabets. An *unranked* alphabet Σ is always a finite, non-empty set, whereas a *ranked* alphabet Σ is a pair (σ, ar) of a finite, non-empty set of symbols σ and a function $ar : \sigma \to \mathbb{N}$. Since the symbol set σ of Σ is finite, there exists a smallest $m \in \mathbb{N}$ such that there is the unique partition $\sigma = \Sigma_0 \,\dot\cup\, \Sigma_1 \,\dot\cup\, \ldots \,\dot\cup\, \Sigma_m$ where

$$\Sigma_m \neq \varnothing \qquad \text{and} \qquad \Sigma_i = \{\alpha \in \sigma \mid ar(\alpha) = i\}, \qquad 0 \leqslant i \leqslant m.$$

We define $rk_{\max}(\Sigma) := m$. To denote an element $\alpha \in \sigma$ for a ranked alphabet $\Sigma = (\sigma, ar)$, we often write $\alpha \in \Sigma$.

[1] An inductive definition of the set $\mathcal{B}^+(M)$: Let M be a set.
(Base) $M \subseteq \mathcal{B}^+(M)$
(Inductive Step) Let ψ and φ be elements of $\mathcal{B}^+(M)$, then we have
(I) $(\psi \wedge \varphi) \in \mathcal{B}^+(M)$
(II) $(\psi \vee \varphi) \in \mathcal{B}^+(M)$

The size $|\Sigma|$ of an unranked alphabet Σ is the number of its elements. For a ranked alphabet $\Sigma = (\sigma, ar)$ we define:

$$|\Sigma| := \sum_{\alpha \in \sigma} (1 + ar(\alpha)).$$

2.2 Complexity Classes

Next, we recall some definitions concerning the complexity classes. Let $T : \mathbb{N} \to \mathbb{N}$ be a function. The sets $\textsc{Dtime}(T(n))$ and $\textsc{Atime}(T(n))$ denote the classes of languages accepted, that means decision problems decided, in time $T(n)$ by deterministic and alternating Turing machines, respectively. The classes $\textsc{Dspace}(S(n))$ and $\textsc{Aspace}(S(n))$, for a function $S : \mathbb{N} \to \mathbb{N}$, are defined analogously, in the context of the resource space. In this thesis, we consider the following complexity classes of decision problems.

$$\textsc{Exptime} := \bigcup_{k \in \mathbb{N}} \textsc{Dtime}(2^{(n^k)})$$

$$\textsc{AExpspace} := \bigcup_{k \in \mathbb{N}} \textsc{Aspace}(2^{(n^k)})$$

$$\textsc{2Exptime} := \bigcup_{k \in \mathbb{N}} \textsc{Dtime}(2^{(2^{(n^k)})})$$

As a consequence of the Time Hierarchy Theorem [HS65] it holds that $\textsc{Exptime} \subsetneq \textsc{2Exptime}$. By [CKS81], we know that $\textsc{AExpspace} = \textsc{2Exptime}$. For further background on complexity classes and (alternating) Turing machines, we refer to textbooks like [Pap94] and [AB09].

2.3 Relational Structures

As usual, a *schema* (or *relational signature*) τ consists of a finite number of relation symbols R, each of a fixed *arity* $ar(R) \in \mathbb{N}_{\geqslant 1}$. A *$\tau$-structure* $\mathcal{A}$ consists of a finite, non-empty set A called the *domain* (or *universe*) of $\mathcal{A}$, and a relation $R^{\mathcal{A}} \subseteq A^{ar(R)}$ for each relation symbol $R \in \tau$. Sometimes it will be convenient to identify $\mathcal{A}$ with the *set of atomic facts of* $\mathcal{A}$ that is the set

$$atoms(\mathcal{A}) \;:=\; \{\; R(a_1, \ldots, a_r) \;:\; R \in \tau,\;\; r = ar(R),\;\; (a_1, \ldots, a_r) \in R^{\mathcal{A}} \;\}.$$

If τ and τ' are schemas such that $\tau \subseteq \tau'$ and $\mathcal{A}$ is a τ-structure and $\mathcal{B}$ a τ'-structure, then $\mathcal{A}$ is the *τ-reduct* of $\mathcal{B}$ and $\mathcal{B}$ is a *τ'-expansion* of $\mathcal{A}$ if $\mathcal{A}$ and $\mathcal{B}$ have the same domain and $R^{\mathcal{A}} = R^{\mathcal{B}}$ is true for all $R \in \tau$.

If τ is a schema and ℓ is a list of relation symbols, we write τ^{ℓ} to denote the extension of the schema τ by the relation symbols in ℓ. Furthermore, τ_{Σ} denotes the extension of τ by new unary relation symbols $\mathbf{label}_{\alpha}$ for all $\alpha \in \Sigma$.

2.4 Tree Structures

Throughout the whole thesis, we consider finite, node labeled trees and distinguish between *ordered* and *unordered* trees.

2.4.1 Unordered Trees

Let Σ be a ranked or unranked alphabet.

An *unordered Σ-labeled tree* $T = (V^T, \lambda^T, E^T)$ consists of a finite set V^T of nodes, a function $\lambda^T : V^T \to \Sigma$ assigning to each node v of T a label $\lambda(v) \in \Sigma$, and a set $E^T \subseteq V^T \times V^T$ of directed edges such that the following holds true:

- There is exactly one node $root^T \in V^T$ with in-degree 0. This node is called the *root* of T.
- Every node $v \in V^T$ with $v \neq root^T$ has in-degree 1, and there is exactly one directed path from $root^T$ to v.
- If Σ is a ranked alphabet, then additionally every node v labeled by $\lambda^T(v)$ has exactly $ar(\lambda^T(v))$ children.

If we want to point out that an unordered Σ-labeled tree is labeled by symbols from a ranked alphabet, then we call it a *ranked unordered Σ-labeled tree*. Otherwise, if it is labeled by symbols from an unranked alphabet, we call it an *unranked unordered Σ-labeled tree*.

Note that the alphabet is always finite. For a fixed ranked alphabet Σ this implies a bounded maximal number of children for all nodes of any ranked Σ-labeled tree. Unranked trees are not affected by this restriction.

As done in [ABMW13], we represent an unordered Σ-labeled tree T by a $\tau_{u,\Sigma}$-structure $\mathcal{S}_u(T)$ with the unary predicates $\mathbf{label}_\alpha$ for all $\alpha \in \Sigma$ and the binary predicate **child** where

- the domain of $\mathcal{S}_u(T)$ is the set V^T of all nodes of T,
- for each label $\alpha \in \Sigma$, $\mathbf{label}_\alpha^{\mathcal{S}_u(T)}$ consists of the set of all nodes labeled α that is

$$\mathbf{label}_\alpha^{\mathcal{S}_u(T)} = \{v \in V^T \,:\, \lambda^T(v) = \alpha\}, \text{ and}$$

- $\mathbf{child}^{\mathcal{S}_u(T)} = E^T$.

We fix the schema

$$\tau_u := \{\mathbf{child}\}$$

for the $\tau_{u,\Sigma}$-structures, and when no confusion arises we simply write T instead of $\mathcal{S}_u(T)$.

Example 2.1. *Let T be the unranked Σ-labeled unordered tree from Figure 2.1,*[2] *for $\Sigma = \{Black, White\}$. The $\tau_{u,\Sigma}$-structure $\mathcal{A} = \mathcal{S}_u(T)$ representing T consists of the domain*

$$A \;=\; \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

and the relations

[2]Note that an unordered tree does not contain any information on the relative order of the children of a node. Thus, the arrangement of children given in the picture is only one of many possibilities to draw the tree.

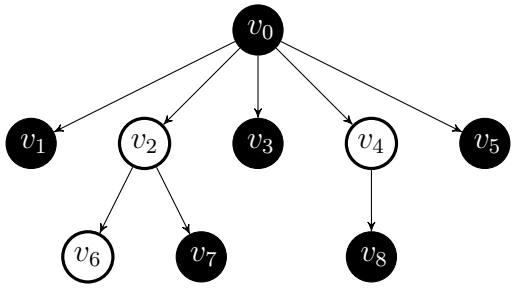

Figure 2.1: An example tree T labeled by symbols from the unranked alphabet $\Sigma = \{Black, White\}$.

- $\mathbf{label}^{\mathcal{A}}_{Black} = \{v_0, v_1, v_3, v_5, v_7, v_8\}$,
- $\mathbf{label}^{\mathcal{A}}_{White} = \{v_2, v_4, v_6\}$,
- $\mathbf{child}^{\mathcal{A}} = \left\{ \begin{array}{l} (v_0, v_1), (v_0, v_2), (v_0, v_3), (v_0, v_4), (v_0, v_5), \\ (v_2, v_6), (v_2, v_7), (v_4, v_8) \end{array} \right\}$.

The set of atomic facts of $\mathcal{A}$ is the set atoms($\mathcal{A}$) =

$$\left\{ \begin{array}{l} \mathbf{label}_{Black}(v_0),\ \mathbf{label}_{Black}(v_1),\ \mathbf{label}_{Black}(v_3),\ \mathbf{label}_{Black}(v_5), \\ \mathbf{label}_{Black}(v_7),\ \mathbf{label}_{Black}(v_8),\ \mathbf{label}_{White}(v_2),\ \mathbf{label}_{White}(v_4), \\ \mathbf{label}_{White}(v_6),\ \mathbf{child}(v_0, v_1),\ \mathbf{child}(v_0, v_2),\ \mathbf{child}(v_0, v_3), \\ \mathbf{child}(v_0, v_4),\ \mathbf{child}(v_0, v_5),\ \mathbf{child}(v_2, v_6),\ \mathbf{child}(v_2, v_7),\ \mathbf{child}(v_4, v_8) \end{array} \right\}.$$

⌟

We will also consider extensions of the schema τ_u by the binary predicate **desc** (as a shortcut for descendant) and the unary predicates **root** and **leaf**.

For the set $M = \{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$, the $\tau^M_{u,\Sigma}$-representation $\mathcal{S}^M_u(T)$ of an unordered Σ-labeled tree T is the expansion of $\mathcal{S}_u(T)$ by the relations

- $\mathbf{desc}^{\mathcal{S}^M_u(T)}$ which is the transitive (and non-reflexive) closure of E^T,
- $\mathbf{root}^{\mathcal{S}^M_u(T)}$ consists of the root node $root^T$ of T,
- $\mathbf{leaf}^{\mathcal{S}^M_u(T)}$ consists of all leaves of T, i.e., all $v \in V^T$ that have out-degree 0.

For a set $N \subseteq \{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$ we let

$$\tau^N_u \ := \ \tau_u \cup N,$$

and for every Σ-labeled unordered tree T we let $\mathcal{S}^N_u(T)$ be the τ^N_u-reduct of $\mathcal{S}^M_u(T)$. If N is a singleton set, we omit the curly brackets — in particular, we write $\tau^{\mathbf{desc}}_u$ instead of $\tau^{\{\mathbf{desc}\}}_u$ and $\mathcal{S}^{\mathbf{desc}}_u(T)$ instead of $\mathcal{S}^{\{\mathbf{desc}\}}_u(T)$.

2.4.2 Ordered Trees

An *ordered Σ-labeled tree* $T = (V^T, \lambda^T, E^T, \mathit{order}^T)$ consists of the same components as an unordered Σ-labeled tree and, in addition, order^T fixes, for each node u of T, a strict linear order of all the children of u in T.

If we want to point out that an ordered Σ-labeled tree is labeled by symbols from a ranked alphabet, then we call it *ranked ordered Σ-labeled tree*. Otherwise, if it is labeled by symbols from an unranked alphabet, we call it *unranked ordered Σ-labeled tree*.

We represent an ordered Σ-labeled tree T by a relational $\tau_{o,\Sigma}$-structure $\mathcal{S}_o(T)$ with the unary predicates $\mathbf{label}_\alpha$ (for every $\alpha \in \Sigma$) and the binary predicates **fc** (as a shortcut for *first child*) and **ns** (as a shortcut for *next sibling*) where **fc** and **ns** have arity two and $\mathbf{label}_\alpha$ has arity one as follows:

- The domain of $\mathcal{S}_o(T)$ is the set V^T of all nodes of T,
- for each $\alpha \in \Sigma$, the relation $\mathbf{label}_\alpha^{\mathcal{S}_o(T)}$ is defined in the same way as for unordered trees,
- $\mathbf{fc}^{\mathcal{S}_o(T)}$ consists of all tuples (u, v) of nodes such that v is the first child of u in T (that is order^T lists v as the first child of u),
- $\mathbf{ns}^{\mathcal{S}_o(T)}$ consists of all tuples (v, v') of nodes such that v and v' have the same parent, i.e., there is an $u \in V^T$ such that $(u, v) \in E^T$ and $(u, v') \in E^T$, and v' is the immediate successor of v in the linear order of the children of u given by order^T,

and we fix the schema

$$\tau_o \;:=\; \{\, \mathbf{fc}, \mathbf{ns} \,\}.$$

Often we will also consider the extensions of the schema by the binary predicates **child** and **desc** as well as the unary predicates **root**, **leaf**, and **ls** (as a shortcut for *last sibling*).

For the set $M = \{\, \mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf}, \mathbf{ls} \,\}$, the τ_o^M-representation $\mathcal{S}_o^M(T)$ of an ordered Σ-labeled tree T is the expansion of $\mathcal{S}_o(T)$ by the relations

- $\mathbf{child}^{\mathcal{S}_o^M(T)}$, $\mathbf{desc}^{\mathcal{S}_o^M(T)}$, $\mathbf{root}^{\mathcal{S}_o^M(T)}$, and $\mathbf{leaf}^{\mathcal{S}_o^M(T)}$ which are defined in the same way as for unordered trees and
- $\mathbf{ls}^{\mathcal{S}_o^M(T)}$ which consists of all nodes such that order^T lists v as the last child of its parent u.

For a set $N \subseteq \{\mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf},\ \mathbf{ls}\}$ we let

$$\tau_o^N \;:=\; \tau_o \cup N,$$

and for every ordered Σ-labeled tree T we let $\mathcal{S}_o^N(T)$ be the τ_o^N-reduct of $\mathcal{S}_o^M(T)$. If N is a singleton set, we omit the curly brackets.

In [GK04] Gottlob and Koch represent ordered Σ-labeled trees T by $\tau_{GK,\Sigma}$-structures $\mathcal{S}_{GK}(T) := \mathcal{S}_o^{\{\mathbf{root},\mathbf{leaf},\mathbf{ls}\}}(T)$ for the schema

$$\tau_{GK} \;:=\; \tau_o^{\{\mathbf{root},\mathbf{leaf},\mathbf{ls}\}} \;=\; \{\, \mathbf{fc},\ \mathbf{ns},\ \mathbf{root},\ \mathbf{leaf},\ \mathbf{ls} \,\}.$$

Example 2.2. *Let T be the unranked ordered Σ-labeled tree from Figure 2.1 for $\Sigma = \{Black, White\}$ where the order of the children of each node is from left to right as depicted in the illustration. The $\tau_{GK,\Sigma}$-structure $\mathcal{B} = \mathcal{S}_{GK}(T)$ representing T has the domain*

$$B \;=\; \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

and the relations

- $\mathbf{label}^{\mathcal{B}}_{Black} = \{v_0, v_1, v_3, v_5, v_7, v_8\}$,
- $\mathbf{label}^{\mathcal{B}}_{White} = \{v_2, v_4, v_6\}$,
- $\mathbf{root}^{\mathcal{B}} = \{v_0\}$,
- $\mathbf{leaf}^{\mathcal{B}} = \{v_1, v_3, v_5, v_6, v_7, v_8\}$,
- $\mathbf{fc}^{\mathcal{B}} = \{\, (v_0, v_1),\ (v_2, v_6),\ (v_4, v_8) \,\}$,
- $\mathbf{ns}^{\mathcal{B}} = \{\, (v_1, v_2),\ (v_2, v_3),\ (v_3, v_4),\ (v_4, v_5),\ (v_6, v_7) \,\}$,
- $\mathbf{ls}^{\mathcal{B}} = \{v_5, v_7, v_8\}$.

Note that the root node of T is not included in any sibling relation. ⌟

2.5 Syntax and Semantics of Datalog

The following definition of datalog, and monadic datalog respectively, is basically taken from [GK04].

A *datalog rule* is an expression of the form

$$h \leftarrow b_1, \ldots, b_n,$$

for $n \in \mathbb{N}_{\geqslant 1}$, where $h, b_1, \ldots, b_n$ are called *atoms* of the rule, h is called the rule's *head*, and $b_1, \ldots, b_n$ (understood as a conjunction of atoms) is called the *body*. Each atom is of the form $P(x_1, \ldots, x_m)$ where P is a predicate of some arity $m \in \mathbb{N}_{\geqslant 1}$ and $x_1, \ldots, x_m$ are variables. Rules are required to be *safe* in the sense that all variables appearing in the head also have to appear in the body.

A *datalog program* is a finite set of datalog rules. Let $\mathcal{P}$ be a datalog program and let r be a datalog rule. We write $\mathrm{var}(r)$ for the set of all variables occurring in the rule r, and we let $\mathrm{var}(\mathcal{P}) := \bigcup_{r \in \mathcal{P}} \mathrm{var}(r)$. Predicates that occur in the head of a rule of $\mathcal{P}$ are called *intensional*, whereas predicates that only occur in the body of rules of $\mathcal{P}$ are called *extensional*. We write $\mathrm{idb}(\mathcal{P})$ and $\mathrm{edb}(\mathcal{P})$ to denote the sets of intensional and extensional predicates of $\mathcal{P}$, and we say that $\mathcal{P}$ *is of schema* τ if $\mathrm{edb}(\mathcal{P}) \subseteq \tau$ for a schema τ. A datalog program belongs to *monadic datalog* (mDatalog, for short), if all its *intensional* predicates have arity 1.

For defining the semantics of datalog, let τ be a schema, let $\mathcal{P}$ be a datalog program of schema τ, let A be a domain, and let

$$F_{\mathcal{P},A} \;:=\; \{\; R(a_1, \ldots, a_r) \,:\, R \in \tau \cup \mathrm{idb}(\mathcal{P}),\ r = ar(R),\ a_1, \ldots, a_r \in A \;\}$$

be the set of all possible *atomic facts over* A. A *valuation* β *for* $\mathcal{P}$ *in* A is a function $\beta : \big(\mathrm{var}(\mathcal{P}) \cup A\big) \to A$ where $\beta(a) = a$ for all $a \in A$. For an atom $P(x_1, \ldots, x_m)$ occurring in a rule of $\mathcal{P}$ we let

$$\beta\big(P(x_1, \ldots, x_m)\big) \;:=\; P\big(\beta(x_1), \ldots, \beta(x_m)\big).$$

The *immediate consequence operator* $\mathcal{T}_\mathcal{P} : 2^{F_{\mathcal{P},A}} \to 2^{F_{\mathcal{P},A}}$ induced by the datalog program $\mathcal{P}$ on domain A maps every $C \subseteq F_{\mathcal{P},A}$ to

$$\mathcal{T}_\mathcal{P}(C) \;:=\; C \;\cup\; \left\{ \beta(h) \;:\; \begin{array}{l} \text{there is a rule } h \leftarrow b_1, \ldots, b_n \text{ in } \mathcal{P} \\ \text{and a valuation } \beta \text{ for } \mathcal{P} \text{ in } A \text{ such} \\ \text{that } \beta(b_1), \ldots, \beta(b_n) \in C \end{array} \right\}.$$

Clearly, $\mathcal{T}_\mathcal{P}$ is *monotone*, i.e., for $C \subseteq D \subseteq F_{\mathcal{P},A}$ we have $\mathcal{T}_\mathcal{P}(C) \subseteq \mathcal{T}_\mathcal{P}(D)$.

Letting $\mathcal{T}^0_\mathcal{P}(C) := C$ and $\mathcal{T}^{i+1}_\mathcal{P}(C) := \mathcal{T}_\mathcal{P}\big(\mathcal{T}^i_\mathcal{P}(C)\big)$ for all $i \in \mathbb{N}$, it is straightforward to see that

$$C = \mathcal{T}^0_\mathcal{P}(C) \subseteq \mathcal{T}^1_\mathcal{P}(C) \subseteq \cdots \subseteq \mathcal{T}^i_\mathcal{P}(C) \subseteq \mathcal{T}^{i+1}_\mathcal{P}(C) \subseteq \cdots \subseteq F_{\mathcal{P},A}.$$

For a finite domain A, the set $F_{\mathcal{P},A}$ is finite, and hence there is an $i_0 \in \mathbb{N}$ such that $\mathcal{T}^{i_0}_\mathcal{P}(C) = \mathcal{T}^i_\mathcal{P}(C)$ for all $i \geqslant i_0$. In particular, the set $\mathcal{T}^\omega_\mathcal{P}(C) := \mathcal{T}^{i_0}_\mathcal{P}(C)$ is a *fixpoint* of the immediate consequence operator $\mathcal{T}_\mathcal{P}$. By Knaster and Tarski's theorem we know that this fixpoint is the *smallest* fixpoint of $\mathcal{T}_\mathcal{P}$ which contains C.

Theorem 2.3 (Knaster and Tarski [Tar55])**.** *Let* τ *be a schema, let* $\mathcal{P}$ *be a datalog program of schema* τ*, and let* A *be a finite domain. For every* $C \subseteq F_{\mathcal{P},A}$ *we have*

$$\begin{aligned} \mathcal{T}^\omega_\mathcal{P}(C) \;&=\; \bigcap \{\, D \;:\; \mathcal{T}_\mathcal{P}(D) = D \;\; \textit{and} \;\; C \subseteq D \subseteq F_{\mathcal{P},A} \,\} \\ &=\; \bigcap \{\, D \;:\; \mathcal{T}_\mathcal{P}(D) \subseteq D \;\; \textit{and} \;\; C \subseteq D \subseteq F_{\mathcal{P},A} \,\}. \end{aligned}$$

⌟

A *k-ary datalog query* of schema τ is a tuple $Q = (\mathcal{P}, P)$ where $\mathcal{P}$ is a datalog program of schema τ and P is an (intensional or extensional) predicate of arity k occurring in $\mathcal{P}$. $\mathcal{P}$ and P are called the *program* and the *query predicate* of Q. A k-ary *monadic* datalog query is a k-ary datalog query $Q = (\mathcal{P}, P)$ where $\mathcal{P}$ is a *monadic* datalog program. When evaluated in a finite τ-structure $\mathcal{A}$, the query Q results in the following k-ary relation over A:

$$Q(\mathcal{A}) \;:=\; \{\, (a_1, \ldots, a_k) \in A^k \;:\; P(a_1, \ldots, a_k) \;\in\; \mathcal{T}^\omega_\mathcal{P}\big(\mathit{atoms}(\mathcal{A})\big) \,\}.$$

Unary queries are queries of arity $k = 1$. On Σ-labeled (un)ordered trees, a unary query Q assigns to each (un)ordered Σ-labeled tree T a set $Q(T) \subseteq V^T$ of nodes.

The *Boolean monadic datalog query* Q_{Bool} specified by an unary monadic datalog query $Q = (\mathcal{P}, P)$ is the Boolean query with $Q_{\mathit{Bool}}(T) = \mathbf{yes}$ if and only if the tree's root node belongs to $Q(T)$.

The *size* $\|Q\|$ of a monadic datalog query Q is the length of $Q = (\mathcal{P}, P)$ viewed as a string over a suitable alphabet.

Example 2.4. *Consider the schema $\tau_{GK,\Sigma}$ introduced in Section 2.4.2 for representing ordered Σ-labeled trees for $\Sigma = \{Black,\ White\}$. We present a unary monadic datalog query $Q = (\mathcal{P}, Ans)$ of schema $\tau_{GK,\Sigma}$ such that for every ordered Σ-labeled tree T we have*

$$Q\big(\mathcal{S}_{GK}(T)\big) \quad = \quad \begin{cases} \{\, root^T \,\} & \text{if the number of White labeled} \\ & \text{children of } T\text{'s root is exactly two,} \\ \\ \varnothing & \text{otherwise.} \end{cases}$$

To this end, we let $\mathcal{P}$ consist of the following rules:

$$\begin{aligned}
Ans(x) &\leftarrow \mathbf{root}(x),\ \mathbf{fc}(x,y),\ White_2(y) \\
White_2(x) &\leftarrow \mathbf{label}_{Black}(x),\ \mathbf{ns}(x,y),\ White_2(y) \\
White_2(x) &\leftarrow \mathbf{label}_{White}(x),\ \mathbf{ns}(x,y),\ White_1(y) \\
White_1(x) &\leftarrow \mathbf{label}_{Black}(x),\ \mathbf{ns}(x,y),\ White_1(y) \\
White_1(x) &\leftarrow \mathbf{label}_{White}(x),\ \mathbf{ns}(x,y),\ White_0(y) \\
White_0(x) &\leftarrow \mathbf{label}_{Black}(x),\ \mathbf{ns}(x,y),\ White_0(y) \\
White_1(x) &\leftarrow \mathbf{label}_{White}(x),\ \mathbf{ls}(x) \\
White_0(x) &\leftarrow \mathbf{label}_{Black}(x),\ \mathbf{ls}(x)
\end{aligned}$$

The program starts on every last sibling and 'counts' in the reverse order of $order^T$ the occurrences of White-labeled nodes. It gets stuck if there are more then two. But if there are exactly two occurrences, then the first node of all the siblings will be marked by $White_2$ and so, by the first rule, we have $Ans(root^T) \in \mathcal{T}^{\omega}_{\mathcal{P}}(T)$ if the root of T has exactly two children labeled with the symbol White.

In particular, Q returns $\{root^T\}$ on the tree from Example 2.2; and the Boolean Query $Q_{Bool} = (\mathcal{P}, Ans)$ answers on this input with **yes**. ⌟

Next, we present an alternative definition of the semantics of datalog. We will show that the *proof-theoretic semantics* is equivalent to the aforementioned semantics that is called *fixpoint semantics*. Having both definitions gives us the freedom to choose in ongoing proofs the variant that is more suitable.

The main idea of the proof-theoretic semantics is that the answer of a datalog program consists of the set of facts that can be proven from the atomic facts using the rules of the program as proof rules in the given structure. A proof of such an atomic fact is represented by a *proof tree*.

Let A be a domain. For $k \in \mathbb{N}$ and $a_1, \ldots, a_k \in A$, a *proof tree* $T_{\mathcal{P},\mathcal{A}}$ of the fact $P(a_1, \ldots, a_k)$ from the datalog program $\mathcal{P}$ and the finite structure $\mathcal{A}$ with universe A is a finite labeled tree where

- each vertex of the tree is labeled by an atomic fact,
- the root is labeled by $P(a_1, \ldots, a_k)$,
- each leaf is labeled by an atomic fact from $atoms(\mathcal{A})$, and
- for each non-leaf vertex v with children $v_1, \ldots, v_l$ with $l \in \mathbb{N}$ there exists a rule $h(x) \leftarrow b_1(x_1), \ldots, b_l(x_l)$ in $\mathcal{P}$ and a valuation β such that:

(1) v is labeled by $h(\beta(x))$

(2) the child v_i is labeled by $b_i(\beta(x_i))$ for every $1 \leqslant i \leqslant l$.

Example 2.5. *Consider the monadic datalog query Q presented in Example 2.4. Let T be the ordered Σ-labeled tree from Example 2.2.*

The depicted tree is a proof tree of the fact $Ans(v_0)$ from $\mathcal{P}$ and T. As v_0 is the root of T it witness the answer **yes** *of $Q_{Bool} = (\mathcal{P}, Ans)$ on input T.*

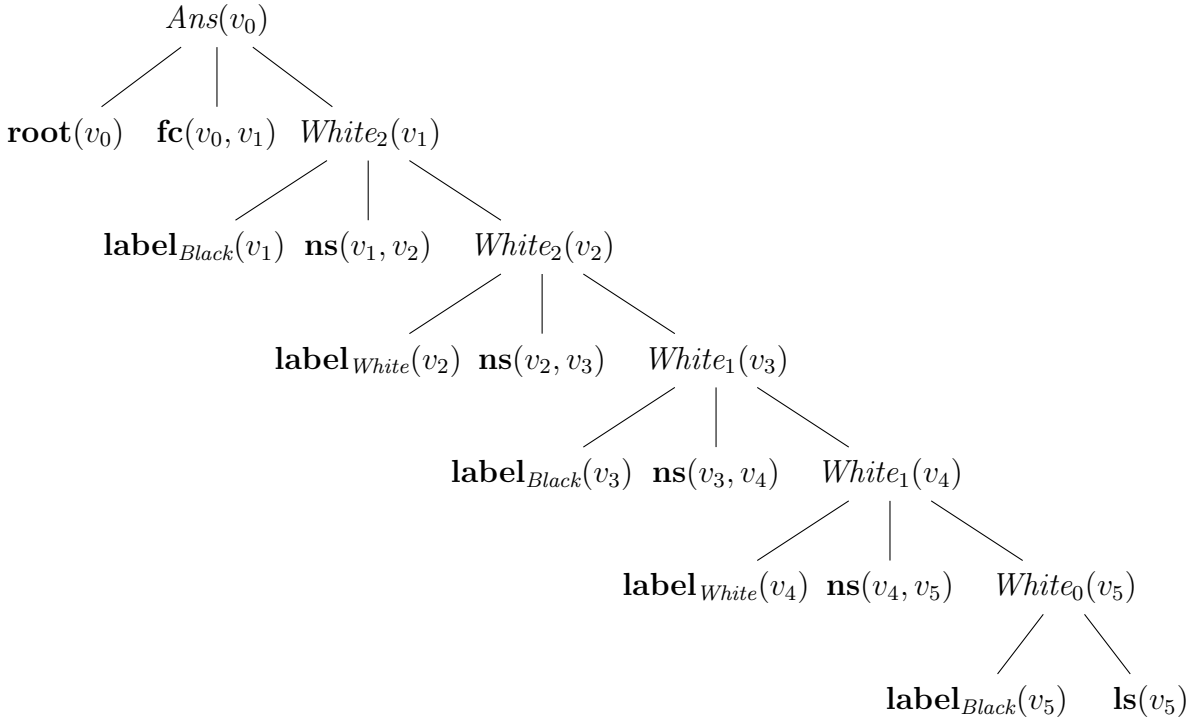

Figure 2.2: A proof tree of the fact $Ans(v_0)$ from $\mathcal{P}$ and T.

The equivalence of both semantics is a consequence of the following proposition.

Proposition 2.6 (Folklore)**.** *Let A be a domain. Let $(a_1, \ldots, a_k) \in A^k$, $k \in \mathbb{N}$. For each datalog program $\mathcal{P}$, all finite structures $\mathcal{A}$ and all atomic facts $P(a_1, \ldots, a_k) \in F_{\mathcal{P},A}$ it holds:*

$$P(a_1, \ldots, a_k) \in \mathcal{T}_{\mathcal{P}}^{\omega}(atoms(\mathcal{A})) \qquad \text{if and only if} \qquad \begin{array}{c}\text{there exists a proof tree} \\ \text{for } P(a_1, \ldots, a_k) \text{ from } \mathcal{P} \text{ and } \mathcal{A}.\end{array}$$

Proof. For the *only if*-direction, we show by induction over the stages of the fixpoint process, that for every fact $P(a_1, \ldots, a_k) \in \mathcal{T}_{\mathcal{P}}^{\omega}(atoms(\mathcal{A}))$ there is a proof tree of (at most) height i.
(Base case.) Let the atomic fact $P(a_1, \ldots, a_k) \in T_{\mathcal{P}}^{0}(atoms(\mathcal{A}))$. Then, by definition the fact is an element of $atoms(\mathcal{A})$. Now, the tree consisting of only one node labeled by $P(a_1, \ldots, a_k)$ is a proof tree for $P(a_1, \ldots, a_k)$ from $\mathcal{P}$ and $\mathcal{A}$.
(Induction step.) Assuming for each fact in $\mathcal{T}_{\mathcal{P}}^{i}(atoms(\mathcal{A}))$ there exists a proof tree of height (at most) i. Let $P(a_1, \ldots, a_k) \in T_{\mathcal{P}}^{i+1}(atoms(\mathcal{A}))$. If $P(a_1, \ldots, a_k)$ is already an

element of $T^i_{\mathcal{P}}(atoms(\mathcal{A}))$ then, by hypothesis, there exists a proof tree for $P(a_1, \ldots, a_k)$ of height less than $i + 1$.

Otherwise, we have $P(a_1, \ldots, a_k) \in T^{i+1}_{\mathcal{P}}(atoms(\mathcal{A})) \setminus T^i_{\mathcal{P}}(atoms(\mathcal{A}))$. Now, by the definition of the $\mathcal{T}_{\mathcal{P}}$ operator there exists a rule $h \leftarrow b_1, \ldots, b_l$ and a valuation β such that $\beta(b_1), \ldots, \beta(b_l) \in T^i_{\mathcal{P}}(atoms(\mathcal{A}))$ and $P(a_1, \ldots, a_k) = \beta(h)$. By induction hypothesis there exists for every j, $1 \leqslant j \leqslant l$ a proof tree T_j for the atomic fact $\beta(b_j)$ from $\mathcal{P}$ and $\mathcal{A}$ of height (at most) i. Now the proof tree consisting of the root $P(a_1, \ldots, a_k)$ connected with the root nodes of these proof trees T_j as children, forms a proof tree for fact $P(a_1, \ldots, a_k)$ from $\mathcal{P}$ and $\mathcal{A}$.

For the converse direction, we proceed by an induction over the height of the proof tree of the fact $P(a_1, \ldots, a_k)$ from $\mathcal{P}$ and $\mathcal{A}$.
(Base case.) Assuming the proof tree for an atomic fact $P(a_1, \ldots, a_k)$ has only one node. As this node is a leaf, the fact must be an element of $atoms(\mathcal{A})$. Then, by definition, $P(a_1, \ldots, a_k)$ is an element of $\mathcal{T}^0_{\mathcal{P}}(atoms(\mathcal{A}))$ and therefore, $P(a_1, \ldots, a_k)$ is an element of $\mathcal{T}^\omega_{\mathcal{P}}(atoms(\mathcal{A}))$.
(Induction step.) Assuming for every fact witnessed by a proof tree from $\mathcal{P}$ and $\mathcal{A}$ of height i it holds that the atomic fact is an element of $\mathcal{T}^\omega_{\mathcal{P}}(atoms(\mathcal{A}))$. Let T be a proof tree for the fact $P(a_1, \ldots, a_k)$ from $\mathcal{P}$ and $\mathcal{A}$ of height $i+1$. For some $l \in \mathbb{N}$, let $v_1, \ldots, v_l$ be the children of the root node v of T. Let $P_1(\vec{a}_1), \ldots, P_l(\vec{a}_l)$ be the labels of the children nodes. The subtrees $T_1, \ldots, T_l$ rooted at the children of v, themselves are proof trees for the facts $P_1(\vec{a}_1), \ldots, P_l(\vec{a}_l)$. By induction hypothesis, these facts are elements of $\mathcal{T}^\omega_{\mathcal{P}}(atoms(\mathcal{A}))$. As v is the parent node of $v_1, \ldots, v_l$ in the proof tree for the fact $P(a_1, \ldots, a_k)$, there exists a rule $h(x) \leftarrow b_1(x_1), \ldots, b_l(x_l)$ in $\mathcal{P}$ and a valuation β such that v is labeled by $h(\beta(x))$ and the children $v_1, \ldots, v_l$ are labeled by $b_1(\beta(x_1)) = P_1(\vec{a}_1), \ldots, b_l(\beta(x_l)) = P_l(\vec{a}_l)$. By the definition of the $\mathcal{T}_{\mathcal{P}}$ operator this implies that $h(\beta(x)) = P(a_1, \ldots, a_k)$ is contained in $\mathcal{T}^\omega_{\mathcal{P}}(atoms(\mathcal{A}))$. □

Now, let us recall two useful properties of mDatalog queries.

Remark 2.7 (Folklore)**.** *The monotonicity of the immediate consequence operator implies that datalog queries Q of schema τ are monotone in the following sense: If $\mathcal{A}$ and $\mathcal{B}$ are τ-structures with $atoms(\mathcal{A}) \subseteq atoms(\mathcal{B})$, then $Q(\mathcal{A}) \subseteq Q(\mathcal{B})$.*

Proof. Let $Q = (\mathcal{P}, P)$ be a datalog query of schema τ, let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures with $atoms(\mathcal{A}) \subseteq atoms(\mathcal{B})$. The proof is by an easy induction over the fixpoint process.
(Base case.) $\mathcal{T}^0_{\mathcal{P}}(atoms(\mathcal{A})) = atoms(\mathcal{A}) \subseteq atoms(\mathcal{B}) = \mathcal{T}^0_{\mathcal{P}}(atoms(\mathcal{B}))$.
(Inductive step.) Let $i \in \mathbb{N}$. By induction, we have $\mathcal{T}^i_{\mathcal{P}}(atoms(\mathcal{A})) \subseteq \mathcal{T}^i_{\mathcal{P}}(atoms(\mathcal{B}))$. Therefore, by monotonicity,

$$\mathcal{T}^{i+1}_{\mathcal{P}}(atoms(\mathcal{A})) = \mathcal{T}_{\mathcal{P}}(\mathcal{T}^i_{\mathcal{P}}(atoms(\mathcal{A})) \subseteq \mathcal{T}_{\mathcal{P}}(\mathcal{T}^i_{\mathcal{P}}(atoms(\mathcal{B})) = \mathcal{T}^{i+1}_{\mathcal{P}}(atoms(\mathcal{B})).$$ □

The second property is the well-known fact that datalog is *preserved under homomorphisms* in the following sense. A *homomorphism* from a τ-structure $\mathcal{A}$ to a τ-structure $\mathcal{B}$ is a mapping $h : A \to B$ such that for all $R \in \tau$ and all tuples $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$ we have $(h(a_1), \ldots, h(a_r)) \in R^{\mathcal{B}}$. As a shorthand, for any set $S \subseteq A^k$ we let $h(S) = \{\big(h(a_1), \ldots, h(a_k)\big) : (a_1, \ldots, a_k) \in S\}$.

Lemma 2.8 (Folklore)**.** *Any k-ary datalog query Q of schema τ is preserved under homomorphisms in the following sense: If $\mathcal{A}$ and $\mathcal{B}$ are τ-structures, and h is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, then $h\big(Q(\mathcal{A})\big) \subseteq Q(\mathcal{B})$.*

Proof. Let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures and let $h : A \to B$ be a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. Furthermore, let $Q = (\mathcal{P}, P)$ where $\mathcal{P}$ is a datalog program of schema τ. For an atomic fact $f = R(a_1, \ldots, a_r) \in F_{\mathcal{P},A}$, let $h(f) := R(h(a_1), \ldots, h(a_r))$ be the corresponding atomic fact in $F_{\mathcal{P},B}$. Furthermore, for a set $S \subseteq F_{\mathcal{P},A}$, let $h(S) := \{h(f) : f \in S\}$ be the corresponding subset of $F_{\mathcal{P},B}$.

First, note that by the definition of the immediate consequence operator $\mathcal{T}_{\mathcal{P}}$ it is straightforward to see that the following is true: If $C \subseteq F_{\mathcal{P},A}$ and $D \subseteq F_{\mathcal{P},B}$ such that $h(C) \subseteq D$, then $h(\mathcal{T}_{\mathcal{P}}(C)) \subseteq \mathcal{T}_{\mathcal{P}}(D)$.

Next, note that this immediately implies that the following is true: If $C \subseteq F_{\mathcal{P},A}$ and $D \subseteq F_{\mathcal{P},B}$ such that $h(C) \subseteq D$, then $h\big(\mathcal{T}^{\omega}_{\mathcal{P}}(C)\big) \subseteq \mathcal{T}^{\omega}_{\mathcal{P}}(D)$.

Finally, note that h is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, and thus $h\big(\mathit{atoms}(\mathcal{A})\big) \subseteq \mathit{atoms}(\mathcal{B})$. Consequently, $h\big(\mathcal{T}^{\omega}_{\mathcal{P}}\big(\mathit{atoms}(\mathcal{A})\big)\big) \subseteq \mathcal{T}^{\omega}_{\mathcal{P}}\big(\mathit{atoms}(\mathcal{B})\big)$. In particular, this means that $h\big(Q(\mathcal{A})\big) \subseteq Q(\mathcal{B})$. □

2.6 A Note on Conjunctive Queries

In this section, we take a brief look at the datalog programs consisting of only one non recursive rule.[3] Each of these rules can be read as a *conjunctive query*. Formally, for a schema τ, a conjunctive query Q (CQ(τ)-query, for short) is an expression of the form

$$\mathit{Ans}(u) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_n)$$

where $n \in \mathbb{N}_{\geqslant 1}$, $\{R_1, \ldots, R_n\} \subseteq \tau$, $u, u_1, \ldots, u_n$ are free tuples,[4] $ar(u_i) = ar(R_i)$ for every $i \in \{1, \ldots, n\}$, and finally, every variable in u must occur at least once in $u_1, \ldots, u_n$. The set $\mathrm{var}(Q)$ is the set of all variables occurring in Q. The semantics for a given conjunctive query Q in a τ-structure $\mathcal{A}$ with universe A is defined by

$$Q(\mathcal{A}) \;:=\; \left\{ \beta(u) \;\middle|\; \begin{array}{l} \beta \text{ is a valuation for } Q \text{ in } A \text{ and} \\ \beta(R_i(u_i)) \in \mathit{atoms}(\mathcal{A}) \text{ for all } i \in \{1, \ldots, n\} \end{array} \right\}.$$

Note, that conjunctive queries are not syntactically contained in datalog queries. Indeed for $k \in \mathbb{N}_{\geqslant 1}$, the k-ary conjunctive query Q_{CQ} of the form

$$\mathit{Ans}(u) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_n)$$

is equivalent to the k-ary datalog query $Q = (\mathcal{P}, \mathit{Ans})$ where

$$\mathcal{P} \;:=\; \{\; \mathit{Ans}(u) \leftarrow R_1(u_1), \ldots, R_n(u_n) \;\}.$$

A *Boolean* conjunctive query Q of the form

$$\mathit{Ans}() \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_n)$$

evaluated on a τ-structure $\mathcal{A}$ yields **yes** if and only if there exists a valuation β for Q in A with $\beta(R_i(u_i)) \in \mathit{atoms}(\mathcal{A})$ for all i in $\{1, \ldots, n\}$.

To translate Boolean conjunctive queries on trees into datalog, we use the following lemma.

[3] A datalog rule r is non recursive if its head predicate does not occur in its body.

[4] This means that every component is either a variable or a constant.

Lemma 2.9. *Let τ be one of the schemas τ_u^N and τ_o^M where N is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}, \mathbf{desc}\}$ and M is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}, \mathbf{child}, \mathbf{desc}, \mathbf{ls}\}$.*

For every Boolean CQ(τ)-query Q_{CQ} there exists an unary mDatalog(τ)-query $Q = (\mathcal{P}, P)$, such that for every finite labeled tree T corresponding to τ we have

$$Q_{CQ}(T) = Q_{Bool}(T).$$

The mDatalog-query Q can be obtained from Q_{CQ} in time linear in the size of Q_{CQ}.

Proof. Let Q_{CQ} be a conjunctive query of the form

$$Ans() \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_n)$$

for an $n \in \mathbb{N}_{\geqslant 1}$. Furthermore, let $x \in \mathrm{var}(Q_{\mathrm{CQ}})$ be a variable occurring in Q. Now, if τ is a schema over unordered trees then the desired mDatalog(τ)-query $Q = (\mathcal{P}, P)$ consists of the following program

$$\mathcal{P} := \left\{ \begin{array}{ll} P(x) & \leftarrow \mathbf{child}(x, y), P(y) \\ P(x) & \leftarrow R_1(u_1), \ldots, R_n(u_n) \end{array} \right\}.$$

Otherwise, if τ is a schema over ordered trees then the program $\mathcal{P}$ consists of the following rules.

$$\mathcal{P} := \left\{ \begin{array}{ll} P(x) & \leftarrow \mathbf{fc}(x, y), P(y) \\ P(x) & \leftarrow \mathbf{ns}(x, y), P(y) \\ P(x) & \leftarrow R_1(u_1), \ldots, R_n(u_n) \end{array} \right\}.$$

It is straightforward to see that $Q_{Bool}(T) = \mathbf{yes}$ if and only if $Q_{CQ}(T) = \mathbf{yes}$ for all finite labeled trees T. □

2.7 Considered Problems during a Static Analysis

As we have seen in Chapter 1 it is fundamental to answer the three following questions for a considered database language and arbitrary queries Q, Q_1, and Q_2 during a static analysis.

(1.) Does query Q produce the empty result on every database D?

(2.) Does query Q_1 produce the same result as query Q_2 on every database D?

(3.) Is the result of Q_1 on every database D a subset of the result of Q_2 on D?

As usual, we formalize such questions as decision problems dealing with the considered languages and databases. To this end, let τ be one of the schemas introduced in Section 2.4.1 or 2.4.2 for representing (unordered or ordered) Σ-labeled trees as a relational structures.

Addressing question (1.), a unary query Q such that for every Σ-labeled tree T we have $Q(T) = \varnothing$ is called *unsatisfiable*, and we write $Q = \varnothing$ to indicate that Q is unsatisfiables. If Q is a Boolean query of schema τ, we write $Q = \varnothing$ to indicate that there is no Σ-labeled tree T, such that $Q(T) = \mathbf{yes}$. We write $Q \neq \varnothing$ to indicate that $Q = \varnothing$ does not hold.

Example 2.10. *For every non-empty alphabet Σ and every unordered Σ-labeled tree T the result of the query $Q^{\varnothing}_{\tau_u} = (\mathcal{P}_u, P_u)$ with*

$$\mathcal{P}_u := \{P_u(x) \leftarrow \mathbf{child}(x, x)\}$$

is empty. Analogously, for every non-empty alphabet Σ and every ordered Σ-labeled tree T the result of the query $Q^{\varnothing}_{\tau_o} = (\mathcal{P}_o, P_o)$ with

$$\mathcal{P}_o := \{P_o(x) \leftarrow \mathbf{fc}(x, x)\}$$

is empty. ⌟

The corresponding decision problem is called the *emptiness problem* and defined as follows.

The emptiness problem for mDatalog(τ) on labeled trees	
Input:	A finite alphabet Σ and an unary (or Boolean) mDatalog(τ_Σ)-query Q.
Question:	Is $Q = \varnothing$?

Addressing question (2.), for two unary queries Q_1 and Q_2 of schema τ we write $Q_1 \equiv Q_2$ to indicate that for every Σ-labeled tree T we have $Q_1(T) = Q_2(T)$. Similarly, if Q_1 and Q_2 are Boolean queries of schema τ, we write $Q_1 \equiv Q_2$ to indicate that for every Σ-labeled tree T we have $Q_1(T) = \mathbf{yes}$ if and only if $Q_2(T) = \mathbf{yes}$. We write $Q_1 \not\equiv Q_2$ to indicate that $Q_1 \equiv Q_2$ does not hold.

The corresponding decision problem is called the *equivalence problem* and defined as follows.

The equivalence problem for mDatalog(τ) on labeled trees	
Input:	A finite alphabet Σ and two unary (or Boolean) mDatalog(τ_Σ)-queries Q_1 and Q_2.
Question:	Is $Q_1 \equiv Q_2$?

For the remaining task (3.) and two unary queries Q_1 and Q_2 of schema τ we write $Q_1 \subseteq Q_2$ to indicate that for every Σ-labeled tree T we have $Q_1(T) \subseteq Q_2(T)$. Similarly, if Q_1 and Q_2 are Boolean queries of schema τ, we write $Q_1 \subseteq Q_2$ to indicate that for every Σ-labeled tree T, if $Q_1(T) = \mathbf{yes}$ then also $Q_2(T) = \mathbf{yes}$. We write $Q_1 \not\subseteq Q_2$ to indicate that $Q_1 \subseteq Q_2$ does not hold. The *query containment problem* (QCP, for short) is defined as follows:

The QCP for mDatalog(τ) on labeled trees	
Input:	A finite alphabet Σ and two unary (or Boolean) mDatalog(τ_Σ)-queries Q_1 and Q_2.
Question:	Is $Q_1 \subseteq Q_2$?

For $\mathbb{P}$ being one of these problems, we often write "$\mathbb{P}$ for unary mDatalog(τ)" to denote that only unary queries are considered. Analogously we write "$\mathbb{P}$ for Boolean mDatalog(τ)" to indicate that only Boolean queries are considered. Sometimes, we also restrict these problems to ranked/unranked and ordered/unordered trees.

Chapter 3

The Logic Point of View:
Expressive Power and Decidability

In the first part of this chapter we will consider the expressive power of monadic datalog on the presented tree structures by using the introduced schemas. A difference in the expressive power is often accompanied by a difference in the complexity of a considered problem, but we will see later on this is not necessarily the case.

The second part is dedicated to the decidability of our problems. Since we know about the decidability of the query containment problem on finite structures (in 2-fold exponential time) by Cosmadakis, Gaifman, Kanellakis, and Vardi [CGKV88], we might be tempted to infer the decidability on trees. The canonical way to decide the containment of two monadic datalog queries on trees would be to combine them with a query Q_T stating that an arbitrary finite structure is a tree. And afterwards, to decide the containment problem of the resulting queries on arbitrary finite structure. However, there does not exits such a query Q_T by the following proposition.

Proposition 3.1.

(a) Let M be equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}, \mathbf{desc}\}$. There is no query Q in mDatalog(τ_u^M), such that Q yields **yes** *on a set S of atomic facts of schema τ_u^M if and only if there exists an unordered Σ-labeled tree T such that $S = atoms(S_u^M(T))$.*

(b) Let N be equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}, \mathbf{child}, \mathbf{ls}, \mathbf{desc}\}$. There is no query Q in mDatalog(τ_o^N), such that Q answers **yes** *on a set S of atomic facts of schema τ_o^N if and only if there exists an ordered Σ-labeled tree T such that $S = atoms(S_o^N(T))$.*

Proof. (a) We choose $\Sigma = \{a\}$ and the labeled unordered tree T as illustrated in Figure 3.1 (a) described by

$$atoms(T) = \left\{ \begin{array}{c} \mathbf{child}(v_0, v_1), \mathbf{child}(v_1, v_2), \\ \mathbf{label}_a(v_0), \mathbf{label}_a(v_1), \mathbf{label}_a(v_2) \end{array} \right\}.$$

We assume the mDatalog(τ_u)-query Q can decide whether a given structure describes a tree or not. Therefore, Q yields **yes** on $atoms(T)$. If we add the atom $\mathbf{child}(v_2, v_0)$ to $atoms(T)$ then the structure becomes cyclic and by assumption Q answers **no** on the input $atoms(T) \cup \{\mathbf{child}(v_2, v_0)\}$ (cf. Figure 3.1 (b)). But this is a contradiction

to the monotonicity of (monadic) datalog (cf. Remark 2.7) since we have $atoms(T) \subseteq atoms(T) \cup \{\mathbf{child}(v_2, v_0)\}$ and so the assumption must be false.

For extensions of τ_u to τ_u^M with $M \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{desc}\}$ and the appropriate extensions of $atoms(T)$ the proof proceeds in the same way.

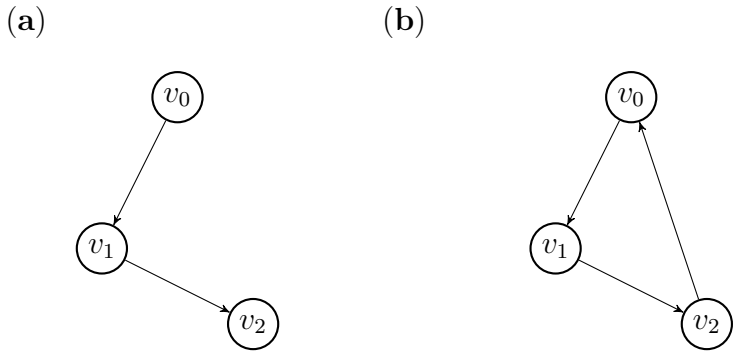

Figure 3.1: (a) The tree T and (b) its extension to a non-tree structure.

(b) The proof proceeds analogously to (a) with the sets

$$atoms(T') = \left\{ \begin{array}{c} \mathbf{fc}(v_0, v_1), \mathbf{fc}(v_1, v_2), \\ \mathbf{label}_a(v_0), \mathbf{label}_a(v_1), \mathbf{label}_a(v_2) \end{array} \right\}.$$

and $atoms(T') \cup \mathbf{fc}(v_2, v_0)$. □

3.1 Monadic Second-Order Logic (MSO)

The set MSO(τ) of all monadic second-order formulas of schema τ is defined as usual, cf. e.g. [Lib04]: There are two kinds of variables, namely *node variables*, denoted by lower-case letters x, y, ..., x_1, x_2, ... ranging over elements of the domain, and *set variables*, denoted by upper-case letters X, Y, ..., X_1, X_2, ... ranging over sets of elements of the domain.
An *atomic* MSO(τ)-formula is of the form

(A1) $R(x_1, \ldots, x_r)$, where $R \in \tau$, $r = ar(R)$, and $x_1, \ldots, x_r$ are node variables,

(A2) $x = y$, where x and y are node variables, or

(A3) $X(x)$, where x is a node variable and X is a set variable.

If x is a node variable, X a set variable, and φ and ψ are MSO(τ)-formulas, then

(BC) $\neg\varphi$ and $(\varphi \vee \psi)$ are MSO(τ)-formulas,

(Q1) $\exists x\varphi$ and $\forall x\varphi$ are MSO(τ)-formulas,

(Q2) $\exists X\varphi$ and $\forall X\varphi$ are MSO(τ)-formulas.

Quantifiers of the form (Q1) are called *first-order quantifiers*; quantifiers of the form (Q2) are called *set quantifiers*. MSO(τ)-formulas without set quantifiers are called *first-order formulas* (FO(τ)-*formulas*, for short). The *size* $\|\varphi\|$ of a formula φ is the length of φ viewed as a string over the alphabet

$$\tau \cup \{x, y, z, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \cup \{(,)\} \cup \{=, \neg, \vee, \exists, \forall\} \cup \{,\}.$$

As abbreviations we use the Boolean connectives $(\varphi \wedge \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$, the statement $x \neq y$ for node variables, and the statements $X = Y$, $X \neq Y$, and $X \subseteq Y$ for set variables. Note that all these can easily be expressed in first-order logic. To improve readability of formulas, we will sometimes add or omit parentheses if no ambiguity arises.

By *free*(φ) we denote the set of (node or set) variables that occur freely (i.e., not within the range of a node or set quantifier) in φ. A *sentence* is a formula without free variables. We write $\varphi(x_1, \ldots, x_k, X_1, \ldots, X_\ell)$ to indicate that φ has k free node variables $x_1, \ldots, x_k$ and ℓ free set variables $X_1, \ldots, X_\ell$. For a τ-structure $\mathcal{A}$, elements $a_1, \ldots, a_k \in A$, and sets $A_1, \ldots, A_\ell \subseteq A$, we write $\mathcal{A} \models \varphi(a_1, \ldots, a_k, A_1, \ldots, A_\ell)$ to indicate that $\mathcal{A}$ satisfies the formula φ when interpreting the free occurrences of the variables $x_1, \ldots, x_k, X_1, \ldots, X_\ell$ with $a_1, \ldots, a_k, A_1, \ldots, A_\ell$. A formula $\varphi(x_1, \ldots, x_k)$ with k free node variables and no free set variable defines a k-ary query on τ-structures which, when evaluated in a τ-structure $\mathcal{A}$, results in the k-ary relation

$$\varphi(\mathcal{A}) \;:=\; \{\; (a_1, \ldots, a_k) \in A^k \;:\; \mathcal{A} \models \varphi(a_1, \ldots, a_k) \;\}.$$

Example 3.2. *Consider the schema $\tau_{u,\Sigma}$ which we introduced in Section 2.4.1 in order to represent unordered Σ-labeled trees. Let $\Sigma = \{Black, White\}$. We present a unary* FO($\tau_{u,\Sigma}$)*-query $\varphi(x)$ such that for every unordered Σ-labeled tree T we have*

$$\varphi\big(\mathcal{S}_u(T)\big) \;=\; \begin{cases} \{\, root^T \,\} & \text{if the number of White labeled} \\ & \text{children of } T\text{'s root is exactly two,} \\ \varnothing & \text{otherwise.} \end{cases}$$

The FO($\tau_{u,\Sigma}$)*-formula $\varphi(x)$ is as follows:*

$$\neg\exists u\, \mathbf{child}(u, x) \;\wedge$$
$$\exists y\, \exists z \Big(y \neq z \;\wedge \mathbf{child}(x, y) \wedge \mathbf{child}(x, z) \wedge \mathbf{label}_{White}(y) \wedge \mathbf{label}_{White}(z) \wedge$$
$$\forall v \Big(\mathbf{child}(x, v) \;\rightarrow\; (v = y \vee v = z \vee \neg\, \mathbf{label}_{White}(v)) \Big) \Big) \qquad \lrcorner$$

A $\forall\exists$-MSO(τ)-formula is an MSO(τ)-formula of the form

$$\forall X_1 \cdots \forall X_m \; \exists x_1 \cdots \exists x_k \; \xi$$

where $m, k \in \mathbb{N}$, $X_1, \ldots, X_m$ are set variables, $x_1, \ldots, x_k$ are node variables, and ξ is a formula that does not contain any (node or set) quantifier.

It is well-known that unary monadic datalog queries can be translated into equivalent $\forall\exists$-MSO queries.

Proposition 3.3 (Folklore, cf.[GK04]). *Let τ be a schema. For every unary monadic datalog query $Q = (\mathcal{P}, P)$ of schema τ there exists a $\forall\exists$-MSO(τ)-formula $\varphi(x)$ such that $Q(\mathcal{A}) = \varphi(\mathcal{A})$ is true for any finite τ-structure $\mathcal{A}$.*

Furthermore, there is an algorithm which computes φ from Q in time polynomial in the size of Q.

Proof. Let $\{X_1, \ldots, X_m\} = \text{idb}(\mathcal{P})$ be the set of intensional predicates of $\mathcal{P}$, and w.l.o.g let $X_1 = P$. For every rule r of $\mathcal{P}$ of the form $h \leftarrow b_1, \ldots, b_n,$ with $\{z_1, \ldots, z_k\} = \text{var}(r)$ let

$$\psi_r(X_1, \ldots, X_m) \;:=\; \forall z_1 \cdots \forall z_k \,\big(\, (\, b_1 \wedge \cdots \wedge b_n \,) \;\rightarrow\; h \,\big).$$

Now, let $\chi(X_1, \ldots, X_m) := \bigwedge_{r\in\mathcal{P}} \psi_r(X_1, \ldots, X_m)$. Finally, let x be a node variable that does not occur in $\chi(X_1, \ldots, X_m)$ and let

$$\varphi(x) \;:=\; \forall X_1 \cdots \forall X_m \,\big(\, \chi(X_1, \ldots, X_m) \;\rightarrow\; X_1(x) \,\big).$$

Obviously, $\varphi(x)$ is equivalent, on the class of all τ-structures, to the formula

$$\forall X_1 \cdots \forall X_m \big(X_1(x) \vee \neg\chi\big),$$

and $\neg\chi$ is equivalent to $\bigvee_{r\in\mathcal{P}} \neg\psi_r$ while $\neg\psi_r$ is equivalent to

$$\exists z_1 \cdots \exists z_k \,\neg\big((b_1 \wedge \cdots \wedge b_n) \rightarrow h\big).$$

Thus, it is straightforward to see that $\varphi(x)$ is equivalent to a $\forall\exists$-MSO(τ)-formula and this formula can be constructed in time polynomial in the size of Q.

It remains to verify that $Q(\mathcal{A}) = \varphi(\mathcal{A})$ for every τ-structure $\mathcal{A}$. To this end, let $\mathcal{A}$ be an arbitrary τ-structure. By the construction of $\varphi(x)$ we know for $a \in A$ that

$$a \in \varphi(\mathcal{A}) \iff a \in X_1^{\mathcal{A}'} \text{ for every } \tau \cup \{X_1, \ldots, X_m\}\text{-expansion } \mathcal{A}' \text{ of } \mathcal{A} \text{ with } \mathcal{A}' \models \chi.$$

Now let $C := \mathit{atoms}(\mathcal{A})$. Moreover, consider arbitrary sets $A_1, \ldots, A_m \subseteq A$, let $\mathcal{A}'$ be the $\tau \cup \{X_1, \ldots, X_m\}$-structure obtained as the expansion of $\mathcal{A}$ by $X_i^{\mathcal{A}} := A_i$ for all $i \in \{1, \ldots, m\}$, and let $D := \mathit{atoms}(\mathcal{A}')$. Clearly, $C \subseteq D \subseteq F_{\mathcal{P},A}$. Furthermore, note that χ is constructed in such a way that the following is true:

$$\mathcal{A}' \models \chi \quad\iff\quad \mathcal{T}_{\mathcal{P}}(D) \subseteq D.$$

By the theorem of Knaster and Tarski (Theorem 2.3), we know that

$$\mathcal{T}_{\mathcal{P}}^{\omega}(C) \;=\; \bigcap \{\, D \,:\, \mathcal{T}_{\mathcal{P}}(D) \subseteq D \text{ and } C \subseteq D \subseteq F_{\mathcal{P},A} \,\}.$$

Thus, for $a \in A$ we have

$$\begin{aligned}
& a \in Q(\mathcal{A}) \\
\iff\; & X_1(a) \in \mathcal{T}_{\mathcal{P}}^{\omega}(C) \\
\iff\; & X_1(a) \in D \text{ for every } D \text{ with } \mathcal{T}_{\mathcal{P}}(D) \subseteq D \text{ and } C \subseteq D \subseteq F_{\mathcal{P},A} \\
\iff\; & a \in X_1^{\mathcal{A}'} \text{ for every } \tau \cup \{X_1, \ldots, X_m\}\text{-expansion } \mathcal{A}' \text{ of } \mathcal{A} \text{ with } \mathcal{A}' \models \chi \\
\iff\; & a \in \varphi(\mathcal{A}).
\end{aligned}$$

This completes the proof of Proposition 3.3. □

3.2 Expressive Power of mDatalog on Ordered Trees

Let τ be one of the schemas introduced in Section 2.4.2, i.e., $\tau = \tau_{o,\Sigma}^{M}$ for some $M \subseteq \{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$. We say that a unary query q on Σ-labeled ordered trees is *mDatalog(τ)-definable* if and only if there exists a unary monadic datalog query Q of schema τ such that for every ordered Σ-labeled tree T we have $q(T) = Q(\mathcal{S}_o^M(T))$. Similarly, for any subset L of MSO, q is called *$L(\tau)$-definable* if and only if there exists an $L(\tau)$-formula $\varphi(x)$ such that for every ordered Σ-labeled tree T we have $q(T) = \varphi(\mathcal{S}_o^M(T))$.

Often, we will simply write $Q(T)$ instead of $Q(\mathcal{S}_o^M(T))$, and $\varphi(T)$ instead of $\varphi(\mathcal{S}_o^M(T))$.

3.2.1 Expressive Power on Unranked Ordered Trees

Proposition 3.3 implies that unary queries on Σ-labeled ordered trees which are definable in mDatalog(τ) are also definable in MSO(τ). In [GK04] it was shown that for unranked Σ-labeled ordered trees using the particular schema $\tau = \tau_{GK,\Sigma}$ the converse also holds:

Theorem 3.4 (Gottlob and Koch [GK04])**.** *A unary query on unranked ordered Σ-labeled trees is definable in mDatalog($\tau_{GK,\Sigma}$) if, and only if, it is definable in* MSO($\tau_{GK,\Sigma}$)*. Furthermore, there is an algorithm which translates a given unary mDatalog($\tau_{GK,\Sigma}$)-query into an equivalent unary* MSO($\tau_{GK,\Sigma}$)*-query, and vice versa.* ⌟

In the remainder of this subsection, we will see that adding the **child** and **desc** relations does not increase the expressive power of mDatalog or MSO on ordered Σ-labeled trees. However, omitting any of the relations **root**, **leaf**, or **ls** substantially decreases the expressive power of mDatalog on unranked ordered trees, but does not decrease the expressive power of MSO.

Proposition 3.5 (Folklore)**.** *Let M be the set* $\{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$*. There exist* MSO($\tau_{o,\Sigma}$)*-formulas*

$$\varphi_{\mathbf{child}}(x,y),\ \ \varphi_{\mathbf{desc}}(x,y),\ \ \varphi_{\mathbf{root}}(x),\ \ \varphi_{\mathbf{leaf}}(x),\ \ \varphi_{\mathbf{ls}}(x),$$

such that for every ordered Σ-labeled tree T and all nodes a, b of T we have

$$\begin{array}{lcl}
\mathcal{S}_o(T) \models \varphi_{\mathbf{child}}(a,b) & \iff & \mathcal{S}_o^M(T) \models \mathbf{child}(a,b), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{desc}}(a,b) & \iff & \mathcal{S}_o^M(T) \models \mathbf{desc}(a,b), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{root}}(a) & \iff & \mathcal{S}_o^M(T) \models \mathbf{root}(a), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{leaf}}(a) & \iff & \mathcal{S}_o^M(T) \models \mathbf{leaf}(a), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{ls}}(a) & \iff & \mathcal{S}_o^M(T) \models \mathbf{ls}(a).
\end{array}$$

Proof. Obviously, we can choose

$$\begin{array}{rcl}
\varphi_{\mathbf{root}}(x) & := & \neg\, \exists y \,\big(\, \mathbf{fc}(y,x) \,\vee\, \mathbf{ns}(y,x) \,\big), \\
\varphi_{\mathbf{leaf}}(x) & := & \neg\, \exists y \;\mathbf{fc}(x,y), \\
\varphi_{\mathbf{ls}}(x) & := & \neg\, \exists y \;\mathbf{ns}(x,y).
\end{array}$$

In order to construct $\varphi_{\mathbf{child}}(x,y)$ and $\varphi_{\mathbf{desc}}(x,y)$, we use the following auxiliary formulas: Let $\varrho(x,y)$ be an arbitrary formula, let X be a set variable, and let

$$cl_{\varrho(x,y)}(X) \;:=\; \forall x\, \forall y \,\Big(\, \big(\, X(x) \wedge \varrho(x,y) \,\big) \,\rightarrow\, X(y) \,\Big).$$

Clearly, this formula holds for a set X if and only if X is closed under "ϱ-successors".

In particular, the formula

$$\varphi_{\mathbf{ns}^*}(x,y) \;:=\; \forall X \Big(\big(X(x) \wedge cl_{\mathbf{ns}(x,y)}(X) \big) \rightarrow X(y) \Big)$$

expresses that y is either equal to x or it is a sibling of x which is bigger than x with respect to the linear order of all children of x and y's common parent. Consequently, we can choose

$$\varphi_{\mathbf{child}}(x,y) \;:=\; \exists x' \big(\mathbf{fc}(x,x') \wedge \varphi_{\mathbf{ns}^*}(x',y) \big).$$

Since the **desc**-relation is the transitive (and non-reflexive) closure of the **child**-relation, we can choose

$$\varphi_{\mathbf{desc}}(x,y) \;:=\; x \neq y \wedge \forall X \Big(\big(X(x) \wedge cl_{\varphi_{\mathbf{child}}(x,y)}(X) \big) \rightarrow X(y) \Big). \qquad \square$$

In combination with Theorem 3.4 and Proposition 3.3, this leads to:

Corollary 3.6. *Let M be the set $\{\,\mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf}, \mathbf{ls}\,\}$. The following languages can express exactly the same unary queries on unranked ordered Σ-labeled trees:*

$$\mathit{mDatalog}(\tau_{GK,\Sigma}),\ \ \mathit{mDatalog}(\tau^M_{o,\Sigma}),\ \ \mathrm{MSO}(\tau^M_{o,\Sigma}),\ \ \mathrm{MSO}(\tau_{GK,\Sigma}),\ \ \mathrm{MSO}(\tau_{o,\Sigma}).$$

Furthermore, there exits an algorithm which translates a given unary query on unranked ordered Σ-labeled trees formulated in one of these languages into an equivalent queries formulated in any of the other languages.

In particular, adding the **child** *and* **desc** *relations to τ_{GK} neither increases the expressive power of monadic datalog nor the expressive power of MSO on unranked ordered Σ-labeled trees.*

Proof. Since $\tau_{GK} \subseteq \tau^M_o$, $\mathrm{mDatalog}(\tau_{GK,\Sigma})$ is at most as expressive as $\mathrm{mDatalog}(\tau^M_{o,\Sigma})$ which, by Proposition 3.3, is at most as expressive as $\mathrm{MSO}(\tau^M_{o,\Sigma})$.

By Proposition 3.5, $\mathrm{MSO}(\tau^M_{o,\Sigma})$ is as expressive on ordered Σ-labeled trees as $\mathrm{MSO}(\tau_{o,\Sigma})$ and $\mathrm{MSO}(\tau_{GK,\Sigma})$ which, by Theorem 3.4, is as expressive on unranked ordered Σ-labeled trees as $\mathrm{mDatalog}(\tau_{GK,\Sigma})$.

Furthermore, by Proposition 3.3, Proposition 3.5, and Theorem 3.4, the translation from one language to another is constructive. $\square$

Next, we note that omitting any of the unary relations **root**, **leaf**, or **ls** decreases the expressive power of monadic datalog on unranked ordered Σ-labeled trees.

Proposition 3.7. *Let M be the set $\{\,\mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf}, \mathbf{ls}\,\}$.*
For any relation $\mathbf{rel} \in \{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$, the unary query $q_{\mathbf{rel}}$ with

$$q_{\mathbf{rel}}(T) = \{v \in V^T \,:\, \mathcal{S}^M_o(T) \models \mathbf{rel}(v)\}$$

for every unranked ordered Σ-labeled tree T can be expressed in $\mathit{mDatalog}(\{\mathbf{rel}\})$, but not in $\mathit{mDatalog}(\tau^M_{o,\Sigma} \setminus \{\mathbf{rel}\})$.

Proof. It is obvious that the query $q_{\mathbf{rel}}$ can be expressed in mDatalog($\{\mathbf{rel}\}$).

Let $N \subseteq \{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$ be such that $\tau_o^N = \tau_o^M \setminus \{\mathbf{rel}\}$. Assume, for contradiction, that $q_{\mathbf{rel}}$ is expressed by an mDatalog(τ_o^N)-query $Q = (\mathcal{P}, P)$.

First, consider the case where $\mathbf{rel} = \mathbf{root}$. Let T_0 be the tree consisting of a single node v labeled $\alpha \in \Sigma$, and let T_1 be the tree consisting of two nodes u, v, both labeled α, such that v is the unique child of u. Since $\tau_o^N = \tau_o^M \setminus \{\mathbf{root}\}$, we have

$$\begin{aligned} atoms\big(\mathcal{S}_o^N(T_0)\big) &= \{\, \mathbf{label}_\alpha(v),\ \mathbf{leaf}(v)\,\}, \quad \text{and} \\ atoms\big(\mathcal{S}_o^N(T_1)\big) &= atoms\big(\mathcal{S}_o^N(T_0)\big) \cup \left\{ \begin{array}{l} \mathbf{label}_\alpha(u),\ \mathbf{fc}(u,v), \\ \mathbf{ls}(v),\ \mathbf{child}(u,v), \\ \mathbf{desc}(u,v) \end{array} \right\}. \end{aligned}$$

It holds that $atoms(\mathcal{S}_o^N(T_0)) \subseteq atoms(\mathcal{S}_o^N(T_1))$ and thus, due to the monotonicity stated in Remark 2.7, we have $Q(\mathcal{S}_o^N(T_0)) \subseteq Q(\mathcal{S}_o^N(T_1))$ for all queries Q. This contradicts the fact that $v \in q_{\mathbf{root}}(T_0) = Q(\mathcal{S}_o^N(T_0))$ but $v \notin q_{\mathbf{root}}(T_1) = Q(\mathcal{S}_o^N(T_1))$.

Next, consider the case where $\mathbf{rel} = \mathbf{leaf}$, and let T_0 again be the tree consisting of a single node v labeled $\alpha \in \Sigma$. Let T_1' be the tree consisting of two nodes v and w, both labeled α, such that w is the unique child of v. Since $\tau_o^N = \tau_o^M \setminus \{\mathbf{leaf}\}$, it is straightforward to see that $atoms(\mathcal{S}_o^N(T_0)) \subseteq atoms(\mathcal{S}_o^N(T_1'))$. By monotonicity, we have that $Q(\mathcal{S}_o^N(T_0)) \subseteq Q(\mathcal{S}_o^N(T_1'))$ contradicting the fact that $v \in q_{\mathbf{leaf}}(T_0) = Q(\mathcal{S}_o^N(T_0))$ but $v \notin q_{\mathbf{leaf}}(T_1') = Q(\mathcal{S}_o^N(T_1'))$.

Finally, consider the case where $\mathbf{rel} = \mathbf{ls}$. Let T_1 again be the tree consisting of two nodes u, v, both labeled α, such that v is the unique child of u. Let T_2 be the tree consisting of three nodes u, v, w, all labeled α, such that v and w are the first and the second child of u. Since $\tau_o^N = \tau_o^M \setminus \{\mathbf{ls}\}$, it is straightforward to see that $atoms(\mathcal{S}_o^N(T_1)) \subseteq atoms(\mathcal{S}_o^N(T_2))$. By monotonicity, we have $Q(\mathcal{S}_o^N(T_1)) \subseteq Q(\mathcal{S}_o^N(T_2))$, contradicting the fact that $v \in q_{\mathbf{ls}}(T_1) = Q(\mathcal{S}_o^N(T_1))$ but $v \notin q_{\mathbf{ls}}(T_2) = Q(\mathcal{S}_o^N(T_2))$. □

3.2.2 Expressive Power on Ranked Ordered Trees

In this subsection we will consider the expressive power on ranked ordered Σ-labeled trees and may surprisingly point out several differences.

At first, we will see that we can verify in monadic second order logic whether an unranked tree respects a given ranked alphabet over the same symbol set.
By the next proposition, we obtain for the **root** relation a result similar to Proposition 3.7 for unranked ordered trees.

Proposition 3.8. *Let M be the set $\{\mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf}, \mathbf{ls}\}$.*
The unary query $Q_{\mathbf{root}}$ with

$$Q_{\mathbf{root}}(T) = \{v \in V^T \,:\, \mathcal{S}_o^M(T) \models \mathbf{root}(v)\}$$

for every ranked ordered Σ-labeled tree T can be expressed in mDatalog($\{\mathbf{root}\}$), but not in mDatalog($\tau_{o,\Sigma}^M \setminus \{\mathbf{root}\}$).

Proof. It is obvious that the query $Q_{\mathbf{root}}$ can be expressed in mDatalog($\{\mathbf{root}\}$).

Let $\Sigma = (\sigma, ar)$ the ranked alphabet where $\sigma = \{a_0, a_1\}$ and where ar is defined as follows:

$$ar(a_0) = 0 \qquad \text{and} \qquad ar(a_1) = 1.$$

Let T_0 be the tree consisting of a single node v labeled by a_0, and let T_1 be the tree consisting of two nodes u and v such that the a_0-labeled node v is the unique child of u which is labeled by a_1. For $N := M \setminus \{\mathbf{root}\}$, we have

$$atoms\big(\mathcal{S}_o^N(T_0)\big) \;= \{\; \mathbf{label}_{a_0}(v),\; \mathbf{leaf}(v) \;\}$$

and

$$atoms\big(\mathcal{S}_o^N(T_1)\big) \;= atoms\big(\mathcal{S}_o^N(T_0)\big) \;\cup\; \left\{ \begin{array}{l} \mathbf{label}_{a_1}(u),\; \mathbf{fc}(u,v), \\ \mathbf{ls}(v),\; \mathbf{child}(u,v), \\ \mathbf{desc}(u,v) \end{array} \right\}.$$

We have that $atoms(\mathcal{S}_o^N(T_0)) \subseteq atoms(\mathcal{S}_o^N(T_1))$ and thus, due to the monotonicity stated in Remark 2.7, we have $Q(\mathcal{S}_o^N(T_0)) \subseteq Q(\mathcal{S}_o^N(T_1))$ for all queries Q. This contradicts the fact that $v \in q_{\mathbf{root}}(T_0) = Q(\mathcal{S}_o^N(T_0))$ but $v \notin q_{\mathbf{root}}(T_1) = Q(\mathcal{S}_o^N(T_1))$. □

Note that in the case of **leaf** and **ls** a construction similar to the proof of Proposition 3.7 is not possible since adding a new child node would contradict the rank labeling of the parent nodes. By using the meta information in the alphabet, we obtain the surprising fact that on ranked trees even the unary relations **leaf** and **ls** are mDatalog($\tau_{o,\Sigma}$)-definable.

Proposition 3.9. *Let M be the set* $\{\,\mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf}, \mathbf{ls}\,\}$.
For any relation $\mathbf{rel} \in \{\mathbf{leaf}, \mathbf{ls}\}$ *and every ranked alphabet Σ, there exists a unary query $Q_{\mathbf{rel}}$ in mDatalog($\tau_{o,\Sigma}$) such that*

$$Q_{\mathbf{rel}}(T) = \{v \in V^T \,:\, \mathcal{S}_o^M(T) \models \mathbf{rel}(v)\}$$

for every ranked ordered Σ-labeled tree T.

Furthermore the query $Q_{\mathbf{rel}}$ can be constructed in time linear in the size of Σ.

Proof. First, we consider $\mathbf{rel} = \mathbf{leaf}$. Since Σ is a ranked alphabet, we know that a node v of a ranked ordered Σ-labeled tree T is labeled by a symbol of Σ_0 if and only if v is a leaf. Therefore, we have $Q_{\mathbf{leaf}} = (\mathcal{P}, \mathbf{leaf})$ where $\mathcal{P}$ is the following set of rules in mDatalog($\tau_{o,\Sigma}$):

$$\big\{\; \mathbf{leaf}(x) \leftarrow \mathbf{label}_\alpha(x) \,|\, \alpha \in \Sigma_0 \;\big\}.$$

Now, we consider the case $\mathbf{rel} = \mathbf{ls}$. Since Σ is a ranked alphabet, we know that a node v of a ranked ordered Σ-labeled tree T is labeled by a symbol of Σ_i, for every $0 \leqslant i \leqslant rk_{\max}(\Sigma)$ if and only if v has i children. We construct the program $\mathcal{P}$ for the query $Q_{\mathbf{ls}} = (\mathcal{P}, \mathbf{ls})$ progressively over the sets Σ_i of the partition of Σ.

For every symbol $\alpha \in \Sigma_1$ we add the rule

$$\mathbf{ls}(x_1) \;\leftarrow\; \mathbf{label}_\alpha(y), \mathbf{fc}(y, x_1)$$

to $\mathcal{P}$. For every symbol $\alpha \in \Sigma_2$ we add the rule

$$\mathbf{ls}(x_2) \;\leftarrow\; \mathbf{label}_\alpha(y), \mathbf{fc}(y, x_1), \mathbf{ns}(x_1, x_2),$$

and we proceed as follows:

For $0 \leqslant i \leqslant rk_{\max}(\Sigma)$ and for every symbol $\alpha \in \Sigma_i$, we add

$$\mathbf{ls}(x_i) \leftarrow \mathbf{label}_\alpha(y), \mathbf{fc}(y, x_1), \mathbf{ns}(x_1, x_2), \mathbf{ns}(x_2, x_3), \ldots, \mathbf{ns}(x_{i-1}, x_i),$$

It is easy to verify that for every ranked ordered Σ-labeled tree T, the query $Q_{\mathbf{ls}}$ yields exactly the set of the last sibling nodes on T.

This approach is constructive and gives us a linear time algorithm for every query $Q_{\mathbf{rel}}$. □

The final remark for this section is the following corollary.

Corollary 3.10. *Let M be the set $\{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$ and let Σ be a ranked alphabet. The following languages can express exactly the same unary queries on ranked ordered Σ-labeled trees:*

mDatalog$(\tau_{o,\Sigma}^{\mathbf{root}})$, *mDatalog*$(\tau_{GK,\Sigma})$, *mDatalog*$(\tau_{o,\Sigma}^{M})$, MSO$(\tau_{o,\Sigma}^{M})$, MSO$(\tau_{GK,\Sigma})$, MSO$(\tau_{o,\Sigma})$.

Furthermore, there is an algorithm which translates a given unary query on ranked ordered Σ-labeled trees formulated in one of these languages into equivalent queries formulated in any of the other languages on ranked ordered Σ-labeled trees.

In particular, adding the **leaf**, **ls**, **child**, *and* **desc** *relations to $\tau_o^{\mathbf{root}}$ does neither increase the expressive power of monadic datalog nor the expressive power of MSO on ranked ordered Σ-labeled trees.*

Proof. Since $\tau_o^{\mathbf{root}} \subseteq \tau_{GK}$, mDatalog$(\tau_{o,\Sigma}^{\mathbf{root}})$ is at most as expressive as mDatalog(τ_{GK}). Further, $\tau_{GK} \subseteq \tau_o^M$ and therefore mDatalog$(\tau_{GK,\Sigma})$ is at most as expressive as mDatalog$(\tau_{o,\Sigma}^{M})$ which, by Proposition 3.3, is at most as expressive as MSO$(\tau_{o,\Sigma}^{M})$.

By Proposition 3.5, MSO$(\tau_{o,\Sigma}^{M})$ is as expressive on Σ-labeled ordered trees as MSO$(\tau_{o,\Sigma})$ and MSO$(\tau_{GK,\Sigma})$ which, by Theorem 3.4, is as expressive on ranked ordered Σ-labeled trees as mDatalog$(\tau_{GK,\Sigma})$. Finally by Proposition 3.9, mDatalog$(\tau_{GK,\Sigma})$ is as expressive on ranked ordered Σ-labeled trees as mDatalog$(\tau_{o,\Sigma}^{\mathbf{root}})$.

Furthermore, by the used propositions and theorems the translation from one language to another is constructive. □

3.3 Expressive Power of mDatalog on Unordered Trees

Let τ be one of the schemas introduced in Section 2.4.1, i.e., $\tau = \tau_{u,\Sigma}^{M}$ for some $M \subseteq \{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$. We say that a unary query q on Σ-labeled unordered trees is *definable* in mDatalog(τ) (or, *mDatalog(τ)-definable*) if and only if there exists a unary monadic datalog query Q of schema τ such that for every unordered Σ-labeled tree T we have $q(T) = Q(\mathcal{S}_u^M(T))$. Similarly, for any subset L of MSO, q is called *$L(\tau)$-definable* if and only if there is an $L(\tau)$-formula $\varphi(x)$ such that for every unordered Σ-labeled tree T we have $q(T) = \varphi(\mathcal{S}_u^M(T))$.

Often, we will simply write $Q(T)$ instead of $Q(\mathcal{S}_u^M(T))$, and $\varphi(T)$ instead of $\varphi(\mathcal{S}_u^M(T))$.

Proposition 3.3 implies that unary queries on unordered Σ-labeled trees which are definable in mDatalog(τ) are also definable in MSO(τ). It is straightforward to see that MSO(τ_u) can express all the relations present in $\tau_u^{\{\mathbf{desc},\mathbf{root},\mathbf{leaf}\}}$:

Proposition 3.11 (Folklore)**.** *Let M be the set* $\{\,\mathbf{desc},\ \mathbf{root},\ \mathbf{leaf}\,\}$*. There are* $\mathrm{MSO}(\tau_{u,\Sigma})$*-formulas*

$$\varphi_{\mathbf{desc}}(x,y),\quad \varphi_{\mathbf{root}}(x),\quad \varphi_{\mathbf{leaf}}(x)$$

such that for every unordered Σ-labeled tree T and all nodes a, b of T we have

$$\begin{array}{lcl} \mathcal{S}_u(T) \models \varphi_{\mathbf{desc}}(a,b) & \iff & \mathcal{S}_u^M(T) \models \mathbf{desc}(a,b), \\ \mathcal{S}_u(T) \models \varphi_{\mathbf{root}}(a) & \iff & \mathcal{S}_u^M(T) \models \mathbf{root}(a), \\ \mathcal{S}_u(T) \models \varphi_{\mathbf{leaf}}(a) & \iff & \mathcal{S}_u^M(T) \models \mathbf{leaf}(a). \end{array}$$

Proof. Obviously, we can choose

$$\begin{array}{rcl} \varphi_{\mathbf{root}}(x) & := & \neg\,\exists y\ \mathbf{child}(y,x), \\ \varphi_{\mathbf{leaf}}(x) & := & \neg\,\exists y\ \mathbf{child}(x,y). \end{array}$$

In order to construct $\varphi_{\mathbf{desc}}(x,y)$, we use the following auxiliary formula: Let $\varrho(x,y)$ be an arbitrary formula, let X be a set variable, and let

$$cl_{\varrho(x,y)}(X) \ := \ \forall x\,\forall y\,\Big(\,\big(\,X(x) \wedge \varrho(x,y)\,\big)\ \rightarrow\ X(y)\,\Big).$$

Clearly, this formula holds for a set X if and only if X is closed under "ϱ-successors".

In particular, the formula

$$\varphi_{\mathbf{child}^*}(x,y) \ := \ \forall X\,\Big(\,\big(\,X(x)\ \wedge\ cl_{\mathbf{child}(x,y)}(X)\,\big)\ \rightarrow\ X(y)\,\Big)$$

expresses that y is either equal to x or it is a descendant of x. Thus, we can choose

$$\varphi_{\mathbf{desc}}(x,y) \ := \ x \neq y\ \wedge\ \varphi_{\mathbf{child}^*}(x,y). \qquad \square$$

3.3.1 Expressive Power on Unranked Unordered Trees

Unlike to the case of unranked ordered trees, $\mathrm{mDatalog}(\tau_{u,\Sigma}^{\{\mathbf{desc},\mathbf{root},\mathbf{leaf}\}})$ cannot express all unary queries expressible in $\mathrm{MSO}(\tau_{u,\Sigma})$ on unranked unordered trees, as the following observation shows.

Proposition 3.12. *Let M be the set* $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$*. The unary query q_{two} with*

$$q_{two}(T) \ = \ \{v \in V^T :\ \textit{exactly two children of } v \textit{ are labeled by } \alpha\}$$

for every unranked unordered Σ-labeled tree T is expressible in $\mathrm{MSO}(\tau_{u,\Sigma})$*, but not in* $\mathit{mDatalog}(\tau_{u,\Sigma}^M)$*.*

Proof. It is obvious that the query q_{two} is defined by the $\mathrm{MSO}(\tau_{u,\Sigma})$-formula $\psi(x) :=$

$$\begin{array}{c} \exists y_1\,\exists y_2\,\Big(\mathbf{child}(x,y_1)\ \wedge\ \mathbf{child}(x,y_2)\ \wedge\ \mathbf{label}_\alpha(y_1)\ \wedge\ \mathbf{label}_\alpha(y_2)\ \wedge\ y_1 \neq y_2\ \wedge \\ \forall z\,\big(\,(\,\mathbf{child}(x,z)\ \wedge\ \mathbf{label}_\alpha(z)\,)\ \rightarrow\ (\,z = y_1\ \vee\ z = y_2\,)\,\big)\Big). \end{array}$$

For contradiction, assume that q_{two} is expressed by an $\mathrm{mDatalog}(\tau_{u,\Sigma}^M)$-query $Q = (\mathcal{P}, P)$ on unranked unordered Σ-labeled trees. Let T_2 be the unranked unordered Σ-labeled

tree consisting of three nodes u, v_1, v_2, all labeled α, such that v_1 and v_2 are children of u. Furthermore, let T_3 be the tree consisting of four nodes u, v_1, v_2, v_3, all labeled α, such that v_1, v_2, v_3 are children of u. Since

$$\tau_{u,\Sigma} \;=\; \{\mathbf{label}_\alpha \,:\, \alpha \in \Sigma\} \;\cup\; \{\,\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}\,\},$$

it is straightforward to see that $\mathit{atoms}(\mathcal{S}_u^M(T_2)) \subseteq \mathit{atoms}(\mathcal{S}_u^M(T_3))$. Thus, due to the monotonicity stated in Remark 2.7, we have $Q(\mathcal{S}_u^M(T_2)) \subseteq Q(\mathcal{S}_u^M(T_3))$ for all queries Q. This contradicts the fact that $u \in q_{two}(T_2) = Q(\mathcal{S}_u^M(T_2))$ but $u \notin q_{two}(T_3) = Q(\mathcal{S}_u^M(T_3))$. □

Next, we note that omitting any of the relations **root** or **leaf** further decreases the expressive power of monadic datalog on unranked unordered Σ-labeled trees.

Proposition 3.13. *Let M be the set $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$.*
For any relation $\mathbf{rel} \in \{\mathbf{root}, \mathbf{leaf}\}$, the query $q_{\mathbf{rel}}$ with

$$q_{\mathbf{rel}}(T) = \{v \in V^T \,:\, \mathcal{S}_u^M(T) \models \mathbf{rel}(v)\}$$

every unranked unordered Σ-labeled tree T can be expressed in mDatalog($\{\mathbf{rel}\}$), but not in mDatalog($\tau_{u,\Sigma}^M \setminus \mathbf{rel}$).

Proof. The proof of is analogous to the corresponding parts of the proof of Proposition 3.7. □

In summary, we immediately obtain the following:

Corollary 3.14. *Let M be the set $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$.*

(a) $\mathrm{MSO}(\tau_{u,\Sigma})$ *can express exactly the same unary queries on unranked unordered Σ-labeled trees as* $\mathrm{MSO}(\tau_{u,\Sigma}^M)$. *There is a polynomial time algorithm which translates a given unary* $\mathrm{MSO}(\tau_{u,\Sigma}^M)$*-query on Σ-labeled unordered trees into an equivalent* $\mathrm{MSO}(\tau_{u,\Sigma})$*-query.*

Furthermore, both languages are capable of expressing strictly more unary queries on unranked unordered Σ-labeled trees than mDatalog($\tau_{u,\Sigma}^M$).

(b) Omitting any of the relations **root** *or* **leaf** *strictly decreases the expressive power of unary mDatalog($\tau_{u,\Sigma}^M$)-queries on unranked unordered Σ-labeled trees.* ⌟

3.3.2 Expressive Power on Ranked Unordered Trees

In the following we are going to establish a result corresponding to Proposition 3.12 for ranked unordered trees, but we have to use a different approach in the proof.

Proposition 3.15. *Let M be the set $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$ and let Σ be a ranked alphabet. The unary query q_{two} with*

$$q_{two}(T) \quad = \quad \{v \in V^T \,:\, \textit{exactly two children of } v \textit{ are labeled by } \alpha\}$$

for every ranked unordered Σ-labeled tree T is expressible in $\mathrm{MSO}(\tau_{u,\Sigma})$, *but not in mDatalog($\tau_{u,\Sigma}^M$).*

Proof. It is obvious that the query q_{two} is defined by the MSO($\tau_{u,\Sigma}$)-formula $\psi(x)$ from the proof of Proposition 3.12.
For the mDatalog part of the proof, we cannot use the monotonicity of datalog since adding a new child to node v within a ranked tree T to obtain T' requires a new label of v since the number of children changes. This implies $atoms(\mathcal{S}_u^M(T)) \not\subseteq atoms(\mathcal{S}_u^M(T'))$.

For contradiction, assume that q_{two} is expressed by an mDatalog($\tau_{u,\Sigma}^M$)-query $Q = (\mathcal{P}, P)$ on ranked unordered Σ-labeled trees. We will conclude the proof by using Lemma 2.8 stating that datalog queries are preserved under homomorphisms.

Let $\Sigma = (\sigma, ar)$ be a ranked alphabet where $\sigma = \{\alpha, \beta, \gamma\}$ and ar is defined as follows:

$$ar(\alpha) = 0, \quad ar(\beta) = 0, \quad \text{and} \quad ar(\gamma) = 2.$$

Let T_1 be the ranked unordered Σ-labeled tree consisting of three nodes a, a_1, a_2 such that a_1 and a_2 are children of a. The nodes a_1 and a_2 of T_1 are each labeled by α and γ is the label of a. Furthermore, let T_2 be the ranked tree consisting of three nodes b (labeled by γ), b_1 (labeled by α), and b_2 (labeled by β) such that b_1 and b_2 are children of b. Let $\mathcal{A} := \mathcal{S}_u^M(T_1)$ and $\mathcal{B} := \mathcal{S}_u^M(T_2)$.

Consider the mapping $h : \mathcal{A} \to \mathcal{B}$ with $h(a) = b$ and $h(a_1) = h(a_2) = b_1$. It is not difficult to see that h is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$ since

- $\mathbf{label}_\alpha^\mathcal{A} = \{a_1, a_2\}$ and $h(\{a_1, a_2\}) = \{b_1\} = \mathbf{label}_\alpha^\mathcal{B}$
- $\mathbf{label}_\beta^\mathcal{A} = \varnothing$ and $\mathbf{label}_\beta^\mathcal{B} = \{b_2\}$
- $\mathbf{label}_\gamma^\mathcal{A} = \{a\}$ and $h(\{a\}) = \{b\} = \mathbf{label}_\gamma^\mathcal{B}$
- $\mathbf{child}^\mathcal{A} = \{(a, a_1), (a, a_2)\}$
  and $h(\{(a, a_1), (a, a_2)\}) = \{(b, b_1)\} \subseteq \{(b, b_1), (b, b_2)\} = \mathbf{child}^\mathcal{B}$
- $\mathbf{desc}^\mathcal{A} = \mathbf{child}^\mathcal{A}$ and $h(\mathbf{child}^\mathcal{A}) \subseteq \mathbf{child}^\mathcal{B} = \mathbf{desc}^\mathcal{B}$
- $\mathbf{root}^\mathcal{A} = \{a\}$ and $h(\{a\}) = \{b\} = \mathbf{root}^\mathcal{B}$
- $\mathbf{leaf}^\mathcal{A} = \{a_1, a_2\}$ and $h(\{a_1, a_2\}) = \{b_1\} \subseteq \{b_1, b_2\} = \mathbf{leaf}^\mathcal{B}$

From Lemma 2.8 we obtain that $h(Q(\mathcal{A})) \subseteq Q(\mathcal{B})$. This contradicts the fact that $a \in q_{two}(T_1) = Q(\mathcal{A})$, but $h(a) = b \notin q_{two}(T_2) = Q(\mathcal{B})$. □

Next, we note that omitting the relation **root** decreases the expressive power of monadic datalog on ranked unordered Σ-labeled trees.

Proposition 3.16. *Let M be the set $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$.*
The unary query $Q_\mathbf{root}$ with

$$Q_\mathbf{root}(T) = \{v \in V^T : \mathcal{S}_u^M(T) \models \mathbf{root}(v)\}$$

for every ranked unordered Σ-labeled tree T can be expressed in mDatalog($\{\mathbf{root}\}$), but not in mDatalog($\tau_{u,\Sigma}^M \setminus \{\mathbf{root}\}$).

Proof. The proof of is analogous to the proof of Proposition 3.8 for the ranked ordered case. □

As done in Proposition 3.9 on ranked ordered trees, the **leaf** predicate can also be expressed in mDatalog($\tau_{u,\Sigma}$) on ranked unordered Σ-labeled trees.

Proposition 3.17. *Let M be the set* $\{$ **child**, **desc**, **root**, **leaf** $\}$.
For every ranked alphabet Σ, there is a unary query $Q_{\mathbf{leaf}}$ in mDatalog($\tau_{u,\Sigma}$) such that

$$Q_{\mathbf{leaf}}(T) = \{v \in V^T \,:\, \mathcal{S}_u^M(T) \models \mathbf{leaf}(v)\}$$

for every ranked unordered Σ-labeled tree T.

Furthermore the query $Q_{\mathbf{leaf}}$ can be constructed in time linear in the size of Σ.

Proof. Since Σ is a ranked alphabet, we know that a node v of a ranked unordered Σ-labeled tree T is labeled by a symbol of Σ_0 if and only if v is a leaf. Therefore, we have $Q_{\mathbf{leaf}} = (\mathcal{P}, \mathbf{leaf})$ where $\mathcal{P}$ is the following set of rules in mDatalog($\tau_{u,\Sigma}$):

$$\left\{\ \mathbf{leaf}(x) \leftarrow \mathbf{label}_\alpha(x) \,|\, \alpha \in \Sigma_0\ \right\}.$$

Obviously, there exists an algorithm that constructs $Q_{\mathbf{leaf}}$ within time linear in the size of the given alphabet Σ. □

In summary, we immediately obtain the following:

Corollary 3.18. *Let M be the set* $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$.

(a) MSO($\tau_{u,\Sigma}$) *can express exactly the same unary queries on ranked unordered Σ-labeled trees as* MSO($\tau_{u,\Sigma}^M$). *There is a polynomial time algorithm which translates a given unary* MSO($\tau_{u,\Sigma}^M$)*-query on Σ-labeled unordered trees into an equivalent* MSO($\tau_{u,\Sigma}$)*-query.*

Furthermore, both languages are capable of expressing strictly more unary queries on ranked unordered Σ-labeled trees than mDatalog($\tau_{u,\Sigma}^M$).

(b) Omitting the relation **root** *strictly decreases the expressive power of unary mDatalog($\tau_{u,\Sigma}^M$)-queries on ranked unordered Σ-labeled trees.* ⌟

3.4 Decidability Results

In this section, we obtain decidability for the problems introduced in Section 2.7. For ordered Σ-labeled trees, the following is known:

Theorem 3.19 (Gottlob and Koch [GK04])**.**
The QCP for unary mDatalog(τ_{GK}) on unranked ordered labeled trees is decidable and Exptime*-hard.* ⌟

Using Corollary 3.6 and the fact that $\tau_{GK,\Sigma} \subseteq \tau_{o,\Sigma}^{\{\mathbf{child},\mathbf{desc},\mathbf{root},\mathbf{leaf},\mathbf{ls}\}}$, this immediately leads to:

Theorem 3.20. *Let M be the set* $\{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$.

The QCP for unary mDatalog(τ_o^M) on unranked ordered labeled trees is decidable and Exptime*-hard.* ⌟

Next, we prove the decidability for ranked ordered Σ-labeled trees.

Proposition 3.21. *Let M be the set $\{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$.*

The QCP for unary mDatalog(τ_o^M) on ranked ordered labeled trees is decidable.

Proof. Let $\Sigma = (\sigma, ar)$ be the ranked alphabet and let $Q_1 = (\mathcal{P}_1, P_1)$ and $Q_2 = (\mathcal{P}_2, P_2)$ be two mDatalog($\tau_{o,\Sigma}^M$)-queries. We want to decide whether it holds for every ranked ordered Σ-labeled tree T that $Q_1(T) \subseteq Q_2(T)$.

In the following Lemma 3.22 we will see that there exists an mDatalog($\tau_{o,\sigma}^M$)-query $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ such that the query predicate P_{rk} yields true for every node of an unranked ordered σ-labeled tree T if and only if T respects the ranked alphabet. This means that there is a ranked ordered Σ-labeled tree T' such that $S_o^M(T) = S_o^M(T')$. If there exists no such a ranked tree then the query predicate is false for every node of T.

Without loss of generality we have $\mathrm{idb}(Q_1) \cap \mathcal{P}_{rk} = \mathrm{idb}(Q_2) \cap \mathcal{P}_{rk} = \varnothing$. For the given query Q_1 we will now construct the query $Q_1' = (\mathcal{P}_1', P_1')$. We let

$$\mathcal{P}_1' = \mathcal{P}_1 \cup \mathcal{P}_{rk} \cup \{\ P_1' \ \leftarrow \ P_1(x), P_{rk}(x)\ \}.$$

For the given query Q_2 we analogously construct the query $Q_2' = (\mathcal{P}_2', P_2')$. We let

$$\mathcal{P}_2' = \mathcal{P}_2 \cup \mathcal{P}_{rk} \cup \{\ P_2' \ \leftarrow \ P_2(x), P_{rk}(x)\ \}.$$

By using Theorem 3.20, we decide **yes** to the $QCP(\Sigma, Q_1, Q_2)$ on ranked ordered σ-labeled trees if and only if the $QCP(\sigma, Q_1', Q_2')$ for unary mDatalog(τ_o^M) on unranked ordered σ-labeled trees is decided to **yes** since we have:

$$\begin{aligned}
& Q_1 \not\subseteq Q_2 \\
\iff\ & \text{there is a ranked ordered } \Sigma\text{-labeled tree } T \text{ and a node } v \in T \\
& \text{such that } v \in \mathcal{T}_{\mathcal{P}_1}^{\omega}(T) \text{ and } v \notin \mathcal{T}_{\mathcal{P}_2}^{\omega}(T) \\
\iff\ & \text{there is a ranked ordered } \Sigma\text{-labeled tree } T, \\
& \text{an unranked ordered } \sigma\text{-labeled tree } T', \text{ and a node } v \in T' \text{ such that} \\
& \mathcal{S}_o^M(T) = \mathcal{S}_o^M(T'),\ v \in \mathcal{T}_{\mathcal{P}_1'}^{\omega}(T'), \text{ and } v \notin \mathcal{T}_{\mathcal{P}_2'}^{\omega}(T') \\
\iff\ & Q_1' \not\subseteq Q_2'
\end{aligned}$$

□

Lemma 3.22. *Let M be a set with $\{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}\} \subseteq M \subseteq \{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{ls}\}$ and let $\Sigma = (\sigma, ar)$ be a ranked alphabet.*

There is a unary mDatalog($\tau_{o,\sigma}^M$)-query $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ such that for every unranked ordered σ-labeled tree T we have

$$\begin{array}{c} P_{rk}(v) \in \mathcal{T}_{\mathcal{P}_{rk}}^{\omega}(T) \\ \text{for every node } v \in T \end{array} \quad \iff \quad \begin{array}{c} \text{there exists a ranked ordered } \Sigma\text{-labeled tree } T' \\ \text{such that } \mathcal{S}_o^M(T) = \mathcal{S}_o^M(T'). \end{array}$$

Moreover, if such a tree T' does not exists then we have for every node v in T that $P_{rk}(v) \notin \mathcal{T}_{\mathcal{P}_{rk}}^{\omega}(T)$.

This query can be constructed in time linear in the size of the input Σ.

Proof. We want to construct a query $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ such that for every unranked ordered σ-labeled tree T and every of its nodes v we have $P_{rk}(v) \in \mathcal{T}_{\mathcal{P}_{rk}}^{\omega}(T)$ if and only

if there exists a ranked ordered Σ-labeled tree T' such that $\mathcal{S}_o^M(T) = S_o^M(T')$. This in particular, means that if such a tree T' does not exist there is no node $v \in T$ with $P_{rk}(v) \in \mathcal{T}^{\omega}_{\mathcal{P}_{rk}}(T)$.

Let $\Sigma = (\sigma, ar)$ be a ranked alphabet. The program $\mathcal{P}'_{rk}$ in $mDatalog(\tau^M_{o,\sigma})$ consists of the following rules.

For every $\alpha \in \Sigma_0$ we have the rule

$$ok(x) \leftarrow \mathbf{leaf}(x), \mathbf{label}_\alpha(x).$$

For every $\alpha \in \Sigma_1$ we have the rule

$$ok(x) \leftarrow \mathbf{label}_\alpha(x), \mathbf{fc}(x, y), ok(y), \mathbf{ls}(y).$$

For every $\alpha \in \Sigma_2$ we have the rule

$$ok(x) \leftarrow \mathbf{label}_\alpha(x), \mathbf{fc}(x, y_1), ok(y_1), \mathbf{ns}(y_1, y_2), ok(y_2), \mathbf{ls}(y_2).$$

And for every $\alpha \in \Sigma_i$ where $2 < i \leqslant rk_{\max}(\Sigma)$ we have

$$ok(x) \leftarrow \mathbf{label}_\alpha(x), \mathbf{fc}(x, y_1), ok(y_1), \mathbf{ns}(y_1, y_2), ok(y_2), \ldots, \mathbf{ns}(y_{i-1}, y_i), ok(y_i), \mathbf{ls}(y_i)$$

in $\mathcal{P}'_{rk}$ and for the root node we have

$$P_{rk}(x) \leftarrow \mathbf{root}(x), ok(x).$$

It is easy to verify by induction over the height of the unranked tree T that the Boolean query $Q = (\mathcal{P}'_{rk}, P_{rk})$ yields **yes** for an unranked ordered σ-labeled tree T if and only if there exists a ranked ordered Σ-labeled tree T' such that $\mathcal{S}_o^M(T) = S_o^M(T')$. Furthermore, we have at most $P_{rk}(v) \in \mathcal{T}^{\omega}_{\mathcal{P}'_{rk}}(T)$ if $v = root^T$.

Now we obtain the demanded query $Q = (\mathcal{P}_{rk}, P_{rk})$ by distributing the membership of the root node to P_{rk} onto the other nodes of T with

$$\mathcal{P}_{rk} = \mathcal{P}'_{rk} \cup \{\ P_{rk}(x) \leftarrow P_{rk}(y), \mathbf{child}(y, x)\ \}$$ □

To also obtain decidability for the case of unordered Σ-labeled trees, we can use the following result:

Theorem 3.23 (Seese [See91]). *The problem*

Satisfiability of MSO($\tau_{u,\Sigma}$)-sentences on unranked unordered Σ-labeled trees	
Input:	*An* MSO($\tau_{u,\Sigma}$)*-sentence* φ.
Question:	*Does there exist an unranked unordered* Σ*-labeled finite tree* T *such that* $\mathcal{S}_u(T) \models \varphi$*?*

is decidable. ⌟

Combining this with Proposition 3.3 and Proposition 3.11, we obtain:

Theorem 3.24. *Let M be the set* $\{$ **desc**, **root**, **leaf** $\}$*. The QCP for unary mDatalog(τ_u^M) on unranked unordered labeled trees is decidable.*

Proof. An algorithm for deciding the QCP for two given unary mDatalog($\tau_{u,\Sigma}^M$)-queries on unranked unordered Σ-labeled trees can proceed as follows:

On the input of two unary mDatalog($\tau_{u,\Sigma}^M$)-queries Q_1 and Q_2 the algorithm needs to decide whether we have $Q_1 \subseteq Q_2$ or not. First use the algorithm from Proposition 3.3 to construct two MSO($\tau_{u,\Sigma}^M$)-formulas $\varphi_1(x)$ and $\varphi_2(x)$ such that, for each $i \in \{1,2\}$, the formula $\varphi_i(x)$ defines the same unary query on unranked unordered Σ-labeled trees as Q_i.

Afterwards, use Proposition 3.11 to translate the MSO($\tau_{u,\Sigma}^M$)-formulas $\varphi_1(x)$ and $\varphi_2(x)$ into MSO($\tau_{u,\Sigma}$)-formulas $\psi_1(x)$ and $\psi_2(x)$ which are equivalent to $\varphi_1(x)$ and $\varphi_2(x)$ on unordered Σ-labeled trees.

Finally, let

$$\varphi \quad := \quad \exists x \, \big(\psi_1(x) \wedge \neg \psi_2(x) \big),$$

and use the algorithm provided by Theorem 3.23 to decide whether there is an unranked unordered Σ-labeled tree T such that $\mathcal{S}_u(T) \models \varphi$. We output "**no**" if this algorithm outputs "**yes**" and we output "**yes**" otherwise.

To verify that this algorithm produces the correct answer, note that for every unranked unordered Σ-labeled tree T the following is true:

$$\begin{array}{ll} & \mathcal{S}_u(T) \models \varphi \\ \iff & \text{there is a node } a \text{ of } T \text{ with } \mathcal{S}_u^M(T) \models \psi_1(a) \text{ and } \mathcal{S}_u^M(T) \not\models \psi_2(a) \\ \iff & \text{there is a node } a \text{ of } T \text{ with } a \in Q_1(\mathcal{S}_u^M(T)) \text{ and } a \notin Q_2(\mathcal{S}_u^M(T)) \\ \iff & Q_1(\mathcal{S}_u^M(T)) \not\subseteq Q_2(\mathcal{S}_u^M(T)). \end{array}$$

Thus, the MSO($\tau_{u,\Sigma}$)-sentence φ is satisfiable on unranked unordered Σ-labeled trees if and only if $Q_1 \not\subseteq Q_2$. □

This proof can easily be extended to a proof for the ranked case.

Theorem 3.25. *Let M be the set* $\{$ **desc**, **root**, **leaf** $\}$*. The QCP for unary mDatalog(τ_u^M) on ranked unordered labeled trees is decidable.*

Proof. As we will see in the following Lemma 3.26 we cannot use the approach of creating a reduction from ranked to unranked trees as we did in the case of renked ordered trees in Proposition 3.21. However, an algorithm deciding the QCP for the given ranked alphabet $\Sigma = (\sigma, ar)$ and two given unary mDatalog($\tau_{u,\sigma}^M$)-queries on ranked unordered Σ-labeled trees can proceed as follows:

Upon input of two unary mDatalog($\tau_{u,\sigma}^M$)-queries Q_1 and Q_2 it is to decide whether we have $Q_1 \subseteq Q_2$ or not. First use the algorithm from Proposition 3.3 to construct two MSO($\tau_{u,\sigma}^M$)-formulas $\varphi_1(x)$ and $\varphi_2(x)$ such that, for each $i \in \{1,2\}$, the formula $\varphi_i(x)$ defines the same unary query on ranked unordered Σ-labeled trees as Q_i.

Afterwards, use Proposition 3.11 to translate the MSO($\tau_{u,\sigma}^M$)-formulas $\varphi_1(x)$ and $\varphi_2(x)$ into MSO($\tau_{u,\sigma}$)-formulas $\psi_1(x)$ and $\psi_2(x)$ which are equivalent to $\varphi_1(x)$ and $\varphi_2(x)$ on ranked unordered Σ-labeled trees.

Let $\varphi_{rk(\Sigma)}$ be the MSO($\tau_{u,\sigma}$)-sentence obtained by Lemma 3.28 such that for every unranked unordered σ-labeled tree T we have

$$S_u(T) \models \varphi_{rk(\Sigma)} \iff \text{Every node labeled by } \alpha \in \sigma \text{ has exactly } ar(\alpha) \text{ children.}$$

Now finally, let

$$\varphi \quad := \quad \exists x \, \big(\psi_1(x) \wedge \neg \psi_2(x) \wedge \varphi_{rk(\Sigma)} \big),$$

and use the algorithm provided by Theorem 3.23 to decide whether there is an unranked unordered σ-labeled tree T such that $\mathcal{S}_u(T) \models \varphi$. We output "**no**" if this algorithm outputs "**yes**" and we output "**yes**" otherwise.

To verify that this algorithm produces the correct answer, note that for every unranked unordered Σ-labeled tree T the following is true:

$$\begin{array}{ll} & \mathcal{S}_u(T) \models \varphi \\ \iff & \mathcal{S}_u^M(T) \models \varphi_{rk(\Sigma)} \text{ and} \\ & \text{there is a node } a \text{ of } T \text{ with } \mathcal{S}_u^M(T) \models \psi_1(a) \text{ and } \mathcal{S}_u^M(T) \not\models \psi_2(a) \\ \iff & \text{there is a ranked unordered } \Sigma\text{-labeled tree } T' \text{ with } \mathcal{S}_u^M(T) = \mathcal{S}_u^M(T') \text{ and} \\ & \text{there is a node } a \text{ of } T' \text{ with } a \in Q_1(\mathcal{S}_u^M(T')) \text{ and } a \notin Q_2(\mathcal{S}_u^M(T')) \\ \iff & \text{there is a ranked unordered } \Sigma\text{-labeled tree } T' \text{ such that} \\ & Q_1(\mathcal{S}_u^M(T')) \not\subseteq Q_2(\mathcal{S}_u^M(T')). \end{array}$$

Thus, the MSO($\tau_{u,\Sigma}$)-sentence φ is satisfiable on unranked unordered σ-labeled trees if and only if $Q_1 \not\subseteq Q_2$ on ranked unordered Σ-labeled trees. □

Lemma 3.26. *Let M be the set $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$ and let $\Sigma = (\sigma, ar)$ be a ranked alphabet.*

There exists no unary mDatalog($\tau_{u,\sigma}^M$)-query $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ such that for every unranked unordered σ-labeled tree T we have

$$\begin{array}{c} P_{rk}(v) \in \mathcal{T}^{\omega}_{\mathcal{P}_{rk}}(T) \\ \text{for every node } v \in T \end{array} \quad \iff \quad \begin{array}{c} \text{there exists a ranked unordered } \Sigma\text{-labeled tree } T' \\ \text{such that } \mathcal{S}_u^M(T) = \mathcal{S}_u^M(T') \end{array}$$

and $P_{rk}(v) \notin \mathcal{T}^{\omega}_{\mathcal{P}_{rk}}(T)$ for every node $v \in T$ if such a tree T' does not exists.

Proof. We assume that a query as described in the lemma exists and let $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ be such a unary mDatalog($\tau_{u,\sigma}^M$)-query.

This implies that $root^T \in \mathcal{T}^{\omega}_{\mathcal{P}_{rk}}(T)$ if and only if there exists a ranked unordered Σ-labeled tree T' such that $\mathcal{S}_u^M(T) = S_u^M(T')$ which is a contradiction to Lemma 3.27 for the Boolean query $Q'_{rk} = (\mathcal{P}_{rk}, P_{rk})$. □

Lemma 3.27. *Let M be the set $\{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$ and let $\Sigma = (\sigma, ar)$ be a ranked alphabet.*

There exists no Boolean mDatalog($\tau_{u,\sigma}^M$)-query $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ such that for every unranked unordered σ-labeled tree T we have

$$Q_{rk}(T) = \mathbf{yes} \quad \iff \quad \begin{array}{c} \text{there exists a ranked unordered } \Sigma\text{-labeled tree } T' \\ \text{such that } \mathcal{S}_u^M(T) = S_u^M(T'). \end{array}$$

Proof. Let $\Sigma = (\sigma, ar)$ be a ranked alphabet where $\sigma = \{\alpha_0, \alpha_1\}$, $ar(\alpha_0) = 0$, and $ar(\alpha_1) = 1$. We assume Q_{rk} is such a query for Σ.

Let T_0 be the tree consisting of two nodes u and v such that v is the unique child of u. The node u is labeled by α_1 and v is labeled by α_0. Let T_1 be a tree consisting of the

three nodes u, v, and w, with u labeled by α_1 and both v and w labeled by α_0. And let the nodes v and w be the children of u. Now we have

$$\begin{aligned} \mathit{atoms}(S_u^M(T_0)) &= \{\, \mathbf{root}(u),\ \mathbf{label}_{\alpha_1}(u),\ \mathbf{child}(u,v),\ \mathbf{desc}(u,v),\ \mathbf{label}_{\alpha_0}(v),\ \mathbf{leaf}(v)\} \\ \mathit{atoms}(S_u^M(T_1)) &= \mathit{atoms}(S_u^M(T_0)) \cup \{\, \mathbf{child}(u,w),\ \mathbf{desc}(u,w),\ \mathbf{label}_{\alpha_0}(w),\ \mathbf{leaf}(w)\} \end{aligned}$$

It is easy to verify that there exists for T_0 a ranked unordered Σ-labeled tree T_0' with $\mathcal{S}_u^M(T_0) = S_u^M(T_0')$, but such a tree for T_1 does not exist since the root node u of T_1 is labeled by the unary symbol α_1 even though it has two children.

It holds that $\mathit{atoms}(\mathcal{S}_u^M(T_0)) \subseteq \mathit{atoms}(\mathcal{S}_u^M(T_1))$ and thus, due to the monotonicity stated in Remark 2.7, we have $Q(\mathcal{S}_u^N(T_0)) \subseteq Q(\mathcal{S}_u^N(T_1))$ for all queries Q. This contradicts the fact that $Q_{rk}(T_0)$ yields **yes** and $Q_{rk}(T_1) = \mathbf{no}$. □

Lemma 3.28. *Let $\Sigma = (\sigma, ar)$ be a ranked alphabet. Let T be an unranked unordered σ-labeled tree. There exists an* $\mathrm{MSO}(\tau_{u,\sigma})$*-sentence $\varphi_{rk(\Sigma)}$ such that*

$$S_u(T) \models \varphi_{rk(\Sigma)} \iff \textit{Every node labeled by } \alpha \in \sigma \textit{ has exactly } ar(\alpha) \textit{ children.}$$

We can furthermore construct this formula within polynomial time for any given ranked alphabet.

Proof. Let $\varphi_{\mathbf{leaf}}$ the $\mathrm{MSO}(\tau_{u,\sigma})$-formula from Proposition 3.11. With the formula

$$\varphi_0 \quad := \quad \bigwedge_{\alpha \in \Sigma_0} \forall x \big(\mathbf{label}_\alpha(x) \ \leftrightarrow\ \varphi_{\mathbf{leaf}}(x) \big)$$

we have for every unranked unordered σ-labeled tree T

$$S_u(T) \models \varphi_0 \iff \text{Every node labeled by } \alpha \in \Sigma_0 \text{ has no children.}$$

Next, we define the formulas $\varphi_1, \varphi_2, \ldots, \varphi_m$ which correspond to the sets Σ_1, Σ_2, …, Σ_m

$$\varphi_1 \quad := \quad \bigwedge_{\alpha \in \Sigma_1} \forall x \Big(\mathbf{label}_\alpha(x) \ \leftrightarrow\ \exists y_1 \big(\mathbf{child}(x, y_1) \wedge \forall y (\mathbf{child}(x, y) \rightarrow y_1 = y) \big) \Big)$$

$$\varphi_2 \quad := \quad \bigwedge_{\alpha \in \Sigma_2} \forall x \Big(\mathbf{label}_\alpha(x) \ \leftrightarrow\ \exists y_1 \exists y_2 \big(\mathbf{child}(x, y_1) \wedge \mathbf{child}(x, y_2) \wedge \neg y_1 = y_2 \wedge \forall y (\mathbf{child}(x, y) \rightarrow (y_1 = y \vee y_2 = y)) \big) \Big)$$

$$\vdots$$

$$\varphi_m \quad := \quad \bigwedge_{\alpha \in \Sigma_m} \forall x \Big(\mathbf{label}_\alpha(x) \ \leftrightarrow\ \exists y_1 \ldots \exists y_m \big(\bigwedge_{1 \leqslant i \leqslant m} \big(\mathbf{child}(x, y_i) \wedge \bigwedge_{1 \leqslant j < i} \neg y_j = y_i \big) \wedge \forall y (\mathbf{child}(x, y) \rightarrow \bigvee_{1 \leqslant i \leqslant m} y_i = y) \big) \Big)$$

and obtain

$$S_u(T) \models \varphi_i \iff \text{Every node labeled by } \alpha \in \Sigma_i \text{ has } i \text{ children}$$

for every $i \in \mathbb{N}$ with $0 \leqslant i \leqslant m$. Finally, we define

$$\varphi \quad := \quad \bigwedge_{0 \leqslant i \leqslant m} \varphi_i$$

and obtain

$$S_u(T) \models \varphi \iff \text{Every node labeled by } \alpha \in \sigma \text{ has exactly } ar(\alpha) \text{ children.}$$

Obviously, there is an algorithm that uses this construction and terminates in time polynomial in Σ. □

By definition, we have $Q_1 \equiv Q_2$ if and only if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. Thus, the decidability of the query containment problem for mDatalog stated in Theorem 3.20 and Theorem 3.24 for the unranked case as well as Proposition 3.21 and Theorem 3.25 for the ranked case immediately leads to the following.

Corollary 3.29.

(a) The equivalence problem for unary mDatalog($\tau_o^{\{\mathbf{child},\mathbf{desc},\mathbf{root},\mathbf{leaf},\mathbf{ls}\}}$) on labeled ordered trees is decidable.

(b) The equivalence problem for unary mDatalog($\tau_u^{\{\mathbf{desc},\mathbf{root},\mathbf{leaf}\}}$) on labeled unordered trees is decidable. ⌟

Finally, Corollary 3.29 together with Example 2.10 leads to the following.

Corollary 3.30.

(a) The emptiness problem for unary mDatalog($\tau_o^{\{\mathbf{child},\mathbf{desc},\mathbf{root},\mathbf{leaf},\mathbf{ls}\}}$) on labeled ordered trees is decidable.

(b) The emptiness problem for unary mDatalog($\tau_u^{\{\mathbf{desc},\mathbf{root},\mathbf{leaf}\}}$) on labeled unordered trees is decidable.

Proof. Let Q be the input query for which we want to decide whether or not it is unsatisfiable on trees. Let $Q^{\varnothing}$ be the unsatisfiable query from Example 2.10.

It is straightforward to see that $Q \equiv Q^{\varnothing}$ if, and only if, Q is unsatisfiable on trees. Thus, we can use the algorithms for deciding equivalence of queries on trees (provided by Corollary 3.29) to decide whether or not Q is satisfiable on trees. □

Chapter 4

The Other Problems Point of View:

On Hardness

By Section 3.4, we know that all the considered problems of Section 2.7 are decidable. In this chapter, as a first step to determine the complexity of these problems, we investigate their hardness. Every hardness result provides us with a lower bound.

4.1 The Hardness of the Emptiness Problem of mDatalog($\tau_{\mathbf{u}}^{\{\mathbf{root},\mathbf{leaf}\}}$) on Unranked Unordered Trees

This section's goal is to prove the following proposition.

Proposition 4.1. *The emptiness problem for Boolean mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) on unranked unordered labeled trees is* EXPTIME*-hard.*

Proof. Our proof proceeds by a reduction from the EXPTIME-complete *two person corridor tiling* problem (TPCT) [Chl86]. The task of the TPCT-problem is to decide whether the first player in the following two person corridor tiling game has a winning strategy.

There are two players: Player 1 (the *Constructor*) and Player 2 (the *Saboteur*). The game board is a corridor of a given width n and an unbounded length. There is a finite set $\mathcal{D}$ of types of *tiles* (or, *dominoes*), and from every tile type an unlimited number of tiles is available. The first row f (of width n) of tiles as well as the combination of tile types in the designated last row ℓ (of width n) are given.

The players alternately select a tile and put it into the next vacant position (row-wise from left to right); Player 1 starts at the leftmost position of the second row. Both players have to respect horizontal and vertical constraints given by two sets $H, V \subseteq \mathcal{D}^2$. A tile d chosen for the j-th column of the i-th row has to fit to its vertical neighbor d_v in the j-th column of the $(i-1)$-th row in the sense that $(d_v, d) \in V$. Furthermore, if $j \geqslant 2$, then tile d also has to fit to its horizontal neighbor d_h in the $(j-1)$-th column of the i-th row in the sense that $(d_h, d) \in H$. If a player is unable to choose a fitting tile, the game ends and Player 1 loses.

The ultimate goal of Player 1 is to produce a tiling whose last row is ℓ; in this case he wins and the game ends. Player 2 wins if either the game goes on for an infinite number

of steps, or one of the players gets stuck in a situation where he cannot find a fitting tile.

The *two person corridor tiling problem* (TPCT) is the following decision problem.

TPCT	
Input:	A tuple $I = (\mathcal{D}, H, V, n, f, \ell)$ such that $\mathcal{D}$ is a finite set, $H, V \subseteq \mathcal{D}^2$, $n \geqslant 2$, $f, \ell \in \mathcal{D}^n$.
Question:	Does Player 1 have a winning strategy in the two person corridor tiling game specified by I?

Theorem 4.2 (Chlebus [Chl86])**.** *The problem TPCT is* EXPTIME*-complete.* ⌟

Note that EXPTIME is closed under complementation. Thus, for proving Proposition 4.1 it suffices to give a polynomial-time reduction from TPCT to the complement of the emptiness problem for mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) on unranked unordered labeled trees. For a given TPCT-instance $I = (\mathcal{D}, H, V, n, f, \ell)$ we will construct a finite alphabet Σ and a Boolean mDatalog($\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}}$)-query Q, such that

$$\begin{array}{ll} & \text{Player 1 has a winning strategy in the} \\ & \text{two person corridor tiling game specified by } I \\ \iff & \text{there exists an unranked unordered } \Sigma\text{-labeled tree } T \\ & \text{such that } Q(T) = \mathbf{yes} \text{ (i.e., } Q \neq \varnothing). \end{array}$$

We will represent strategies for Player 1 by Σ-labeled trees. The query Q will describe necessary properties which are met by every tree that describes a winning strategy for Player 1.

The following representation of a winning strategy for Player 1 is basically taken from [Löd12]. We represent a strategy for Player 1 by an unranked unordered Σ-labeled tree with

$$\Sigma \;:=\; \mathcal{D} \times \{1, 2, \bot, !\}.$$

The first component of a letter $(d, i) \in \Sigma$ corresponds to the tile d that has been played, while the second component indicates whose turn it is to place the next tile (1 for Player 1, 2 for Player 2, $\bot$ if the move is not allowed because a vertical or horizontal constraint was violated, and ! if the game is over because Player 1 has won). In the following, we will say that a node is labeled d (for some $d \in \mathcal{D}$) to express that its label belongs to $\{d\} \times \{1, 2, \bot, !\}$. Accordingly, we will say that a node is labeled i (for some $i \in \{1, 2, \bot, !\}$) to express that its label belongs to $\mathcal{D} \times \{i\}$.

A finite Σ-labeled tree T is called *good* if it satisfies the following conditions (1)–(9). It is not difficult to verify that Player 1 has a winning strategy if, and only if, there exists a finite Σ-labeled tree that is good.

(1) The root is labeled by $(d, 2)$ for some $d \in \mathcal{D}$. (This indicates that at the beginning of the game, Player 1 chooses tile d, and Player 2 is next).

(2) Nodes with labels $\bot$ or ! are leaves.

(3) Nodes with labels in $\mathcal{D} \times \{1\}$ have at least one child. (Such a child describes the choice made by Player 1 in the next step).

(4) Nodes with labels in $\mathcal{D} \times \{2\}$ have at least $|\mathcal{D}|$ children – one for each tile type $d \in \mathcal{D}$. (These children represent the potential choices that Player 2 might make in the next step).

(5) There is no node labeled 1 or 2 such that all of its children are labeled by $\perp$. (I.e., the game never gets stuck).

(6) Labels from $\mathcal{D} \times \{1\}$ and $\mathcal{D} \times \{2\}$ alternate on each path from the root to a leaf. (I.e., both players alternately choose a tile).

(7) If a node x is labeled !, then the number of nodes visited by the path from the root to x is a multiple of n and the last n nodes on this path are labeled according to ℓ. (This means that the last n nodes of the path describe a row which has the desired labeling ℓ.)

(8) At each node x labeled (d, i) with $i \neq \perp$, the tile d respects the horizontal and the vertical constraints.

(9) At each node labeled $(d, \perp)$ for some $d \in \mathcal{D}$, the tile d violates the horizontal or the vertical constraints.

To be precise, the conditions (8) and (9) mean the following. We define the *depth* of a node as follows: The root has depth 1; and for each node x of depth j, all children of x are of depth $j+1$.

(a) A node x labeled with tile $d \in \mathcal{D}$ *respects the horizontal constraints* if x

- is either of depth congruent 1 modulo n (and thus corresponds to a position in the first column of a row),
- or we have $(d_h, d) \in H$, where the parent of x is labeled with tile $d_h \in \mathcal{D}$ (i.e., x corresponds to a position where tile d is chosen in some column $j \geqslant 2$, and this tile fits to its horizontal neighbor d_h in column $j-1$).

(b) A node x labeled with a tile $d \in \mathcal{D}$ *respects the vertical constraints* if x

- is either of depth $j \in \{1, \ldots, n\}$ and we have $(f_j, d) \in H$
  (i.e., x corresponds to the j-th position in the second row and fits to the j-th entry f_j of the first row f),
- or it is of depth $j \geqslant n+1$ and we have $(d_v, d) \in V$, where the ancestor of x at depth $j-n$ is labeled with tile $d_v \in \mathcal{D}$
  (i.e., x corresponds to a position where tile d is chosen in some row $i \geqslant 3$, and this tile fits to its vertical neighbor d_v in row $i-1$).

As noted above, Player 1 has a winning strategy if and only if there exists a finite Σ-labeled tree T that is good, i.e., that satisfies the conditions (1)–(9). The first idea towards completing the proof of Proposition 4.1 is to try to find a monadic Datalog query Q such that for any unranked unordered Σ-labeled tree T the following is true: T is good if, and only if, $Q(T) = \mathbf{yes}$. In fact, it is not difficult to construct for each

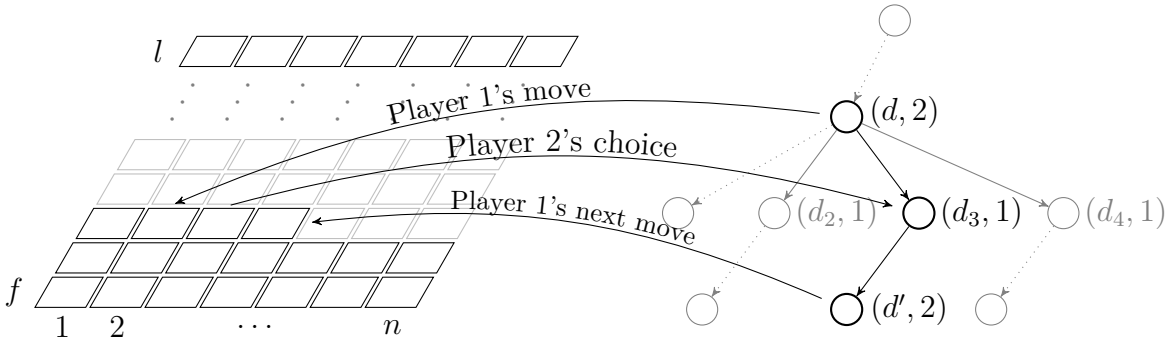

Figure 4.1: Player 1 chooses a tiling $d \in \mathcal{D}$ by following the strategy and for every answer of Player 2 there exists a child node that represents its domino type choice.

condition (c) with $c \neq 4$ and $c \neq 5$ a Boolean mDatalog$(\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}})$-query Q^c such that for any Σ-labeled tree T we have:

$$Q^c(T) = \mathbf{yes} \iff T \text{ fulfills condition } (c).$$

However, for the conditions (4) and (5), we were unable to find an according monadic datalog query which precisely characterizes all trees that fulfill these conditions.

As a remedy, we define a notion of *almost-good* trees in such a way that the following is true:

(i) Every *almost-good* tree T contains a *good* tree;[1] and every *good* tree also is *almost-good*.

(ii) We can find a Boolean mDatalog$(\tau_{u,\Sigma}^{\mathbf{root},\mathbf{leaf}})$-query Q such that for any Σ-labeled tree T the following is true: T is *almost-good* if, and only if, $Q(T) = \mathbf{yes}$.

For defining the notion of *almost-good* trees, we need the following notation. Let T be an unranked unordered Σ-labeled tree. By performing a top-down scan of T, we select a subset of the leaves as *candidates* as follows:

A *leaf* of T is a *candidate* if the path from the root to the parent of the leaf is

- labeled with symbols from $\mathcal{D} \times \{1\}$ and $\mathcal{D} \times \{2\}$ alternately, starting with a symbol from $\mathcal{D} \times \{2\}$ and
- every node on this path fulfills the vertical and the horizontal condition

and the leaf itself is labeled

- by $\perp$ and violates the vertical or the horizontal condition

or it is labeled

[1] How a good tree T will be contained in an almost-good tree T':
We have
- $V^T \subseteq V^{T'}$ • $root^T = root^{T'}$ • $E^T \subseteq E^{T'}$ • $\lambda^T = \lambda^{T'}_{|_{V^T}}$
- $\{v : v \text{ is a leaf in } T\} \subseteq \{v : v \text{ is a leaf in } T'\}$.

- by ! and
  - fulfills the vertical and the horizontal condition,
  - the number of nodes visited by the path from the root to x is a multiple of n,

  and
  - the last n nodes on this path are labeled according to ℓ.

Note that for the subtree T' of T induced by the root and the *candidate* nodes satisfies the conditions (1) - (3) and (6) - (9).

To obtain the set of *candidate* nodes, we construct the program $\mathcal{P}$ in the following way.

For accessing the parts d and i of a node-label $(d,i) \in \Sigma$, it will be convenient to include into $\mathcal{P}$ the rules

$$\mathbf{label}_d(x) \leftarrow \mathbf{label}_{(d,i)}(x) \qquad \text{and} \qquad \mathbf{label}_i(x) \leftarrow \mathbf{label}_{(d,i)}(x)$$

for every letter $(d,i) \in \Sigma$. Furthermore, for all $d, d' \in \mathcal{D}$ and $i, i' \in \{1, 2, \bot, !\}$ with $d \neq d'$ and $i \neq i'$ we add to $\mathcal{P}$ the rules

$$\mathbf{label}_{\neq d}(x) \leftarrow \mathbf{label}_{d'}(x) \qquad \text{and} \qquad \mathbf{label}_{\neq i}(x) \leftarrow \mathbf{label}_{i'}(x).$$

For every node we introduce the predicate *alt* to indicate that the path from the root to this node starts with a label from $\mathcal{D} \times \{2\}$ and continues with an alternation from $\mathcal{D} \times \{1\}$ and $\mathcal{D} \times \{2\}$ by the rules

$$alt(x) \leftarrow \mathbf{root}(x), \mathbf{label}_2(x)$$

and

$$alt(x) \leftarrow alt(y), \mathbf{child}(y,x), \mathbf{label}_i(x), \mathbf{label}_{i'}(y)$$

for $i, i' \in \{1,2\}$ with $i \neq i'$.

To count the multiples of n and to detect a violation or satisfaction of the horizontal and/or vertical condition, it will be convenient to use predicates $Column_j$ for each $j \in \{1, \ldots, n\}$, such that $Column_j(x)$ indicates that node x corresponds to a tile placed in column j of the corridor. Thus, we add to $\mathcal{P}$ the rules

$$Column_1(x) \leftarrow \mathbf{root}(x)$$
$$Column_1(x) \leftarrow \mathbf{child}(y,x), Column_n(y),$$

and for each $j \in \{2, \ldots, n\}$ the rule

$$Column_j(x) \leftarrow \mathbf{child}(y,x), Column_{j-1}(y).$$

Furthermore, for each $j \in \{1, \ldots, n-1\}$ we add to $\mathcal{P}$ the rule

$$Column_{\neq n}(x) \leftarrow Column_j(x)$$

and for each $j \in \{2, \ldots, n\}$ we add to $\mathcal{P}$ the rule

$$Column_{\neq 1}(x) \leftarrow Column_j(x).$$

To verify condition (8) it will be convenient to use predicates $Okay_H$ and $Okay_V$ such that $Okay_H(x)$ (resp., $Okay_V(x)$) indicates that node x satisfies the horizontal (resp., the vertical) constraints. Thus, for all $(d_h, d) \in H$, we add to $\mathcal{P}$ the rules

$$\begin{aligned} Okay_H(x) &\leftarrow Column_1(x) \\ Okay_H(x) &\leftarrow Column_{\neq 1}(x), \mathbf{child}(y, x), \mathbf{label}_{d_h}(y), \mathbf{label}_d(x). \end{aligned}$$

Similarly, for all $(d_v, d) \in V$, we add add to $\mathcal{P}$ the rules

$$\begin{aligned} Okay_V(x) \leftarrow \mathbf{child}(y_1, y_2), \ldots, \mathbf{child}(y_{n-1}, y_n), \mathbf{child}(y_n, x), \\ \mathbf{label}_{d_v}(y_1), \mathbf{label}_d(x). \end{aligned}$$

To detect nodes that correspond to tiles placed in the corridor's second row, i.e., tiles that must fit to the given first row $f = (f_1, \ldots, f_n) \in \mathcal{D}^n$, we furthermore add for each $j \in \{1, \ldots, n\}$ and each $d \in \mathcal{D}$ with $(f_j, d) \in V$ the rule

$$Okay_V(x_j) \leftarrow \mathbf{root}(x_1), \mathbf{child}(x_1, x_2), \ldots, \mathbf{child}(x_{n-1}, x_n), \mathbf{label}_d(x_j).$$

We combine the path conditions above in a additional predicate *path* by the following rule

$$\begin{aligned} path(x) &\leftarrow alt(x), Okay_H(x), Okay_V(x), \mathbf{root}(x) \\ path(x) &\leftarrow alt(x), Okay_H(x), Okay_V(x), \mathbf{child}(y, x), path(y) \end{aligned}$$

To detect a violation of the horizontal or vertical constraint in condition (9), it will be convenient to use predicates $Buggy_H$ and $Buggy_V$, such that $Buggy_H(x)$ (resp., $Buggy_V(x)$) indicates that node x violates the horizontal (resp., the vertical) constraints. Thus, for all $(d_h, d) \in \mathcal{D}^2 \setminus H$, we add to $\mathcal{P}$ the rule

$$Buggy_H(x) \leftarrow Column_{\neq 1}(x), \mathbf{child}(y, x), \mathbf{label}_{d_h}(y), \mathbf{label}_d(x)$$

Similarly, for all $(d_v, d) \in \mathcal{D}^2 \setminus V$, we add add to $\mathcal{P}$ the rule

$$\begin{aligned} Buggy_V(x) \leftarrow \mathbf{child}(y_1, y_2), \ldots, \mathbf{child}(y_{n-1}, y_n), \mathbf{child}(y_n, x), \\ \mathbf{label}_{d_v}(y_1), \mathbf{label}_d(x) \end{aligned}$$

To detect nodes that correspond to tiles placed in the corridor's second row, i.e., tiles that must fit to the given first row $f = (f_1, \ldots, f_n) \in \mathcal{D}^n$, we furthermore add for each $j \in \{1, \ldots, n\}$ and each $d \in \mathcal{D}$ with $(f_j, d) \notin V$, the rule

$$Buggy_V(x_j) \leftarrow \mathbf{root}(x_1), \mathbf{child}(x_1, x_2), \ldots, \mathbf{child}(x_{n-1}, x_n), \mathbf{label}_d(x_j).$$

Next, we introduce a predicate that is true for a node if any violation happened

$$Buggy(x) \leftarrow Buggy_V(x) \qquad \text{and} \qquad Buggy(x) \leftarrow Buggy_H(x)$$

and a predicate $Okay$ for a node if no violation happened.

$$Okay(x) \leftarrow Okay_V(x), Okay_H(x)$$

To verify condition (7) we add the following rule

$$\ell(x_n) \leftarrow \mathbf{label}_{\ell_1}(x_1), \mathbf{child}(x_1, x_2), \mathbf{label}_{\ell_2}(x_2), \ldots \\ , \mathbf{child}(x_{n-1}, x_n), \mathbf{label}_{\ell_n}(x_n), Column_n(x_n),$$

where ℓ_j for $1 \leqslant j \leqslant n$ denotes the j-th position of the designated last row ℓ. And so the predicate ℓ holds for a node if it completes a row where all tiles correspond to the tiles of the given last row.
To finalize the top-down scan, we introduce the rules to assign a leaf as candidate

$$\mathit{Candidate}(x) \leftarrow \mathbf{leaf}(x), \mathbf{label}_{\perp}(x), \mathbf{child}(y, x), \mathit{path}(y), \mathit{Buggy}(x)$$
$$\mathit{Candidate}(x) \leftarrow \mathbf{leaf}(x), \mathbf{label}_{!}(x), \mathbf{child}(y, x), \mathit{path}(y), \mathit{Okay}(x), \ell(x)$$

Now, by performing a bottom-up scan of T, we define the set of nodes that are *candidates* as follows:

- For each node x of T that is labeled 1, x is a *candidate* if x has a child that is a *candidate* and that is not labeled $\perp$.
- For each node x of T that is labeled 2, x is a *candidate* if
  - for each $d \in \mathcal{D}$, x has a child that is a *candidate* and that is labeled d, and
  - x has a child that is a *candidate* and that is *not* labeled $\perp$.

After the bottom-up scan, we can answer the query by **yes** if and only if the root node is a *candidate*.

We mark the root and the attached connected set of *candidates* as *relevant* nodes (performed by a second top-down scan of T) as follows:

- The root of T is *relevant* if it is a *candidate*.
- For each non-root node x of T, x is *relevant* if it is a *candidate* and its parent is *relevant*.

Note that according to this definition, in particular, the following is true:

- Every *relevant* node of T either is a leaf of T or has a child that is *relevant*.
- If the root of T is *relevant*, then it is labeled in $\mathcal{D} \times \{2\}$, and the set of all *relevant* nodes of T forms a tree, which we will call T_{Relevant}.
- Relevant nodes with labels $\perp$ or ! are leaves.
- Every *relevant* node with label in $\mathcal{D} \times \{1\}$ has a *relevant* child that is *not* labeled $\perp$.
- Every *relevant* node with label in $\mathcal{D} \times \{2\}$ has, for each $d \in \mathcal{D}$, a *relevant* child labeled d; and it has a *relevant* child that is *not* labeled $\perp$.

Thus, the following is true for every Σ-labeled tree T:

$(*)$: If the root of T is *relevant*, then
the tree $T_{Relevant}$ satisfies the conditions (1)–(9).

Furthermore, note that if T is *good*, then $T_{Relevant} = T$.

We say that a Σ-labeled tree T is *almost-good* if its root node is *relevant* and the tree $T_{Relevant}$ is *good*, i.e., satisfies the conditions (1)–(9).

Our next goal is to extend $\mathcal{P}$ to an mDatalog($\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}}$)-program $\mathcal{P}_{Relevant}$ which constructs, in an intensional predicate called *Relevant*, the set of all *relevant* nodes. We start with $\mathcal{P}_{Relevant} := \mathcal{P}$.

To describe the *candidate* nodes, we add to $\mathcal{P}_{Relevant}$ the rules

$$\mathit{Candidate}(x) \leftarrow \mathbf{label}_1(x), \mathbf{child}(x,y), \mathit{Candidate}(y), \mathbf{label}_{\neq\perp}(y),$$

as well as the following rule, where $d_1, \ldots, d_m$ is a list of all elements in $\mathcal{D}$:

$$\begin{aligned}\mathit{Candidate}(x) \leftarrow\ & \mathbf{label}_2(x), \mathbf{child}(x,y_1), \ldots, \mathbf{child}(x,y_m),\\ & \mathit{Candidate}(y_1), \ldots, \mathit{Candidate}(y_m),\\ & \mathbf{label}_{d_1}(y_1), \ldots, \mathbf{label}_{d_m}(y_m),\\ & \mathbf{child}(x,y), \mathit{Candidate}(y), \mathbf{label}_{\neq\perp}(y).\end{aligned}$$

To describe the *relevant* nodes, we add to $\mathcal{P}_{Relevant}$ the rules

$$\begin{aligned}\mathit{Relevant}(x) &\leftarrow \mathbf{root}(x), \mathit{Candidate}(x), \mathbf{label}_2(x)\\ \mathit{Relevant}(x) &\leftarrow \mathit{Candidate}(x), \mathbf{child}(y,x), \mathit{Relevant}(y)\end{aligned}$$

This completes the definition of the monadic datalog program $\mathcal{P}_{Relevant}$.

In summary, for each TPCT-instance $I = (\mathcal{D}, H, V, n, f, \ell)$, we can construct within polynomial time the alphabet $\Sigma := \mathcal{D} \times \{1, 2, \perp, !\}$ and a Boolean mDatalog($\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}}$)-query Q such that the following is true for every unranked unordered Σ-labeled tree T:

$$Q(T) = \mathbf{yes} \iff \begin{array}{c}\text{the root of } T \text{ is } \textit{relevant} \text{ and}\\ \text{the tree } T_{Relevant} \text{ satisfies the conditions (1)–(9).}\end{array}$$

Thus, $Q \neq \varnothing$ if, and only if, Player 1 has a winning strategy in the two person corridor tiling game specified by I. Hence, we have established a polynomial-time reduction from TPCT to the complement of the emptiness problem for Boolean mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) on unranked unordered labeled trees. This completes the proof of Proposition 4.1. □

Remark 4.3. *Note that in this approach the relations* **root** *and* **leaf** *are used as anchor for the scans. By Proposition 3.7, we know there is no possibility to extend this program by intensional predicates* **root** *and* **leaf** *that would fulfill this role on the same tree.*

4.2 The Hardness of the Emptiness Problem of mDatalog on Unranked Unordered Trees in General

In this section we generalize the result of Section 4.1 to unranked unordered labeled trees in general and we will prove the following proposition.

Proposition 4.4. *The emptiness problem for Boolean mDatalog(τ_u) on unranked unordered labeled trees is* Exptime*-hard.*

The proof of the proposition needs the term of a connected query and Lemma 4.5 given hereafter.

Let τ be a schema consisting of relations of arity at most two. Let r be a rule of a monadic datalog query of schema τ. The *undirected rule graph* G_r is the graph whose vertex set is the set of variables of r, and for two nodes x and y from $\mathrm{var}(r)$ we have $(x, y) \in E^{G_r}$ if a binary atom of the form $R(x, y)$ or $R(y, x)$ for a binary symbol $R \in \tau$ occurs in the rule's body. A rule in monadic datalog is *connected* if its rule graph G_r is connected. A query $Q = (\mathcal{P}, P)$ is *connected* if every rule of its program $\mathcal{P}$ is connected. Every connected rule can only speak about local node properties and their relations. An unconnected rule indicates that a property of the whole tree is tested and the property does not have to be complied nearby the node that will be evaluated with the head variable.

Lemma 4.5. *For every mDatalog($\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}}$)-query Q there exists an mDatalog($\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}}$)-query Q_c that is connected and equivalent on labeled unordered trees. There is an algorithm that computes Q_c from Q in time polynomial in the size of Q.*

Proof. The key idea of the proof is to identify the unconnected parts of the rules. These parts speak about properties of the whole tree and do not depend on the node associated with the head variable. We introduce new idb-predicates as switches which hold for every node of the tree if the tree fulfills the property and for none if not.

Let $Q = (\mathcal{P}, P)$ be the input query. The algorithm can proceed rulewise since a query is connected if every rule of its program is connected. Initially, let $\mathcal{P}'$ be the empty set. If the i-th rule r_i of $\mathcal{P}$ is connected, we add r_i to $\mathcal{P}'$. Otherwise, let G_{r_i} consist of n connected components C_0^i, C_1^i, …, C_{n-1}^i, for an $n \in \mathbb{N}_{\geqslant 1}$. W.l.o.g, let C_0^i contain the head variable and for a component C_j^i let $atoms(C_j^i)$ be the set of all (unary and binary) body atoms which contain variables of C_j^i and occur in r_i. Furthermore, let x be the head variable and $head_{r_i}$ be the head atom. Now, we choose for every $j \in \mathbb{N}_{\geqslant 1}$ with $1 \leqslant j \leqslant n-1$ a variable x_j that is contained in C_j^i and add the following rules to $\mathcal{P}'$:

$$C_j^i(x_j) \leftarrow atoms(C_j^i)$$
$$C_j^i(x_j) \leftarrow \mathbf{child}(x, x_j), C_j^i(x)$$
$$C_j^i(x_j) \leftarrow \mathbf{child}(x_j, x), C_j^i(x)$$

By these rules we distribute the tree properties tested in the components over all nodes of the tree and can retrieve these properties in the head variable by the following rule

$$head_{r_i}(x) \leftarrow atoms(C_0^i), C_1^i(x), \ldots, C_{n-1}^i(x).$$

Finally, the query $Q' = (\mathcal{P}', P)$ is an equivalent query that is connected. This is computed in polynomial time in the size of the input query and the correctness can be verified quickly by the semantics of mDatalog over labeled unordered trees.

Note that this lemma and its proof can be easily modified for all introduced schemas over unordered and ordered trees. □

Now we are ready to prove Proposition 4.4.

Proof of Proposition 4.4:

By Proposition 4.1 it suffices to construct a polynomial time reduction from the emptiness problem for mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) to the emptiness problem for mDatalog(τ_u). As we know by the proof of Proposition 4.1 and Remark 4.3, the knowledge of the root node and the leaves is essential for the reduction from the TPCT and so an extension of the proof of Proposition 4.1 is less promising. Similarly, as a reduction from the emptiness problem of mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) that constructs a query for the same (potentially relabeled) tree does not seem to be very promising.

To prove Proposition 4.4, we will modify the input query Q in mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) such that the obtained query in mDatalog(τ_u) accepts exactly the trees labeled by an extended alphabet $\tilde{\Sigma}$ which contain a tree that will be accepted by the original query Q. Since a $\tilde{\Sigma}$-labeled tree potentially represents more than one Σ-labeled tree, subtrees are separated by a unique symbol in the node above the root-labeled node. In a first step, the program $\mathcal{P}_{Area}$ marks these subtrees with the *Area*-predicate and their roots and leaves by the **root** and **leaf**-predicate. The *Area*-predicate together with the premise of a connected query ensures that tree properties of a subtree will only affect the evaluation in this subtree because if one rule leaves one of these subtrees it must speak about at least one separator node that is not a member of any area.

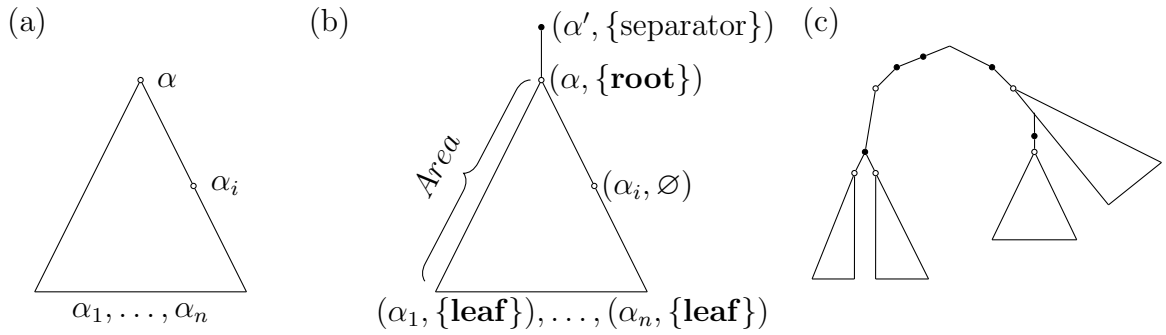

Figure 4.2: (a) A Σ-labeled tree T and (b) its corresponding $\tilde{\Sigma}$-labeled tree $\tilde{T}$ where α' is any symbol from Σ. In $\tilde{T}$ the node above the original root is labeled as separator, the originial root is additionally labeled by **root** such as the originial leaves are additionally labeled by **leaf**, and finally all inner nodes carry the $\varnothing$-symbol. (c) All separator nodes are filled black and all root nodes of subtrees corresponding to a tree depicted in (b) are filled white. In particular, a subtree in (c) only consists of its root node. The constructed query will answer on this input **yes** if and only if at least one of the five depicted subtrees is accepted by the original query.

Let Σ and $Q = (\mathcal{P}, P)$ be an input for the emptiness problem for mDatalog($\tau_u^{\mathbf{root},\mathbf{leaf}}$). We choose $\tilde{\Sigma} := \Sigma \times 2^{\{\text{separator},\mathbf{root},\mathbf{leaf}\}}$.

Let $\mathcal{P}^1_{Area}$ be the first part of the mDatalog($\tau_{u,\tilde{\Sigma}}$)-program $\mathcal{P}_{Area}$ consisting of the

following rules:

$$\begin{aligned}
\mathbf{label}_{\alpha}(x) &\leftarrow \mathbf{label}_{(\alpha,I)}(x)\\
\mathit{separator}(x) &\leftarrow \mathbf{label}_{(\alpha,\{\text{separator}\})}(x)\\
\mathit{cand}_{\text{root_and_leaf}}(x) &\leftarrow \mathbf{label}_{(\alpha,\{\mathbf{root},\mathbf{leaf}\})}(x)\\
\mathit{cand}_{\text{root_not_leaf}}(x) &\leftarrow \mathbf{label}_{(\alpha,\{\mathbf{root}\})}(x)\\
\mathit{cand}_{\text{leaf_not_root}}(x) &\leftarrow \mathbf{label}_{(\alpha,\{\mathbf{leaf}\})}(x)\\
\mathit{cand}_{\text{inner_node}}(x) &\leftarrow \mathbf{label}_{(\alpha,\varnothing)}(x)
\end{aligned}$$

for all $\alpha \in \Sigma$ and all $I \subseteq \{\text{separator}, \mathbf{root}, \mathbf{leaf}\}$. The first rule of $\mathcal{P}^1_{Area}$ provides the original labels for the rewritten rules of the original query later on. Furthermore, we mark the separator nodes as well as the nodes as candidates for the root node, the leaves, and the inner nodes. Note if a node carries an inconsistent label like $(\alpha, \{\mathbf{root}, \text{separator}\})$, it will not longer be considered for a potential subtree.

The mDatalog($\tau_{u,\tilde{\Sigma}}$)-program $\mathcal{P}^2_{Area}$ marks potential root nodes as root candidates if their parent node is a separator:

$$\begin{aligned}
\mathit{cand}_{root}(x) &\leftarrow \mathit{separator}(y), \mathbf{child}(y,x), \mathit{cand}_{\text{root_and_leaf}}(x)\\
\mathit{cand}_{root}(x) &\leftarrow \mathit{separator}(y), \mathbf{child}(y,x), \mathit{cand}_{\text{root_not_leaf}}(x)
\end{aligned}$$

The mDatalog($\tau_{u,\tilde{\Sigma}}$)-program $\mathcal{P}^3_{Area}$ distributes the **root**- and **leaf**-predicates for all subtrees consisting of only one node:

$$\begin{aligned}
\mathbf{root}(x) &\leftarrow \mathit{cand}_{root}(x), \mathit{cand}_{\text{root_and_leaf}}(x)\\
\mathbf{leaf}(x) &\leftarrow \mathit{cand}_{root}(x), \mathit{cand}_{\text{root_and_leaf}}(x)
\end{aligned}$$

For subtrees with several nodes an assignment in this way is not possible since we have to check whether on the path between any leaf candidate and its root there exists a node that is not labeled as inner node or if there is a root candidate located above the leaf candidate. To do so, we parse the potential subtree twice by the following mDatalog($\tau_{u,\tilde{\Sigma}}$)-program $\mathcal{P}^4_{Area}$.

$$\begin{aligned}
\mathbf{leaf}_{\mathit{inner_cand}}(x) &\leftarrow \mathit{cand}_{\text{inner_node}}(x), \mathbf{child}(x,y), \mathit{cand}_{\text{leaf_not_root}}(y)\\
\mathbf{leaf}_{\mathit{inner_cand}}(x) &\leftarrow \mathit{cand}_{\text{inner_node}}(x), \mathbf{child}(x,y), \mathbf{leaf}_{\mathit{inner_cand}}(y)\\
\mathbf{root}(x) &\leftarrow \mathit{cand}_{root}(x), \mathbf{child}(x,y), \mathbf{leaf}_{\mathit{inner_cand}}(y)\\
\mathit{inner_node}(x) &\leftarrow \mathbf{root}(y), \mathit{cand}_{\mathit{root_not_leaf}}(y), \mathbf{child}(y,x), \mathbf{leaf}_{\mathit{inner_cand}}(x)\\
\mathit{inner_node}(x) &\leftarrow \mathit{inner_node}(y), \mathbf{child}(y,x), \mathbf{leaf}_{\mathit{inner_cand}}(x)\\
\mathbf{leaf}(x) &\leftarrow \mathit{inner_node}(y), \mathbf{child}(y,x), \mathit{cand}_{\text{leaf_not_root}}(x)
\end{aligned}$$

And $\mathcal{P}^5_{Area}$ considers the case that between a leaf and its root there is no inner node:

$$\begin{aligned}
\mathbf{root}(x) &\leftarrow \mathit{cand}_{root}(x), \mathit{cand}_{\text{root_not_leaf}}(x), \mathbf{child}(x,y), \mathit{cand}_{\text{leaf_not_root}}(y)\\
\mathbf{leaf}(x) &\leftarrow \mathbf{root}(y), \mathit{cand}_{\text{root_not_leaf}}(y), \mathbf{child}(y,x), \mathit{cand}_{\text{leaf_not_root}}(x)
\end{aligned}$$

Finally, $\mathcal{P}^6_{Area}$ marks every correctly labeled subtree as tree area:

$$\begin{aligned}
\mathit{Area}(x) &\leftarrow \mathbf{leaf}(x)\\
\mathit{Area}(x) &\leftarrow \mathbf{root}(x)\\
\mathit{Area}(x) &\leftarrow \mathit{inner_node}(x)
\end{aligned}$$

It is easy to verify that the mDatalog($\tau_{u,\tilde{\Sigma}}$)-program $\mathcal{P}_{Area} := \mathcal{P}^1_{Area} \cup \mathcal{P}^2_{Area} \cup \mathcal{P}^3_{Area} \cup \mathcal{P}^4_{Area} \cup \mathcal{P}^5_{Area} \cup \mathcal{P}^6_{Area}$ is constructed completely independent of the input query Q and therefore in constant time. Furthermore, it provides the predicates **root** and **leaf** for all subtrees representing a tree of the original input and marks all nodes of these subtrees by $Area$. Additionally, it provides the orginal labels through the $\mathbf{label}_\alpha$-predicates.

Now, for the input query $Q = (\mathcal{P}, P)$, we assume Q is connected, otherwise we use Lemma 4.5 to obtain a connected query. From $\mathcal{P}$ we obtain the program $\mathcal{P}'$ by adding rulewise to every rule r the atoms

$$\{Area(x) \mid x \in \mathrm{var}(r)\}.$$

The resulting query $\tilde{Q} = (\tilde{\mathcal{P}}, \tilde{P})$ with

$$\tilde{\mathcal{P}} := \mathcal{P}' \cup \mathcal{P}_{Area} \cup \{\ \tilde{P}(x) \leftarrow \mathbf{root}(x), P(x)\ \} \cup \{\ \tilde{P}(x) \leftarrow \mathbf{child}(x,y), \tilde{P}(y)\ \}$$

is obtained from Q in polynomial time.

Let T be a Σ-labeled tree such that $Q(T) = \mathbf{yes}$. Then we know by construction that $\tilde{Q}(\tilde{T}) = \mathbf{yes}$ for the $\tilde{\Sigma}$-labeled tree $\tilde{T}$ obtained from T (cf. Figure 4.2).

Let $\tilde{T}$ be a $\tilde{\Sigma}$-labeled tree such that $\tilde{Q}(\tilde{T}) = \mathbf{yes}$. Then at least one with $Area$ marked subtree of $\tilde{T}$ is accepted by the mDatalog($\tau_{u,\tilde{\Sigma}}$)-query $\tilde{Q}$. Let $\tilde{T}_{sub}$ be one of them. In particular, we know that for the **root**-labeled node v of $\tilde{T}_{sub}$, we have

$$P(v) \in \mathcal{T}^\omega_{\tilde{\mathcal{P}}}(\tilde{T}) \quad \text{and} \quad \tilde{P}(v) \in \mathcal{T}^\omega_{\tilde{\mathcal{P}}}(\tilde{T}).$$

Now clearly, the Σ-reduct T of $\tilde{T}_{sub}$ is a witness for $Q \neq \varnothing$.

Thus, we presented a reduction from the emptiness problem for Boolean mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) to the emptiness problem for Boolean mDatalog(τ_u) within polynomial time such that for the input Σ and Q, as well as for the extended alphabet $\tilde{\Sigma}$ and the constructed query $\tilde{Q}$ the following is true.

There exists a Σ-labeled unordered tree T such that $Q(T) = \mathbf{yes}$.	$\Leftrightarrow$	There exists a $\tilde{\Sigma}$-labeled unordered tree T' such that $\tilde{Q}(T') = \mathbf{yes}$.

By this reduction and Proposition 4.1 we obtain the statement of Proposition 4.4. □

4.3 The Hardness of the Emptiness Problem of mDatalog on Ranked Unordered Trees

The section's goal is to prove the EXPTIME-hardness of the emptiness problem of mDatalog on ranked unordered trees. Just like in the unranked case, the proof will proceed by a reduction from the two person corridor tiling problem, but in contrast to the proof of Proposition 4.1 the strategy tree directly will be the accepted tree of our datalog program and it will not be a hidden subtree. Additionally, the next step which simulates the **root** and **leaf**-predicate is unlikely nicer since the rank gives us the necessary possibilities.

Proposition 4.6. *The emptiness problem for Boolean mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) on ranked unordered labeled trees is* EXPTIME*-hard.*

Proof. Similar to the proof of Proposition 4.1, our proof proceeds by a polynomial time reduction from TPCT which is Exptime-complete by Theorem 4.2. The problem is described in the mentioned proof on page 41 f. Note that Exptime is closed under complementation. Thus, for proving Proposition 4.6 it suffices to give a polynomial-time reduction from TPCT to the complement of the emptiness problem for mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf}\}}$) on ranked unordered labeled trees. For a given TPCT-instance $I = (\mathcal{D}, H, V, n, f, \ell)$ we will construct a ranked finite alphabet Σ and a Boolean mDatalog($\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}}$)-query Q, such that

$$\begin{array}{ll} & \text{Player 1 has a winning strategy in the} \\ & \text{two person corridor tiling game specified by } I \\ \iff & \text{there exists an ranked unordered } \Sigma\text{-labeled tree } T \\ & \text{such that } Q(T) = \mathbf{yes} \text{ (i.e., } Q \neq \varnothing). \end{array}$$

Again, we will represent strategies for Player 1 by Σ-labeled trees. The query Q will describe necessary properties which are met by every tree that describes a winning strategy for Player 1 and we represent a strategy for Player 1 in the same manner. Hence, we choose the ranked alphabet $\Sigma = (\sigma, ar)$ with

$$\sigma \;:=\; \mathcal{D} \times \{1, 2, \bot, !\}$$

and for $\alpha = (a, I) \in \sigma$ with $a \in \mathcal{D}$ and $I \in \{1, 2, \bot, !\}$ we define

$$ar(\alpha) = \begin{cases} 0 & \text{if } I = \bot \text{ or } I = ! \\ 1 & \text{if } I = 1 \\ |\mathcal{D}| & \text{if } I = 2 \end{cases}$$

Obviously, we could now take this ranked alphabet and the proof of Proposition 4.1 would work as well for ranked unordered trees: However, we can perform it in a shorter, more elegant way. We recall that Player 1 has a winning strategy if and only if there exists a finite Σ-labeled tree that in the actual context is ranked and satisfies the following conditions (1)–(9).

(1) The root is labeled by $(d, 2)$ for some $d \in \mathcal{D}$. (This indicates that at the beginning of the game, Player 1 chooses tile d, and Player 2 is next).

(2) Nodes with labels $\bot$ or ! are leaves.

(3) Nodes with labels in $\mathcal{D} \times \{1\}$ have at least one child. (Such a child describes the choice made by Player 1 in the next step).

(4) Nodes with labels in $\mathcal{D} \times \{2\}$ have at least $|\mathcal{D}|$ children – one for each tile type $d \in \mathcal{D}$. (These children represent the potential choices that Player 2 might make in the next step).

(5) There is no node labeled 1 or 2 such that all of its children are labeled by $\bot$. (I.e., the game never gets stuck).

(6) Labels from $\mathcal{D} \times \{1\}$ and $\mathcal{D} \times \{2\}$ alternate on each path from the root to a leaf. (I.e., both players alternately choose a tile).

(7) If a node x is labeled !, then the number of nodes visited by the path from the root to x is a multiple of n and the last n nodes on this path are labeled according to ℓ. (This means that the last n nodes of the path describe a row which has the desired labeling ℓ.)

(8) At each node x labeled (d, i) with $i \neq \perp$, the tile d respects the horizontal and the vertical constraints.

(9) At each node labeled $(d, \perp)$ for some $d \in \mathcal{D}$, the tile d violates the horizontal or the vertical constraints.

Remember, the conditions (8) and (9) mean the following. We define the *depth* of a node as follows: The root has depth 1; and for each node x of depth j, all children of x are of depth $j+1$.

(a) A node x labeled with tile $d \in \mathcal{D}$ *respects the horizontal constraints* if x

- is either of depth congruent 1 modulo n (and thus corresponds to a position in the first column of a row),
- or we have $(d_h, d) \in H$, where the parent of x is labeled with tile $d_h \in \mathcal{D}$ (i.e., x corresponds to a position where tile d is chosen in some column $j \geqslant 2$, and this tile fits to its horizontal neighbor d_h in column $j-1$).

(b) A node x labeled with a tile $d \in \mathcal{D}$ *respects the vertical constraints* if x

- is either of depth $j \in \{1, \ldots, n\}$ and we have $(f_j, d) \in H$
  (i.e., x corresponds to the j-th position in the second row and fits to the j-th entry f_j of the first row f),
- or it is of depth $j \geqslant n+1$ and we have $(d_v, d) \in V$, where the ancestor of x at depth $j-n$ is labeled with tile $d_v \in \mathcal{D}$
  (i.e., x corresponds to a position where tile d is chosen in some row $i \geqslant 3$, and this tile fits to its vertical neighbor d_v in row $i-1$).

As noted above, Player 1 has a winning strategy if and only if there exists a finite Σ-labeled tree T that satisfies the conditions (1)–(9). We realize that the conditions (2) is already granted by the defined ranked alphabet Σ. For condition (3) the alphabet granted exactly one child and the first part of condition (4) the ranked alphabet Σ granted exactly $|\mathcal{D}|$ children. This implies that the described tree defines exactly one strategy. In the former definition, the tree can contain more than one strategy.

Now, we start to construct the program $\mathcal{P}$ such that finally for the Boolean mDatalog($\tau_{u,\Sigma}^{\{\mathbf{root},\mathbf{leaf}\}}$)-query $Q = (\mathcal{P}, P)$ and every ranked Σ-labeled tree T we have:

$$Q(T) = \mathbf{yes} \iff T \text{ fulfills conditions (1)–(9).}$$

For accessing the parts d and i of a node-label $(d, i) \in \Sigma$, it will be convenient to include into $\mathcal{P}$ the rules

$$\mathbf{label}_d(x) \leftarrow \mathbf{label}_{(d,i)}(x) \qquad \text{and} \qquad \mathbf{label}_i(x) \leftarrow \mathbf{label}_{(d,i)}(x)$$

for every letter $(d, i) \in \Sigma$. Furthermore, for all $d, d' \in \mathcal{D}$ and $i, i' \in \{1, 2, \bot, !\}$ with $d \neq d'$ and $i \neq i'$ we add to $\mathcal{P}$ the rules

$$\mathbf{label}_{\neq d}(x) \leftarrow \mathbf{label}_{d'}(x) \qquad \text{and} \qquad \mathbf{label}_{\neq i}(x) \leftarrow \mathbf{label}_{i'}(x).$$

For every node we introduce the predicate *alt* to indicate that the path from the root to this node starts with a label from $\mathcal{D} \times \{2\}$ and continues with and alternation from $\mathcal{D} \times \{1\}$ and $\mathcal{D} \times \{2\}$ by the rules

$$alt(x) \leftarrow \mathbf{root}(x), \mathbf{label}_2(x)$$

and

$$alt(x) \leftarrow alt(y), \mathbf{child}(y, x), \mathbf{label}_i(x), \mathbf{label}_{i'}(y)$$

for $i, i' \in \{1, 2\}$ with $i \neq i'$.

To count the multiples of n and to detect a violation or satisfaction of the horizontal and/or vertical condition, it will be convenient to use predicates $Column_j$ for each $j \in \{1, \ldots, n\}$, such that $Column_j(x)$ indicates that node x corresponds to a tile placed in column j of the corridor. Thus, we add to $\mathcal{P}$ the rules

$$\begin{aligned} Column_1(x) &\leftarrow \mathbf{root}(x) \\ Column_1(x) &\leftarrow \mathbf{child}(y, x), Column_n(y), \end{aligned}$$

and for each $j \in \{2, \ldots, n\}$ the rule

$$Column_j(x) \leftarrow \mathbf{child}(y, x), Column_{j-1}(y).$$

Furthermore, for each $j \in \{1, \ldots, n-1\}$ we add to $\mathcal{P}$ the rule

$$Column_{\neq n}(x) \leftarrow Column_j(x)$$

and for each $j \in \{2, \ldots, n\}$ we add to $\mathcal{P}$ the rule

$$Column_{\neq 1}(x) \leftarrow Column_j(x).$$

To verify condition (8) it will be convenient to use predicates $Okay_H$ and $Okay_V$ such that $Okay_H(x)$ (resp., $Okay_V(x)$) indicates that node x satisfies the horizontal (resp., the vertical) constraints. Thus, for all $(d_h, d) \in H$, we add to $\mathcal{P}$ the rules

$$\begin{aligned} Okay_H(x) &\leftarrow Column_1(x) \\ Okay_H(x) &\leftarrow Column_{\neq 1}(x), \mathbf{child}(y, x), \mathbf{label}_{d_h}(y), \mathbf{label}_d(x). \end{aligned}$$

Similarly, for all $(d_v, d) \in V$, we add add to $\mathcal{P}$ the rules

$$\begin{aligned} Okay_V(x) \leftarrow &\mathbf{child}(y_1, y_2), \ldots, \mathbf{child}(y_{n-1}, y_n), \mathbf{child}(y_n, x), \\ &\mathbf{label}_{d_v}(y_1), \mathbf{label}_d(x). \end{aligned}$$

To detect nodes that correspond to tiles placed in the corridor's second row, i.e., tiles that must fit to the given first row $f = (f_1, \ldots, f_n) \in \mathcal{D}^n$, we furthermore add for each $j \in \{1, \ldots, n\}$ and each $d \in \mathcal{D}$ with $(f_j, d) \in V$ the rule

$$Okay_V(x_j) \leftarrow \mathbf{root}(x_1), \mathbf{child}(x_1, x_2), \ldots, \mathbf{child}(x_{n-1}, x_n), \mathbf{label}_d(x_j).$$

We combine acceptance of the vertical and the horizontal constraint in a node by the following rule

$$Okay(x) \leftarrow Okay_H(x), Okay_V(x)$$

To detect a violation of the horizontal or vertical constraint in condition (9), it will be convenient to use predicates $Buggy_H$ and $Buggy_V$, such that $Buggy_H(x)$ (resp., $Buggy_V(x)$) indicates that node x violates the horizontal (resp., the vertical) constraints. Thus, for all $(d_h, d) \in \mathcal{D}^2 \setminus H$, we add to $\mathcal{P}$ the rule

$$Buggy_H(x) \leftarrow Column_{\neq 1}(x), \mathbf{child}(y, x), \mathbf{label}_{d_h}(y), \mathbf{label}_d(x)$$

Similarly, for all $(d_v, d) \in \mathcal{D}^2 \setminus V$, we add add to $\mathcal{P}$ the rule

$$\begin{aligned} Buggy_V(x) \leftarrow & \mathbf{child}(y_1, y_2), \ldots, \mathbf{child}(y_{n-1}, y_n), \mathbf{child}(y_n, x), \\ & \mathbf{label}_{d_v}(y_1), \mathbf{label}_d(x) \end{aligned}$$

To detect nodes that correspond to tiles placed in the corridor's second row, i.e., tiles that must fit to the given first row $f = (f_1, \ldots, f_n) \in \mathcal{D}^n$, we furthermore add for each $j \in \{1, \ldots, n\}$ and each $d \in \mathcal{D}$ with $(f_j, d) \notin V$, the rule

$$Buggy_V(x_j) \leftarrow \mathbf{root}(x_1), \mathbf{child}(x_1, x_2), \ldots, \mathbf{child}(x_{n-1}, x_n), \mathbf{label}_d(x_j).$$

Next, we introduce a predicate that is true for a node if any violation happened.

$$Buggy(x) \leftarrow Buggy_V(x) \qquad \text{and} \qquad Buggy(x) \leftarrow Buggy_H(x)$$

To verify condition (7) we add the following rule

$$\begin{aligned} l(x_n) \leftarrow & \mathbf{label}_{l_1}(x_1), \mathbf{child}(x_1, x_2), \mathbf{label}_{l_2}(x_2), \ldots \\ & , \mathbf{child}(x_{n-1}, x_n), \mathbf{label}_{l_n}(x_n), Column_n(x_n), \end{aligned}$$

where l_j for $1 \leqslant j \leqslant n$ denotes the j-th position of the designated last row l. And so the predicate l holds for a node if it completes a row where all tiles correlates to the tiles of the given last row.

To verify condition (5) we introduce the predicate $notall_\perp$ which is true for a node if and only if not every child is labeled by $\perp$.

$$\begin{aligned} notall_\perp(x) &\leftarrow \mathbf{child}(x, y), \mathbf{label}_1(y) \\ notall_\perp(x) &\leftarrow \mathbf{child}(x, y), \mathbf{label}_2(y) \\ notall_\perp(x) &\leftarrow \mathbf{child}(x, y), \mathbf{label}_!(y) \end{aligned}$$

Now, by performing a bottom-up scan of T, we define the set of nodes x by the predicate $Scan_{ok}$ such that the subtree defined by all the descendants of x and the node x as root defines a strategy for Player 1 from this point in the game, which means it fulfills all the conditions (2)–(9) for this part of the original tree. And after all, if the root is labeled by $(d, 2)$ for some $d \in \mathcal{D}$ then we have that the whole tree fulfills all the conditions (1)–(9).

$$\begin{aligned} Scan_{ok}(x) &\leftarrow \mathbf{leaf}(x), alt(x), \mathbf{label}_\perp(x), Buggy(x) \\ Scan_{ok}(x) &\leftarrow \mathbf{leaf}(x), alt(x), \mathbf{label}_!(x), Okay(x), l(x) \end{aligned}$$

In this way for a leaf x we have $Scan_{ok}(x) \in \mathcal{T}^{\omega}_{\mathcal{P}}(T)$ for a ranked unordered Σ-labeled tree if and only if the leaf corresponds to the last step in a winning game, which means the leaf itself satisfies conditions (1)–(9).

To continue, we proceed with the inner nodes

$$Scan_{ok}(x) \leftarrow \mathbf{label}_1(x), alt(x), notall_{\perp}(x), Okay(x), \mathbf{child}(x, y_1), Scan_{ok}(y_1)$$
$$Scan_{ok}(x) \leftarrow \mathbf{label}_2(x), alt(x), notall_{\perp}(x), Okay(x), \mathbf{child}(x, y_1), \ldots, \mathbf{child}(x, y_m), \mathbf{label}_{d_1}(y_1), \ldots, \mathbf{label}_{d_m}(y_m), Scan_{ok}(y_1), \ldots, Scan_{ok}(y_m)$$

where $d_1, \ldots, d_m$ is a list of all elements in $\mathcal{D}$. In this way ensure the remaining part of the condition (4) in the second rule.

And finally for the root not we add

$$P(x) \leftarrow \mathbf{label}_2(x), Scan_{ok}(x), \mathbf{root}(x).$$

In summary, for each TPCT-instance $I = (\mathcal{D}, H, V, n, f, \ell)$, we can construct within polynomial time the ranked alphabet Σ and a Boolean mDatalog($\tau^{\{\mathbf{root},\mathbf{leaf}\}}_{u,\Sigma}$)-query Q such that the following is true for every ranked unordered Σ-labeled tree T:

$$Q(T) = \mathbf{yes} \iff \text{the tree } T \text{ satisfies the conditions (1)–(9).}$$

Thus, $Q \neq \varnothing$ if, and only if, Player 1 has a winning strategy in the two person corridor tiling game specified by I. Hence, we have established a polynomial-time reduction from TPCT to the complement of the emptiness problem for Boolean mDatalog($\tau^{\{\mathbf{root},\mathbf{leaf}\}}_{u}$) on ranked unordered labeled trees. This completes the proof of Proposition 4.6. □

Proposition 4.7. *The emptiness problem for Boolean mDatalog($\tau^{\mathbf{root}}_{u}$) on ranked unordered labeled trees is* Exptime*-hard.*

Proof. The proof proceeds by a reduction from the emptiness problem for Boolean mDatalog($\tau^{\{\mathbf{root},\mathbf{leaf}\}}_{u}$) on ranked unordered labeled trees which is Exptime-hard by Proposition 4.6.

Let Σ and $Q = (\mathcal{P}, P)$ be the input for the original emptiness problem, thus a ranked alphabet and a Boolean mDatalog($\tau^{\{\mathbf{root},\mathbf{leaf}\}}_{u,\Sigma}$)-query. It is to decide whether we have $Q = \varnothing$.

By Proposition 3.17 we have an mDatalog($\tau_{u,\Sigma}$)-query $Q_{\mathbf{leaf}} = (\mathcal{P}_{\mathbf{leaf}}, \mathbf{leaf})$ constructible in linear time in the size of Σ such that $\mathbf{leaf}(v) \in \mathcal{T}^{\omega}_{\mathcal{P}}(T)$ for every ranked unordered tree T if and only if v is a leaf of T. Clearly, for the Boolean mDatalog($\tau^{\mathbf{root}}_{u,\Sigma}$)-query $Q' = (\mathcal{P}_{\mathbf{leaf}} \cup \mathcal{P}, P)$ we have $Q' = \varnothing$ if and only if $Q = \varnothing$. □

Proposition 4.8. *The emptiness problem for Boolean mDatalog(τ_u) on ranked unordered labeled trees is* Exptime*-hard.*

Proof. The proof proceeds by a reduction from the emptiness problem for Boolean mDatalog($\tau^{\mathbf{root}}_{u}$) on ranked unordered labeled trees which is Exptime-hard by Proposition 4.7.

Let the ranked alphabet Σ and the Boolean mDatalog($\tau^{\mathbf{root}}_{u,\Sigma}$)-query $Q = (\mathcal{P}, P)$ be the input for the original emptiness problem. It is to decide whether we have $Q = \varnothing$

by using the emptiness problem for Boolean mDatalog(τ_u) on ranked unordered labeled trees.

We define for Σ and $m = rk_{\max}(\Sigma)$ the set $RK = \{\mathbf{root}, 1, \ldots, m\}$ and the following ranked alphabet $\tilde{\Sigma} = (\tilde{\sigma}, \tilde{ar})$, where $\tilde{\sigma} = \Sigma \times RK$ and for a symbol $\alpha = (a, I) \in \Sigma \times RK$, we define $\tilde{ar}(\alpha) = ar(a)$. Now, for every Σ-labeled tree there is a set of $\tilde{\Sigma}$-labeled trees where the second component of the labeling defines the root and an order on all children relating to their parent nodes. This gives us the possibility to separate the children of a node.

Let $\tilde{\mathcal{P}}_{labels}$ be the mDatalog($\tau_{u,\tilde{\Sigma}}$)-program consisting of the rules

$$\begin{aligned} \mathbf{label}_a(x) &\leftarrow \mathbf{label}_{(a,I)}(x) \\ cand_{\mathbf{root}}(x) &\leftarrow \mathbf{label}_{(a,\mathbf{root})}(x) \\ 1st(x) &\leftarrow \mathbf{label}_{(a,1)}(x) \\ 2nd(x) &\leftarrow \mathbf{label}_{(a,2)}(x) \\ &\vdots \\ mth(x) &\leftarrow \mathbf{label}_{(a,m)}(x) \end{aligned}$$

for all $a \in \Sigma$ and all $I \in RK$.

Let $\tilde{\mathcal{P}}_{cons}$ be the mDatalog($\tau_{u,\tilde{\Sigma}}$)-program consisting of the rules of $\tilde{\mathcal{P}}_{labels} \cup \mathcal{P}_{\mathbf{leaf}}$ (cf. Proposition 4.7) along with the following rules:

$$P_{cons}(x) \leftarrow \mathbf{leaf}(x)$$

and for every $\alpha \in \tilde{\Sigma}$ and its arity $n = \tilde{ar}(\alpha)$ we have

$$\begin{aligned} P_{cons}(x) \leftarrow \mathbf{label}_\alpha(x), \mathbf{child}(x, y_1), 1st(y_1), P_{cons}(y_1), \\ \mathbf{child}(x, y_2), 2nd(y_2), P_{cons}(y_2), \ldots, \mathbf{child}(x, y_n), nth(y_n), P_{cons}(y_n) \end{aligned}$$

and finally

$$\mathbf{root}(x) \leftarrow P_{cons}(x), cand_{\mathbf{root}}(x).$$

It is easy to verify that for a ranked unordered $\tilde{\Sigma}$-labeled tree and its node $v \in T$ we have $P_{cons}(v) \in \mathcal{T}^{\omega}_{\tilde{\mathcal{P}}_{cons}}(T)$ if itself and all its descendants have numerated children and in particular, none of its descendant is labeled as root. This implies that the Boolean mDatalog($\tau_{u,\tilde{\Sigma}}$)-query $Q_{\mathbf{root}} = (\tilde{\mathcal{P}}_{cons}, \mathbf{root})$ yields **yes** on input T only if its root node and only its root node is labeled by $(a, \mathbf{root})$ for any symbol $a \in \Sigma$.

Now, we choose the resulting mDatalog($\tau_{u,\tilde{\Sigma}}$)-query $Q' = (\mathcal{P}', P')$ as follows:

$$\mathcal{P}' := \mathcal{P} \cup \tilde{\mathcal{P}}_{cons} \cup \{\ P'(x) \leftarrow P(x), \mathbf{root}(x)\ \}$$

This construction is done in polynomial time in the size of the input alphabet Σ together with Q and it remains to verify the following claim:

$$Q = \varnothing \qquad \text{if and only if} \qquad Q' = \varnothing.$$

Assume $Q \neq \varnothing$ which means there exists a ranked unordered Σ-labeled tree T such that $Q(T) = \mathbf{yes}$. Clearly, there exists a $\tilde{\Sigma}$-labeled tree T' such that the Σ-reduct [2] of T' is T, the root node of T' is labeled as **root**, all children of any node are enumerated, T fulfills Q and therefore we have $Q'(T') = \mathbf{yes}$.

Assume $Q' \neq \varnothing$ that implies there exists a ranked unordered $\tilde{\Sigma}$-labeled tree T' such that $Q'(T') = \mathbf{yes}$ which implies we have $P(root^{T'}) \in \mathcal{T}^{\omega}_{\mathcal{P}'}(T')$ and $\mathbf{root}(root^{T'}) \in \mathcal{T}^{\omega}_{\mathcal{P}'}(T')$. The latter ensures that for T' the root node and only the root node is labeled correctly as root. This together with the fact $P(root^{T'}) \in \mathcal{T}^{\omega}_{\mathcal{P}'}(T')$ ensures that $Q(T) = \mathbf{yes}$ for the Σ-reduct T of T'. □

Via polynomial-time reduction from the emptiness problem we obtain Exptime-hardness for the remaining problems on unordered trees.

Corollary 4.9.

(a) The equivalence problem for Boolean mDatalog(τ_u) on labeled unordered trees is Exptime*-hard.*

(b) The query containment problem for Boolean mDatalog(τ_u) on labeled unordered trees is Exptime*-hard.*

Proof. (a) This proof proceeds by reduction from the emptiness problem for Boolean mDatalog(τ_u) on labeled unordered trees which is Exptime-hard by Proposition 4.4 in the unranked case and by Proposition 4.8 in the ranked case. The task is to decide whether a given Boolean mDatalog($\tau_{u,\Sigma}$)-query Q is empty or not. Let $Q^{\varnothing}_{\tau_u}$ be a query that is unsatisfiable, for expample the query $Q^{\varnothing}_{\tau_u}$ from Example 2.10. Now, the following is true

$$Q \equiv \varnothing \qquad \Leftrightarrow \qquad Q \equiv Q^{\varnothing}_{\tau_u}$$

Thus, we obtain a polynomial-time reduction from the emptiness problem for Boolean mDatalog(τ_u) to the equivalence problem for Boolean mDatalog(τ_u).

(b) Likewise to (a), this proof proceeds by a reduction from the emptiness problem for Boolean mDatalog(τ_u) on labeled unordered trees which is Exptime-hard. The task by using the query containment problem is to decide whether a given Boolean mDatalog($\tau_{u,\Sigma}$)-query Q is empty or not. Let $Q^{\varnothing}_{\tau_u}$ be a query in monadic datalog of the schema τ_u that is unsatisfiable, for expample the query $Q^{\varnothing}_{\tau_u}$ from Example 2.10. Now, the following is true.

$$Q \equiv \varnothing \qquad \Leftrightarrow \qquad Q \subseteq Q^{\varnothing}_{\tau_u}$$

Thus, we obtain a polynomial-time reduction from the emptiness problem for Boolean mDatalog(τ_u) to the query containment problem for Boolean mDatalog(τ_u). □

[2] The Σ-reduct T is obtained from T' by relabeling every node by its own first component of the label.

4.4 Transferring the Hardness Results to the Corresponding Problems of mDatalog on Ordered Trees

The aim of this section is to transfer the results of the previous sections to the emptiness, the equivalence, and the query containment problem on labeled ordered trees.

Corollary 4.10. *The emptiness problem for Boolean mDatalog(τ_o) on labeled ordered trees is* EXPTIME*-hard.*

The proof is via a polynomial-time reduction from the emptiness problem for Boolean mDatalog(τ_u) on labeled unordered trees which, according to Proposition 4.4 in the unranked case and Proposition 4.8 in the ranked case, is EXPTIME-hard.

For establishing the reduction, we will rewrite monadic datalog programs of schema $\tau_{u,\Sigma}$ into suitable programs of schema $\tau_{o,\Sigma}$ (i.e., we will rewrite the **child** relation by means of the relations **fc** and **ns**). For doing this, we can use a result by Gottlob and Koch [GK04] which transforms monadic datalog programs into a certain normal form called *tree-marking normal form* (TMNF).

Definition 4.11. *Let τ be a schema that consists of relation symbols of arity at most two. A monadic datalog program $\mathcal{P}$ of schema τ is in TMNF if each rule of $\mathcal{P}$ is of one of the following forms:*[3]

(i) $X(x) \leftarrow R(x,y), Y(y)$

(ii) $X(x) \leftarrow R(y,x), Y(y)$

(iii) $X(x) \leftarrow Y(x), Z(x)$

where R is a binary predicate from τ, $X \in \mathrm{idb}(\mathcal{P})$, and the unary predicates Y and Z are either intensional or belong to τ.

Theorem 4.12 (Gottlob and Koch [GK04, Theorem 5.2])**.**
For each monadic datalog program $\mathcal{P}$ of schema $\tau_{GK}^{\mathbf{child}}$, there is an equivalent program in TMNF of schema τ_{GK} which can be computed in time $O(\|\mathcal{P}\|)$.

A detailed analysis shows that the proof given in [GK04] in fact also proves the following:

Corollary 4.13 (implicit in [GK04])**.** *For each monadic datalog program $\mathcal{P}$ of schema $\tau_o^{\mathbf{child}}$, there is an equivalent program in TMNF of schema τ_o, which can be computed in time $O(\|\mathcal{P}\|)$.*

We are now ready for the proof of Corollary 4.10.

Proof of Corollary 4.10:
From Proposition 4.4 and Proposition 4.8 we already know the EXPTIME-hardness of the emptiness problem for Boolean mDatalog(τ_u) on labeled unordered trees. Thus, it suffices to give a polynomial-time reduction from this problem to the emptiness problem for Boolean mDatalog(τ_o) on labeled ordered trees.

[3] Gottlob and Koch [GK04] also allow rules of the form $X(x) \leftarrow Y(x)$. Note that such a rule is equivalent to the rule $X(x) \leftarrow Y(x), Y(x)$.

For this, note that $\tau_u \subseteq \tau_o^{\mathbf{child}}$. Thus, upon input of a query Q in Boolean mDatalog($\tau_{u,\Sigma}$), we can apply Corollary 4.13 to compute, in linear time, a Boolean mDatalog($\tau_{o,\Sigma}$)-query Q' such that $Q'(T) = Q(T)$ is true for all ordered trees T. Furthermore, since Q is of schema $\tau_{u,\Sigma}$, we have that $Q(T) = Q(\tilde{T})$ is true for all ordered trees T and their unordered version $\tilde{T}$. Thus, we have $Q = \varnothing$ if and only if $Q' = \varnothing$. Now we have established a polynomial-time reduction from the emptiness problem for unordered trees using schema $\tau_{u,\Sigma}$ to the emptiness problem for ordered trees using schema $\tau_{o,\Sigma}$. This completes the proof of Corollary 4.10. □

By rewriting monadic datalog programs of schema $\tau_{u,\Sigma}$ into suitable programs of schema $\tau_{o,\Sigma}$ analogously to the proof of Corollary 4.10 and Corollary 4.9 we immediately obtain the following corollary.

Corollary 4.14.

(a) The equivalence problem problem and

(b) the query containment

for Boolean mDatalog(τ_o) on labeled ordered trees are Exptime*-hard.*

Chapter 5

The Automata Point of View:
On Membership

In this chapter we consider the upper bound for the several problems in the case of an omitted *descendant*-axis. Programs containing a *desc*-predicate are considered in Chapter 6. First, we present a matching upper bound for the QCP for monadic datalog on unranked trees. To preserve clarity of the proof, we start with a sketch to give an idea of the proof and carry out the full proof step by step later on. After that, we proceed on by adopting the upper bound for the ranked case and finally, we close this chapter by extending the results to the membership of the emptiness problem and equivalence problem.

Theorem 5.1.
The query containment problem for unary mDatalog($\tau_{GK}^{\mathbf{child}}$) on unranked ordered labeled trees belongs to EXPTIME.

Sketch. Consider a schema $\tau \subseteq \tau_{GK,\Sigma}^{\mathbf{child},\mathbf{desc}}$. By using the automata-theoretic approach [CGKV88], a canonical method for deciding the QCP for unary mDatalog(τ) proceeds as follows:

(1) Transform the input queries Q_1 and Q_2 into *Boolean* queries Q_1' and Q_2' on *binary* trees such that $Q_1 \subseteq Q_2$ if and only if $Q_1' \subseteq Q_2'$.

(2) Construct tree automata $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$ such that $\mathtt{A}_1^{\mathbf{yes}}$ (resp. $\mathtt{A}_2^{\mathbf{no}}$) accepts exactly those trees T with $Q_1'(T) = \mathbf{yes}$ (resp. $Q_2'(T) = \mathbf{no}$).

(3) Construct the product automaton $\mathtt{B}$ of $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$ such that $\mathtt{B}$ accepts exactly those trees that are accepted by $\mathtt{A}_1^{\mathbf{yes}}$ and by $\mathtt{A}_2^{\mathbf{no}}$. Afterwards, check if the tree language described by $\mathtt{B}$ is empty. Note that this is the case if and only if $Q_1 \subseteq Q_2$.

With time polynomial in the size of Q_1 and Q_2, step (1) can be achieved in a standard way by appropriately extending the labeling alphabet Σ.

For step (3), if $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$ are nondeterministic bottom-up tree automata, the construction of $\mathtt{B}$ takes time polynomial in the sizes of $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$, and the emptiness test can be done in time polynomial in the size of $\mathtt{B}$ (see e.g. [CDG$^+$08]).

The first idea for tackling step (2) is to use a standard translation of Boolean monadic datalog queries into monadic second-order (MSO) sentences: By Proposition 3.3 we know

that any Boolean mDatalog(τ_Σ)-query Q can be translated in polynomial time into an equivalent MSO-sentence φ_Q of the form

$$\forall X_1 \cdots \forall X_n \; \exists z_1 \cdots \exists z_\ell \; \bigvee_{j=1}^{m} \gamma_j$$

where n is the number of intensional predicates of Q's monadic datalog program $\mathcal{P}$, ℓ and m are linear in the size of Q. Each γ_j is a conjunction of at most b atoms or negated atoms where b is linear in the maximum number of atoms occurring in the body of a rule of $\mathcal{P}$. Applying the standard method for translating MSO-sentences into tree automata (cf., e.g., [Tho97]), we can translate the sentence $\neg\varphi_Q$ into a nondeterministic bottom-up tree-automaton $\mathtt{A}^{\mathbf{no}}$ that accepts a tree T if and only if $Q(T) = \mathbf{no}$. This automaton has $2^{(m' \cdot c^{b'})}$ states, where m' and b' are linear in m and b, respectively, and c is a constant not depending on Q or Σ. Thus, $\mathtt{A}^{\mathbf{no}}$ can be constructed in time polynomial in $|\Sigma| \cdot 2^{n+\ell+m' \cdot c^{b'}}$.

Using the subset construction, we obtain an automaton $\mathtt{A}^{\mathbf{yes}}$ which accepts a tree T if and only if $Q(T) = \mathbf{yes}$. This automaton has $2^{2^{(m' \cdot c^{b'})}}$ states.

Note that, a priori, b' might be linearly related to the size of Q. Thus, the approach described so far leads to a 3-fold exponential algorithm that solves the QCP for unary mDatalog(τ).

In case that τ does not contain the **desc**-predicate, we obtain a 2-fold exponential algorithm as follows: At the end of step (1) we rewrite Q'_1 and Q'_2 into queries that do not contain the **child**-predicate and we transform both queries into *tree marking normal form* (TMNF). This is a normal form in which bodies of rules consist of at most two atoms where at least one is unary. By Theorem 4.12 we obtain that these transformations can be done in time polynomial in the size of Q'_1 and Q'_2. Note that for TMNF-queries, the parameters b and b' are constants (i.e., they do not depend on the query). Thus, the above description shows that for TMNF-queries the automaton $\mathtt{A}_2^{\mathbf{no}}$ can be constructed in 1-fold exponential time and $\mathtt{A}_1^{\mathbf{yes}}$ can be constructed in 2-fold exponential time.

Finally, the key idea to obtain a 1-fold exponential algorithm solving the QCP is to use a different construction for the automaton $\mathtt{A}_1^{\mathbf{yes}}$, which does not use the detour via an MSO-formula but, instead, takes a detour via a two-way alternating tree automaton (2ATA): We show that a Boolean TMNF-query can be translated in polynomial time into a 2ATA $\hat{\mathtt{A}}_1^{\mathbf{yes}}$ that accepts a tree T if and only if $Q_1(T) = \mathbf{yes}$. It is known that, within 1-fold exponential time, a 2ATA can be transformed into an equivalent nondeterministic bottom-up tree automaton (this was claimed already in [CGKV88]; detailed proofs of more general results can be found in [Var98, MFS10]). In summary, this leads to a 1-fold exponential algorithm for solving the QCP for mDatalog($\tau_{GK}^{\mathbf{child}}$) on ordered trees. □

Since $\tau_u^{\mathbf{root},\mathbf{leaf}} \subseteq \tau_{GK}^{\mathbf{child}}$, Theorem 5.1 immediately implies:

Corollary 5.2. *The query containment problem for unary mDatalog($\tau_u^{\mathbf{root},\mathbf{leaf}}$) on unranked unordered labeled trees belongs to* EXPTIME. ⌟

We proceed as described in the proof sketch given above. Section 5.1 and Section 5.2 contain the transformations of step (1). The construction of the nondeterministic bottom-up tree automata $\mathtt{A}_2^{\mathbf{no}}$ and the automata $\mathtt{A}_1^{\mathbf{yes}}$ of step (2), the latter via an 2ATA, can be found in Section 5.3 and Section 5.4, respectively. In Section 5.5, we use the results of the two previous sections and perform the remaining step (3) to obtain the proof of Theorem 5.1.

5.1 From Unary Queries to Boolean Queries

Let Σ be an unranked finite alphabet, let T be an unranked ordered Σ-labeled tree, and let v be a node of T. Considering the extended alphabet $\Sigma' := \Sigma \times \{0,1\}$, we represent the tuple (T, v) by an unranked ordered Σ'-labeled tree T'_v as follows: T'_v is obtained from T by changing the node labels, so that node v receives label $(\alpha_v, 1)$ and all further nodes u receive labels $(\alpha_u, 0)$ where α_v and α_u denote the labels of the nodes in T.

Lemma 5.3. *Every unary mDatalog($\tau^{\mathbf{child}}_{GK,\Sigma}$)-query Q can be rewritten, in linear time, into a Boolean mDatalog($\tau^{\mathbf{child}}_{GK,\Sigma'}$)-query Q'_{Bool} which satisfies the following:*

- *For every unranked ordered Σ-labeled tree T and every node v of T we have $v \in Q(T) \iff Q'_{Bool}(T'_v) = \mathbf{yes}$.*
- *For every unranked ordered Σ'-labeled tree T' with $Q'_{Bool}(T') = \mathbf{yes}$, there exists an ordered Σ-labeled tree T and a node v of T such that $T' = T'_v$.*

Proof. Let $Q = (\mathcal{P}, P)$. We will construct Q'_{Bool} as follows:

(i) Q'_{Bool} will simulate the program $\mathcal{P}$ of Q.

(ii) In parallel, Q'_{Bool} checks that the input tree contains exactly one node whose label is of the form $(\alpha, 1)$ for some $\alpha \in \Sigma$. We construct Q'_{Bool} in such a way that it returns true if and only if the input tree's root node receives the intensional predicate C_1.

(iii) Finally, the root node receives the query predicate of Q'_{Bool} if and only if it has the C_1-predicate and the query predicate P of the query Q contains a node with label $(\alpha, 1)$ for some $\alpha \in \Sigma$.

To this end, we let Q'_{Bool} be specified by a monadic datalog program $\mathcal{P}'$ and a query predicate P' chosen as follows:

Start with $\mathcal{P}' := \varnothing$. For each letter $\alpha \in \Sigma$, we add to $\mathcal{P}'$ the rules

$$\mathbf{label}_\alpha(x) \leftarrow \mathbf{label}_{(\alpha,0)}(x) \qquad X_0(x) \leftarrow \mathbf{label}_{(\alpha,0)}(x)$$
$$\mathbf{label}_\alpha(x) \leftarrow \mathbf{label}_{(\alpha,1)}(x) \qquad X_1(x) \leftarrow \mathbf{label}_{(\alpha,1)}(x)$$

where X_0 and X_1 are unary relation symbols that do not occur in $\mathcal{P}$.

Next, add to $\mathcal{P}'$ all rules of $\mathcal{P}$. Note that this way we ensure that $\mathcal{P}'$ simulates $\mathcal{P}$, and hence (i) is achieved.

To achieve (ii), we use two intensional predicates C_0 and C_1. We choose rules that proceed the tree built by the **fc** and **ns** relations in a right-to-left and bottom-up manner and propagates, via the predicates C_0 and C_1, whether the subtree rooted at the current node contains exactly none or exactly one node that carry the predicate X_1. This is

achieved by the following list of rules, which we add to $\mathcal{P}'$:

$$C_0(x) \leftarrow \mathbf{leaf}(x), \mathbf{ls}(x), X_0(x)$$
$$C_1(x) \leftarrow \mathbf{leaf}(x), \mathbf{ls}(x), X_1(x)$$

$$C_0(x) \leftarrow \mathbf{leaf}(x), \mathbf{ns}(x, y), X_0(x), C_0(y)$$
$$C_1(x) \leftarrow \mathbf{leaf}(x), \mathbf{ns}(x, y), X_0(x), C_1(y)$$
$$C_1(x) \leftarrow \mathbf{leaf}(x), \mathbf{ns}(x, y), X_1(x), C_0(y)$$

$$C_0(x) \leftarrow \mathbf{ls}(x), \mathbf{fc}(x, y), X_0(x), C_0(y)$$
$$C_1(x) \leftarrow \mathbf{ls}(x), \mathbf{fc}(x, y), X_0(x), C_1(y)$$
$$C_1(x) \leftarrow \mathbf{ls}(x), \mathbf{fc}(x, y), X_1(x), C_0(y)$$

$$C_0(x) \leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_0(x), C_0(y), C_0(z)$$
$$C_1(x) \leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_0(x), C_0(y), C_1(z)$$
$$C_1(x) \leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_0(x), C_1(y), C_0(z)$$
$$C_1(x) \leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_1(x), C_0(y), C_0(z)$$

Finally, we achieve (iii) by letting P' be a new intensional predicate and by adding to $\mathcal{P}'$ the rule

$$P'(x) \leftarrow \mathbf{root}(x), C_1(x), P(y), X_1(y).$$

Clearly, $\mathcal{P}'$ can be generated in time linear in the size of Q. □

As an immediate consequence, we obtain:

Lemma 5.4. *Let Σ be a finite alphabet and let $\Sigma' := \Sigma \times \{0, 1\}$. Within linear time, we can rewrite given unary mDatalog($\tau^{\mathbf{child}}_{GK,\Sigma}$)-queries Q_1 and Q_2 into Boolean mDatalog($\tau^{\mathbf{child}}_{GK,\Sigma'}$)-queries Q'_1 and Q'_2 such that $Q_1 \subseteq Q_2$ if and only if $Q'_1 \subseteq Q'_2$.*

Proof. For each $i \in \{1, 2\}$ let Q'_i be the query obtained by Lemma 5.3.

If $Q_1 \not\subseteq Q_2$, there are an ordered Σ-labeled tree T and a node v of T such that $v \in Q_1(T)$ and $v \notin Q_2(T)$. By Lemma 5.3 we obtain that $Q'_1(T'_v) = \mathbf{yes}$ and $Q'_2(T'_v) = \mathbf{no}$. Thus, $Q'_1 \not\subseteq Q'_2$.

If otherwise $Q'_1 \not\subseteq Q'_2$, there is an ordered Σ'-labeled tree T' such that $Q'_1(T') = \mathbf{yes}$ and $Q'_2(T') = \mathbf{no}$. Since $Q'_1(T') = \mathbf{yes}$, Lemma 5.3 tells us that there are an ordered Σ-labeled tree T and a node v of T such that $T' = T'_v$. Furthermore, by Lemma 5.3 we know that $v \in Q_1(T)$ and $v \notin Q_2(T)$. Thus, $Q_1 \not\subseteq Q_2$. □

Finally, we use Theorem 4.12 to eliminate the **child**-predicate and to obtain queries in TMNF.

Proposition 5.5. *Let Σ be a finite alphabet and let $\Sigma' := \Sigma \times \{0, 1\}$. Within linear time, we can rewrite given unary mDatalog($\tau^{\mathbf{child}}_{GK,\Sigma}$)-queries Q_1 and Q_2 into Boolean mDatalog($\tau_{GK,\Sigma'}$)-queries Q'_1 and Q'_2 such that $Q_1 \subseteq Q_2$ if and only if $Q'_1 \subseteq Q'_2$. Furthermore, the programs of Q'_1 and Q'_2 are in TMNF.*

Proof. We apply Lemma 5.4 to obtain Boolean queries Q'_1 and Q'_2. Afterwards, we apply Theorem 4.12 to rewrite the programs of the queries Q'_1 and Q'_2 into programs in TMNF of schema $\tau_{GK,\Sigma'}$. □

5.2 From Ordered Unranked Trees to Binary Trees

For achieving steps (2) and (3) we use the classical notion of nondeterministic tree automata which operate on ordered binary Σ-labeled trees. This section's goal is to fix notations concerning binary trees and to show that in order to prove Theorem 5.1 it suffices to find a 1-fold exponential algorithm that solves the QCP for Boolean queries in TMNF regarding binary trees.

5.2.1 Binary trees

An *ordered Σ-labeled binary tree* (for short: binary tree) $T = (V^T, \lambda^T, L^T, R^T)$ consists of a finite set V^T of nodes, a function $\lambda^T : V^T \to \Sigma$ assigning to each node v of T a label $\lambda^T(v) \in \Sigma$, and disjoint sets $L^T, R^T \subseteq V^T \times V^T$ such that the graph (V^T, E^T) with $E^T := L^T \cup R^T$ is a rooted directed tree where edges are directed from the root to the leaves, and each node has at most two children. For a tuple $(u, v) \in L^T$ (resp., R^T), we say that node v is the *left child* (resp., the *right child*) of node u.

We represent such a tree T as a relational structure of domain V^T with unary and binary relations: For each label $\alpha \in \Sigma$, $\mathbf{label}_\alpha(x)$ expresses that x is a node with label α; $\mathbf{lc}(x, y)$ (resp., $\mathbf{rc}(x, y)$) expresses that y is the left (resp., right) child of node x; $\mathbf{root}(x)$ expresses that x is the tree's root node; $\mathbf{has\_no\_lc}(x)$ (resp., $\mathbf{has\_no\_rc}(x)$) expresses that node x has no left child (resp., no right child), i.e., there is no node y with $(x, y) \in L^T$ (resp., R^T).

We denote this relational structure representing T by $\mathcal{S}_b(T)$, but when no confusion arises we simply write T instead of $\mathcal{S}_b(T)$. This relational structure is of schema

$$\tau_{b,\Sigma} := \{\mathbf{lc}, \mathbf{rc}\} \cup \{\mathbf{root}, \mathbf{has\_no\_lc}, \mathbf{has\_no\_rc}\} \cup \{\mathbf{label}_\alpha : \alpha \in \Sigma\}.$$

5.2.2 Representing Ordered Unranked Trees by Binary Trees

We use a slight variant of the standard representation (cf., e.g., [Nev02]) of ordered unranked trees by binary trees. We represent an ordered unranked Σ-labeled tree T by a binary tree $bin(T)$ as follows: $bin(T)$ has the same vertex set and the same node labels as T, the left child relation $L^{bin(T)}$ consists of all tuples (x, y) such that y is the first child of x in T (i.e., $\mathbf{fc}(x, y)$ is true in $\mathcal{S}_o(T)$), and the right child relation $R^{bin(T)}$ consists of all tuples (x, y) such that y is the next sibling of x in T (i.e., $\mathbf{ns}(x, y)$ is true in $\mathcal{S}_o(T)$).

Note that the relational structure $\mathcal{S}_b(bin(T))$ is obtained from the structure $\mathcal{S}_o(T)$ as follows:

- drop the relations **child** and **desc**,
- rename the relations **fc**, **ns**, **leaf**, **ls** into **lc**, **rc**, **has_no_lc**, **has_no_rc**, and
- insert the root node into the relation **has_no_rc**.

Furthermore, note that for a binary tree T' there exists an unranked ordered tree T with $T' = bin(T)$ if and only if the root of T' has no right child and in this case the tree T is unique.

Example 5.6. *Figure 5.1 shows how to obtain the binary tree bin(T) for the given tree T from Example 2.2.*

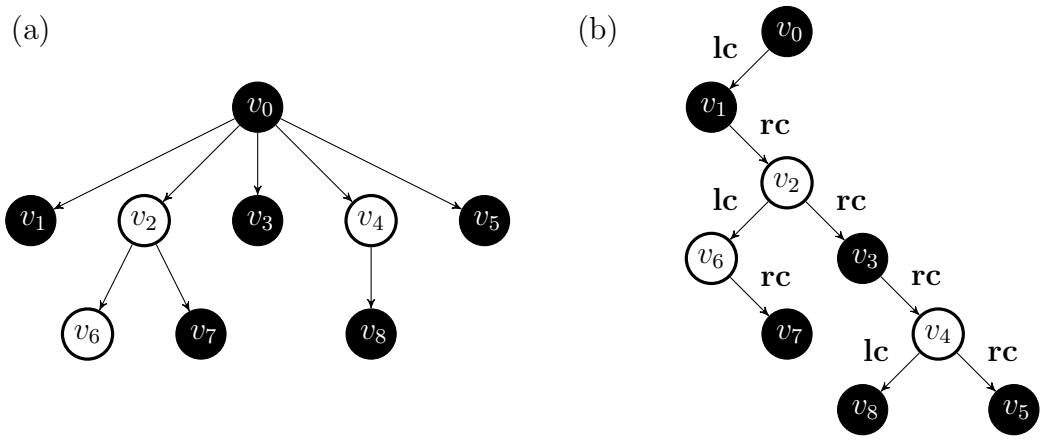

Figure 5.1: (a) Let T be the *ordered* Σ-labeled tree from Example 2.2 for $\Sigma = \{Black, White\}$ where the order of the children of each node is from left to right as depicted in the illustration. (b) The binary version $bin(T)$ of the ordered tree T.

Lemma 5.7. *Every Boolean mDatalog($\tau_{GK,\Sigma}$)-query Q can be rewritten, in linear time, into a Boolean mDatalog($\tau_{b,\Sigma}$)-query Q' which satisfies the following:*

- *For every ordered unranked Σ-labeled tree T we have $Q(T) = \textbf{yes} \iff Q'(bin(T)) = \textbf{yes}$.*
- *For every ordered Σ-labeled binary tree T' with $Q'(T') = \textbf{yes}$ there is an ordered unranked Σ-labeled tree T such that $T' = bin(T)$.*

Furthermore, if the program of Q is in TMNF, then also the program of Q' is in TMNF.

Proof. Let $Q = (\mathcal{P}, P)$. We specify Q' by a monadic datalog program $\mathcal{P}'$ and a query predicate P' as follows: $\mathcal{P}'$ is obtained from $\mathcal{P}$ by renaming in each rule the predicates **fc**, **ns**, **leaf**, and **ls** into the predicates **lc**, **rc**, **has_no_lc**, and **has_no_rc**. Furthermore, we let P' be a new intensional predicate and we add to $\mathcal{P}'$ the rule

$$P'(x) \leftarrow P(x), \textbf{has\_no\_rc}(x).$$

It is straightforward to verify that the resulting Boolean query Q' has the desired properties. □

By combining this lemma with Proposition 5.5, we obtain the following:

Proposition 5.8. *Let Σ be a finite alphabet and let $\Sigma' := \Sigma \times \{0, 1\}$. Within linear time, we can rewrite given unary mDatalog($\tau^{\textbf{child}}_{GK,\Sigma}$)-queries Q_1 and Q_2 (querying ordered Σ-labeled unranked trees) into Boolean mDatalog($\tau_{b,\Sigma'}$)-queries Q'_1 and Q'_2 (querying ordered Σ'-labeled binary trees) such that $Q_1 \subseteq Q_2$ if and only if $Q'_1 \subseteq Q'_2$. Furthermore, the programs of Q'_1 and Q'_2 are in TMNF.*

Proof. We first apply Proposition 5.5 to obtain Boolean mDatalog($\tau_{GK,\Sigma'}$)-queries $\tilde{Q}_1$ and $\tilde{Q}_2$, whose programs are in TMNF, such that $Q_1 \subseteq Q_2$ if and only if $\tilde{Q}_1 \subseteq \tilde{Q}_2$.

Next, we apply Lemma 5.7 to rewrite $\tilde{Q}_1$ and $\tilde{Q}_2$ into Boolean mDatalog($\tau_{b,\Sigma'}$)-queries Q'_1 and Q'_2. It is straightforward to check that $\tilde{Q}_1 \subseteq \tilde{Q}_2$ if and only if $Q'_1 \subseteq Q'_2$:

If $\tilde{Q}_1 \not\subseteq \tilde{Q}_2$, there is an ordered Σ'-labeled unranked tree T such that $\tilde{Q}_1(T) = \mathbf{yes}$ and $\tilde{Q}_2(T) = \mathbf{no}$. By Lemma 5.7 we obtain that $Q'_1(\mathit{bin}(T)) = \mathbf{yes}$ and $Q'_2(\mathit{bin}(T)) = \mathbf{no}$. Thus, $Q'_1 \not\subseteq Q'_2$.

If otherwise $Q'_1 \not\subseteq Q'_2$ then there is an ordered Σ'-labeled binary tree T' such that $Q'_1(T') = \mathbf{yes}$ and $Q'_2(T') = \mathbf{no}$. Since $Q'_1(T') = \mathbf{yes}$, Lemma 5.7 tells us that there is an ordered Σ'-labeled unranked tree T such that $T' = \mathit{bin}(T)$. Furthermore, by Lemma 5.7 we know that $\tilde{Q}_1(T) = \mathbf{yes}$ and $\tilde{Q}_2(T) = \mathbf{no}$. Thus, $\tilde{Q}_1 \not\subseteq \tilde{Q}_2$. □

Proposition 5.8 implies that in order to prove Theorem 5.1 it suffices to show that the following problem can be solved in 1-fold exponential time:

Boolean-TMNF-QCP for mDatalog(τ_b) on binary labeled trees	
Input:	A finite alphabet Σ and two Boolean mDatalog($\tau_{b,\Sigma}$)-queries Q_1 and Q_2 whose programs are in TMNF.
Question:	Is $Q_1 \subseteq Q_2$?

This finishes step (1) of the agenda described in the proof's sketch for Theorem 5.1.

5.3 Nondeterministic Bottom-Up Tree Automata (NBTA)

In this section we recall the classical notion (cf., e.g., [Tho97]) of nondeterministic bottom-up tree automata and show that a Boolean monadic datalog query Q on binary trees can be translated, within 1-fold exponential time, into an NBTA $\mathtt{A}_Q^{\mathbf{no}}$ which accepts exactly those binary trees T for which $Q(T) = \mathbf{no}$.

A *nondeterministic bottom-up tree automaton* (NBTA, for short) $\mathtt{A}$ is specified by a tuple (Σ, S, Δ, F), where Σ is a finite non-empty alphabet, S is a finite set of *states*, $F \subseteq S$ is the set of *accepting states*, and Δ is the *transition relation* with

$$\Delta \quad \subseteq \quad S_\# \times S_\# \times \Sigma \times S, \tag{5.1}$$

where $S_\# := S \cup \{\#\}$ for a symbol $\#$ that does not belong to S.

A *run* of $\mathtt{A}$ on an ordered Σ-labeled binary tree T is a mapping $\rho : V^T \to S$ such that the following is true for all nodes v of T where α denotes the label of v in T:

- If v has no left child and no right child, then $\big(\#, \#, \alpha, \rho(v)\big) \in \Delta$.
- If v has a left child u_ℓ and a right child u_r, then $\big(\rho(u_\ell), \rho(u_r), \alpha, \rho(v)\big) \in \Delta$.
- If v has a left child u_ℓ but no right child, then $\big(\rho(u_\ell), \#, \alpha, \rho(v)\big) \in \Delta$.

- If v has a right child u_r but no left child, then $\big(\#, \rho(u_r), \alpha, \rho(v)\big) \in \Delta$.

A run ρ of $\mathtt{A}$ on T is *accepting* if $\rho(root^T) \in F$ where $root^T$ is the root node of T. The automaton $\mathtt{A}$ *accepts* the tree T if there exists an accepting run of $\mathtt{A}$ on T. A tree T is *rejected* if and only if it is not accepted. The *tree language* $\mathcal{L}(\mathtt{A})$ is the set of all ordered Σ-labeled binary trees T that are accepted by $\mathtt{A}$. A set L of ordered Σ-labeled binary trees is *regular* if $L = \mathcal{L}(\mathtt{A})$ for some NBTA $\mathtt{A}$.

We define the *size* $\|\mathtt{A}\|$ of an NBTA $\mathtt{A}$ to be the length of a reasonable representation of the tuple (Σ, S, Δ, F). In particular, we let $\|\mathtt{A}\| := |\Sigma| + |S| + |\Delta| + |F|$. Note that due to (5.1) we have

$$\|A\| \;=\; O(|S|^3 \cdot |\Sigma|). \tag{5.2}$$

It is well-known that the usual automata constructions for NFAs (i.e., nondeterministic finite automata on words) also apply to NBTAs. For formulating the results needed for our purposes, we introduce the following notation: For finite alphabets Σ and Γ we let $proj_\Sigma$ be the mapping from $\Sigma\times\Gamma$ to Σ with $proj_\Sigma(\alpha, \beta) := \alpha$ for all $(\alpha, \beta) \in \Sigma\times\Gamma$. If T is a $(\Sigma\times\Gamma)$-labeled tree, we write $proj_\Sigma(T)$ to denote the Σ-labeled tree obtained from T by replacing each node label (α, β) by the node label α.

By using standard automata constructions, we obtain:

Proposition 5.9 (Folklore; see e.g. [CDG+08])**.**

Union: *For all NBTAs* $\mathtt{A}_1$ *and* $\mathtt{A}_2$ *over the same alphabet* Σ*, an NBTA* $\mathtt{A}_\cup$ *with* $\mathcal{L}(\mathtt{A}_\cup) = \mathcal{L}(\mathtt{A}_1) \cup \mathcal{L}(\mathtt{A}_2)$ *can be constructed in time linear in* $\|\mathtt{A}_1\|$ *and* $\|\mathtt{A}_2\|$*. Furthermore, if* k_i *is the number of states of* $\mathtt{A}_i$*, for* $i \in \{1, 2\}$*, then the number of states of* $\mathtt{A}_\cup$ *is* $k_1{+}k_2$*.*

Intersection: *For all NBTAs* $\mathtt{A}_1$ *and* $\mathtt{A}_2$ *over the same alphabet* Σ*, an NBTA* $\mathtt{A}_\cap$ *with* $\mathcal{L}(\mathtt{A}_\cap) = \mathcal{L}(\mathtt{A}_1) \cap \mathcal{L}(\mathtt{A}_2)$ *can be constructed in time polynomial in* $\|\mathtt{A}_1\|$ *and* $\|\mathtt{A}_2\|$*. Furthermore, if* k_i *is the number of states of* $\mathtt{A}_i$*, for* $i \in \{1, 2\}$*, then the number of states of* $\mathtt{A}_\cap$ *is* $k_1{\cdot}k_2$*.*

Complementation: *For every NBTA* $\mathtt{A}$*, an NBTA* $\mathtt{A}^c$ *which accepts exactly those trees that are rejected by* $\mathtt{A}$ *can be constructed in time polynomial in* $\|\mathtt{A}\|{\cdot}2^k$*, where* k *denotes the number of states of* $\mathtt{A}$*. Furthermore, the number of states of* $\mathtt{A}^c$ *is* 2^k*.*

Projection: *For every NBTA* $\mathtt{A}$ *over an alphabet of the form* $\Sigma\times\Gamma$*, an NBTA* $\mathtt{A}^p$ *over alphabet* Σ *with* $\mathcal{L}(\mathtt{A}^p) = \{\, proj_\Sigma(T) \;:\; T \in \mathcal{L}(\mathtt{A}) \,\}$ *can be constructed in time polynomial in* $\|\mathtt{A}\|$*. Furthermore, the number of states of* $\mathtt{A}^p$ *is the same as the number of states of* $\mathtt{A}$*.* ⌟

The emptiness problem for NBTAs is defined as follows:

EMPTINESS PROBLEM FOR NBTAS
Input: An NBTA $\mathtt{A} = (\Sigma, S, \Delta, F)$.
Question: Is $\mathcal{L}(\mathtt{A}) = \varnothing$?

Similarly to NFAs, the emptiness problem for NBTAs can be solved efficiently:

Proposition 5.10 (Folklore; see e.g. [CDG+08])**.** *The emptiness problem for NBTAs can be solved in time polynomial in the size of the input automaton.* ⌟

The following result establishes a relation between monadic datalog and NBTAs.

Proposition 5.11. *Let Σ be a finite alphabet and let Q be a Boolean mDatalog($\tau_{b,\Sigma}$)-query whose program is in TMNF. Within time polynomial in $|\Sigma|\cdot 2^{\|Q\|}$ we can construct an NBTA $\mathtt{A}^{\mathbf{no}}$ with $2^{O(\|Q\|)}$ states, which accepts exactly those ordered Σ-labeled binary trees T where $Q(T) = \mathbf{no}$.*

Proof. Our proof proceeds as described in the proof sketch given in the beginning of this chapter. Let $\mathcal{P}$ be the program of Q, let X_1 be the query predicate of Q, and let $X_1, \ldots, X_n$ be the list of all intensional predicates of $\mathcal{P}$.

Step 1: Transform Q into an equivalent monadic second-order sentence φ_Q:
We follow the standard construction (cf., [GK04, Proposition 3.3]) which uses the fact that the result $\mathcal{T}_{\mathcal{P}}^{\omega}(C)$ of a monadic datalog program $\mathcal{P}$ on a set C of atomic facts is the fixed-point of the immediate consequence operator $\mathcal{T}_{\mathcal{P}}$ that contains C:

For any rule r of $\mathcal{P}$ of the form $h^r \leftarrow b_1^r, \ldots, b_m^r$, define the formula

$$\psi_r \;:=\; \forall z_1 \cdots \forall z_\ell \,\big(\, (b_1^r \wedge \cdots \wedge b_m^r) \rightarrow h^r \,\big),$$

where $z_1, \ldots, z_\ell$ is the list of variables appearing in the rule r. As $\mathcal{P}$ is in TMNF, we know that $m = 2$ and $\ell \leqslant 2$. W.l.o.g. we can assume that all rules use variables in $\{z_1, z_2\}$.

Let $SAT(X_1, \ldots, X_n)$ be the conjunction of the formulas ψ_r for all rules r in $\mathcal{P}$, and let

$$\varphi_Q \;:=\; \forall X_1 \cdots \forall X_n \,\big(\, SAT(X_1, \ldots, X_n) \rightarrow X_1(\mathit{root}) \,\big).$$

It is straightforward to verify (see [GK04, Proposition 3.3]) that for any ordered Σ-labeled binary tree T we have $Q(T) = \mathbf{yes}$ if and only if the tree T, expanded by a constant *root* interpreted by the tree's root node, satisfies the MSO-sentence φ_Q.

Clearly, φ_Q is equivalent to $\forall X_1 \cdots \forall X_n \,\big(\, X_1(\mathit{root}) \vee \neg SAT(X_1, \ldots, X_n) \big)$. Furthermore, $\neg SAT$ is equivalent to $\bigvee_{r\in\mathcal{P}} \neg\psi_r$ and $\neg\psi_r$ is equivalent to the formula $\exists z_1 \exists z_2\, (b_1^r \wedge b_2^r \wedge \neg h^r)$, for a TMNF-rule r of the form $h^r \leftarrow b_1^r, b_2^r$. Combined, we obtain that φ_Q is equivalent to the formula

$$\varphi'_Q \;:=\; \forall X_1 \cdots \forall X_n \,\exists z_1 \exists z_2 \,\Big(\, X_1(\mathit{root}) \;\vee\; \bigvee_{r\in\mathcal{P}} \big(b_1^r \wedge b_2^r \wedge \neg h^r\big) \,\Big).$$

Clearly, for any tree T we have $Q(T) = \mathbf{no}$ if and only if T satisfies the formula $\neg\varphi'_Q$, which is equivalent to the formula

$$\tilde{\varphi}_Q \;:=\; \exists X_1 \cdots \exists X_n \,\neg\, \exists z_1 \exists z_2 \,\Big(\, X_1(\mathit{root}) \;\vee\; \bigvee_{r\in\mathcal{P}} \big(b_1^r \wedge b_2^r \wedge \neg h^r\big) \,\Big).$$

Step 2: Transform $\tilde{\varphi}_Q$ into an equivalent NBTA:
We proceed in the same way as in well-known textbook proofs for Büchi's Theorem, respectively, the Theorem by Doner [Don70] as well as Thatcher and Wright [TW68]

stating the equivalence of MSO-definable languages and regular languages of finite words and trees, respectively (cf. e.g. [Tho97, FG06]):

Based on the formula $\tilde{\varphi}_Q$ we give the construction of the desired NBTA $\mathtt{A}^{\mathbf{no}}$ along the composition of the formula.

For the induction base, we have to handle quantifier-free formulas occurring in $\tilde{\varphi}_Q$. For this, we consider trees over alphabet $\Sigma_n := \Sigma \times \Gamma \times \Gamma'$ for $\Gamma := \{0,1\}^n$ and $\Gamma' := \{0,1\}^2$. If a node v has the label $(\alpha, \gamma, \gamma')$, for $\gamma = \gamma_1 \cdots \gamma_n$ and $\gamma' = \gamma'_1\gamma'_2$, we interpret this as the information that v has Σ-label α, belongs to the relation X_i if and only if $\gamma_i = 1$, and is the value of the variable z_j if and only if $\gamma'_j = 1$ (for $i \in \{1, \ldots, n\}$ and $j \in \{1, 2\}$). We will refer to γ'_j (resp., γ_i and α) as the z_j-component (resp., the X_i-component and the Σ-component) of the label.

To check that the values in the z_j-components of a labeling indeed represent a variable assignment, we build for each $j \in \{1, 2\}$ an NBTA $\mathtt{A}_{z_j}$ that accepts exactly those Σ_n-labeled trees where exactly one node carries a label whose z_j-component is 1. For example, the NBTA $\mathtt{A}_{z_2}$ can be chosen as (Σ_n, S, Δ, F) with $S = \{s_0, s_1\}$, $F = \{s_1\}$, and Δ consisting of the transitions

$$(\#, \#, \beta, s_\nu), \;\; (s_0, s_0, \beta, s_\nu), \;\; (s_0, \#, \beta, s_\nu), \;\; (\#, s_0, \beta, s_\nu)$$

for all $\nu \in \{0, 1\}$ and all labels $\beta \in \Sigma \times \Gamma \times \{0, 1\} \times \{\nu\}$, and the transitions

$$(s_1, \#, \beta, s_1), \;\; (\#, s_1, \beta, s_1), \;\; (s_1, s_0, \beta, s_1), \;\; (s_0, s_1, \beta, s_1)$$

for all labels $\beta \in \Sigma \times \Gamma \times \{0, 1\} \times \{0\}$. This automaton performs a bottom-up scan of the tree and remains in state s_0 until it encounters a node whose label has a 1 in its z_2-component. The latter induces a change into state s_1. The automaton gets stuck (i.e., no run exists) if it is in state s_1 and encounters another node whose label has a 1 in its z_2-component.

To check whether an atomic or negated atomic formula χ (occurring in $\tilde{\varphi}_Q$) is satisfied by an input tree, we build an NBTA $\mathtt{A}_\chi$ that accepts an input tree T if and only if T contains for each variable z_j occurring in χ a node v_j whose z_j-component is 1 such that the nodes v_j satisfy χ. If χ involves a unary atom, this can be achieved in a straightforward way using an automaton with 2 states. If χ is a *binary* atom, this is not difficult either. For example, if $\chi = \mathbf{lc}(z_2, z_1)$, the NBTA $\mathtt{A}_\chi$ performs a bottom-up scan of the tree and remains in state s_0 until it encounters a node v_1 whose z_1-component is labeled 1. The latter induces a change into state s_1. From there on, the automaton either gets stuck or it sees that v_1 is the left child of a node v_2 whose z_2-component is one. The latter induces a change into an accepting state s_2 which is propagated to the root.

Note that each of the NBTAs constructed so far has at most three states and, according to (5.2), has size $O(3^3 \cdot |\Sigma_n|) = O(|\Sigma_n|)$.

The formula $\tilde{\varphi}_Q$ contains a conjunction ζ_r of the form $(b_1^r \wedge b_2^r \wedge \neg h^r)$ for each rule $r \in \mathcal{P}$. We already have constructed NBTAs $\mathtt{A}_{b_1^r}$, $\mathtt{A}_{b_2^r}$, $\mathtt{A}_{\neg h^r}$, $\mathtt{A}_{z_1}$, and $\mathtt{A}_{z_2}$, each of which has at most three states and size $O(|\Sigma_n|)$. By using the intersection construction mentioned in Proposition 5.9, we can build the intersection automaton $\mathtt{A}_{\zeta_r}$ of these five NBTAs. This can be achieved in time polynomial in $O(|\Sigma_n|)$ and the resulting automaton has at most 3^5 states and thus, due to (5.2), has size $O(|\Sigma_n|)$.

The quantifier-free part of the formula $\tilde{\varphi}_Q$ is the disjunction of the formula $X_1(\mathit{root})$ and the formulas ζ_r, for all $r \in \mathcal{P}$. We already have available NBTAs $\mathtt{A}_{X_1(\mathit{root})}$ and $\mathtt{A}_{\zeta_r}$ for

each $r \in \mathcal{P}$. Using the union construction mentioned in Proposition 5.9, we can build the union automaton $\mathtt{A}_{qf}$ of these automata. This can be achieved in time polynomial in $O(|\mathcal{P}|\cdot|\Sigma_n|)$ and the resulting automaton has at most $(|\mathcal{P}|+1)\cdot 3^5 = O(|\mathcal{P}|)$ states and thus, due to (5.2), has size $O(|\mathcal{P}|^3\cdot|\Sigma_n|)$.

Note that $\mathtt{A}_{qf}$ is an NBTA over the alphabet $\Sigma\times\Gamma\times\Gamma'$. We now use the projection construction mentioned in Proposition 5.9 to build an NBTA $\mathtt{A}_{\exists z_1\exists z_2}$ accepting the set of all trees of the form $proj_{\Sigma\times\Gamma}(T)$ for T accepted by $\mathtt{A}_{qf}$. The resulting automaton has $O(|\mathcal{P}|)$ states as $\mathtt{A}_{qf}$, has size $O(|\mathcal{P}|^3\cdot|\Sigma\times\Gamma|) = O(|\mathcal{P}|^3\cdot|\Sigma|\cdot 2^n)$, and can be constructed in time polynomial in $O(|\mathcal{P}|^3\cdot|\Sigma_n|)$.

Next, we use the complementation construction mentioned in Proposition 5.9 to build an NBTA $\mathtt{A}_\neg$ which accepts exactly those trees that are rejected by $\mathtt{A}_{\exists z_1\exists z_2}$. The automaton $\mathtt{A}_\neg$ has $2^{O(|\mathcal{P}|)}$ states and thus, due to (5.2), size $O(2^{O(|\mathcal{P}|)}\cdot|\Sigma\times\Gamma|) = O(2^{O(|\mathcal{P}|)}\cdot|\Sigma|\cdot 2^n)$. It can be constructed in time polynomial in the size of $\mathtt{A}_{\exists z_1\exists z_2}$ and $2^{O(|\mathcal{P}|)}$, i.e., polynomial in $2^{O(|\mathcal{P}|)}\cdot|\mathcal{P}|^3\cdot|\Sigma|\cdot 2^n$

Finally, we use the projection construction mentioned in Proposition 5.9 to build an NBTA $\mathtt{A}^{\mathbf{no}}$ accepting the set of all trees of the form $proj_\Sigma(T)$ for T accepted by $\mathtt{A}_\neg$. The resulting automaton has the same number of states as $\mathtt{A}_\neg$, i.e., $2^{O(|\mathcal{P}|)}$ and can be constructed in time polynomial in the size of $\mathtt{A}_\neg$, i.e., polynomial in $2^{O(|\mathcal{P}|)}\cdot|\Sigma|\cdot 2^n = |\Sigma|\cdot 2^{n+O(|\mathcal{P}|)} = |\Sigma|\cdot 2^{O(\|Q\|)}$.

It is straightforward to verify that the NBTA $\mathtt{A}^{\mathbf{no}}$ accepts exactly those Σ-labeled trees T that satisfy the formula $\tilde{\varphi}_Q$, i.e., those trees T with $Q(T) = \mathbf{no}$. The entire construction of the automaton $\mathtt{A}^{\mathbf{no}}$ took time polynomial in $|\Sigma|\cdot 2^{\|Q\|}$. This completes the proof of Proposition 5.11. □

This concludes the construction of the automaton $\mathtt{A}^{\mathbf{no}}$ of step (2) of the agenda presented in the proof's sketch above. By applying to $\mathtt{A}^{\mathbf{no}}$ the complementation construction mentioned in Proposition 5.9, we obtain an NBTA $\mathtt{A}^{\mathbf{yes}}$ which accepts exactly the Σ-labeled trees T with $Q(T) = \mathbf{yes}$. However, the number of states of $\mathtt{A}^{\mathbf{no}}$ is $2^{O(\|Q\|)}$, and hence the construction of $\mathtt{A}^{\mathbf{yes}}$ takes time polynomial in $\|\mathtt{A}^{\mathbf{no}}\|\cdot 2^{2^{O(\|Q\|)}}$, which is 2-fold exponential in the size of the query Q.

To construct an NBTA equivalent to $\mathtt{A}^{\mathbf{yes}}$ within 1-fold exponential time, we use a different automata model, which is described in the next section.

5.4 Two-Way Alternating Tree Automata (2ATA)

In this section we recall the notion (cf., e.g., [CGKV88, Var98, MFS10]) of two-way alternating tree automata (2ATA) and show that a Boolean monadic datalog query Q on binary trees can be translated, within polynomial time, into a 2ATA $\hat{\mathtt{A}}^{\mathbf{yes}}$ which accepts exactly those binary trees T for which $Q(T) = \mathbf{yes}$. The following definitions concerning 2ATAs are basically taken from [CGKV88, Var98].

For navigating in a binary tree T we consider the operations *up*, *stay*, *left*, *right*. They are viewed as functions from $V^T_\perp$ to $V^T_\perp$ where $V^T_\perp = V^T \cup \{\perp\}$ for the node set V^T of T and a symbol $\perp$ not in V^T. Each of the operations in $Op := \{up, stay, left, right\}$ maps $\perp$ to $\perp$. Furthermore, for each node v of T, we have $stay(v) = v$ while $up(v)$ is the parent

of v in T (resp. $\perp$ if v is the root of T), *left*(v) is the left child of v in T (resp. $\perp$ if v has no left child), and *right*(v) is the right child of v in T (resp. $\perp$ if v has no right child).

A *two-way alternating tree automaton* (2ATA, for short) $\hat{\mathtt{A}}$ is specified by a tuple $(\Sigma, S, s_0, \delta, F)$, where

- Σ is a finite non-empty alphabet,
- S is a finite set of *states*,
- $s_0 \in S$ is the *initial state*,
- $F \subseteq S$ is the set of *accepting states*, and
- $\delta : S \times \Sigma \to \mathcal{B}^+(S \times Op)$ is the *transition function*.

As input, $\hat{\mathtt{A}}$ receives a Σ-labeled binary tree T. It starts in the initial state s_0 at T's root node. Whenever $\hat{\mathtt{A}}$ is in a state $s \in S$ and currently visits a node v of T of label $\alpha \in \Sigma$, it can either choose to stop its computation or to perform a further step in which the formula $\theta := \delta(s, \alpha)$ determines what is done next: The automaton nondeterministically guesses a satisfying assignment for θ, i.e., a set $\{ (s_1, o_1), \ldots, (s_k, o_k) \}$ (for some $k \geqslant 1$) which satisfies θ. Then, it starts k independent copies of $\hat{\mathtt{A}}$, namely a copy which starts in state s_i at node $o_i(v)$, for each $i \in \{1, \ldots, k\}$. In case that $o_i(v) = \perp$, the according automaton stops. The acceptance condition demands that for every situation (s, v) in which the automaton stops, s must be an accepting state.

This can be formalised by the following notion of a *run* R that is a labeled tree where the label (s, o, v) of a node w of R denotes a transition into state s via the operation o onto node v.

A *run* of $\hat{\mathtt{A}}$ on a Σ-labeled binary tree T is a finite unordered *unranked* Γ-labeled tree R, for $\Gamma := S \times Op \times V_\perp^T$ which satisfies the following conditions:

(1) The root of R is labeled with $(s_0, \mathit{stay}, \mathit{root}^T)$, where s_0 is the initial state and root^T is the root of T.

(2) If w is a node of R that is labeled (s, o, v) with $v = \perp$, then w is a *leaf* of R.

(3) If w is a node of R that is labeled (s, o, v) such that v is a node of T and w' is a child of w in R that is labeled (s', o', v'), then $v' = o'(v)$.

(4) If w is a node of R that is labeled (s, o, v) such that v is a node of T labeled $\alpha \in \Sigma$ and w has exactly k children labeled $(s_1, o_1, v_1), \ldots, (s_k, o_k, v_k)$, then the formula $\theta := \delta(s, \alpha)$ is satisfied by the set $\{ (s_1, o_1), \ldots, (s_k, o_k) \}$.

A run R of $\hat{\mathtt{A}}$ on T is *accepting* if every leaf of R is labeled with an accepting state, i.e.: whenever (s, o, v) is the label of a leaf of R, we have $s \in F$. The automaton $\hat{\mathtt{A}}$ *accepts* the tree T if there exists an accepting run of $\hat{\mathtt{A}}$ on T. The *tree language* $\mathcal{L}(\hat{\mathtt{A}})$ is the set of all ordered Σ-labeled binary trees T that are accepted by $\hat{\mathtt{A}}$.

The *size* $\|\hat{\mathtt{A}}\|$ of a 2ATA $\hat{\mathtt{A}}$ is defined as the length of a reasonable representation of the tuple $(\Sigma, S, S_0, \delta, F)$.

Example 5.12. *Figure 5.2 illustrates the operation method of a 2ATA.* ⌟

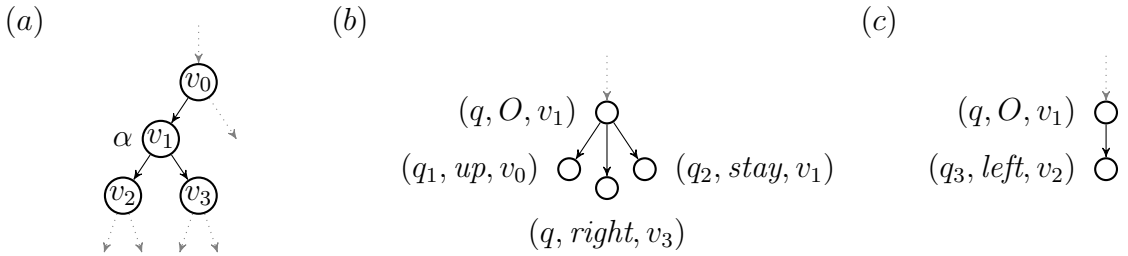

Figure 5.2: Let (a) be a part of an input tree T of a 2ATA $\mathtt{A}$. We assume the actual position during a run of $\mathtt{A}$ on T is labeled with (q, O, v_1) for $O \in \{\mathit{left}, \mathit{right}, \mathit{up}, \mathit{stay}\}$, meaning the automaton has reached node v_1, its state is q. It reads in the next step the label α at node v_1. We assume that $\delta(q, \alpha) = (((q_1, \mathit{up}) \wedge (q, \mathit{right}) \wedge (q_2, \mathit{stay})) \vee (q_3, \mathit{left}))$. Now, there are two possibilities to proceed the next step. (b) The first is to extend the run tree by three new children on the actual position. For the first child a new instance of the automaton starts in state q_1 at $v_0 \in T$ since the automaton moves *up* on the input tree and therefore the label of the first child is (q_1, up, v_0). A second instance moves to the right child v_3 of v_1 an continues the run from the (q, right, v_3)-labeled node in state q, whereas the last instance stays on the input tree at node v_1, switches to q_2 and continues the run starting from the third child. (c) The second possibility is to continue the run with one new child and an instance reading in state q_3 the label of $v_2 \in T$ that is the left child of v_1.

It is known that 2ATAs accept exactly the same tree languages as NBTAs, i.e., the *regular* tree languages. Furthermore, there is a 1-fold exponential algorithm that translates a 2ATA into an equivalent NBTA:

Theorem 5.13 (Cosmadakis et al. [CGKV88])**.** *For every 2ATA* $\hat{\mathtt{A}}$*, an NBTA* $\mathtt{A}$ *with* $\mathcal{L}(\mathtt{A}) = \mathcal{L}(\hat{\mathtt{A}})$ *can be constructed within time 1-fold exponential in* $\|\hat{\mathtt{A}}\|$*.* ⌟

To be precise, [CGKV88] formulated the theorem not in terms of the running time but only in terms of the *size* of the generated NBTA. A proof sketch of the theorem can be found in [CGKV88]; detailed proofs of more general results can be found in [Var98, MFS10].

Our next goal is to construct a polynomial-time algorithm which translates a Boolean monadic datalog query Q in TMNF into an equivalent 2ATA $\hat{\mathtt{A}}$ which accepts exactly those trees T with $Q(T) = \mathbf{yes}$.

To construct such a 2ATA, we will exploit the striking similarity between runs of 2ATAs and *proof trees* characterizing the semantics of datalog. For constructing the desired 2ATA, the following observation will be very convenient:

Let Q be a Boolean mDatalog($\tau_{b,\Sigma}$)-query whose program is in TMNF. Let $\mathcal{P}$ and P be the program and the query predicate of Q, respectively. For a Σ-labeled binary tree T with root node root^T we have $Q(T) = \mathbf{yes}$ if and only if there exists a proof tree PT for the fact $P(\mathit{root}^T)$ such that the leaves of the proof tree are labeled with facts in $\mathit{atoms}(\mathcal{S}_b(T))$. Note that for the particular case of TMNF-programs such a proof tree PT has the following properties:

- The *root* of PT is labeled with the atomic fact $P(root^T)$.
- Each *leaf* of PT is labeled with an atomic fact of one of the following forms:
  - $\mathbf{label}_\alpha(v)$ where $\alpha \in \Sigma$ and v is a node of T labeled α,
  - $\mathbf{root}(root^T)$, where $root^T$ is the root of T,
  - $\mathbf{has\_no\_lc}(v)$ (resp., $\mathbf{has\_no\_rc}(v)$), where v is a node of T that has no left child (resp., has no right child)
  - $\mathbf{lc}(v_1, v_2)$ (resp., $\mathbf{rc}(v_1, v_2)$), where v_2 is the left (resp., right) child of v_1 in T.
- Each non-leaf node of PT is labeled with a fact $X(v)$ where v is a node of T and $X \in \text{idb}(\mathcal{P})$.
- Every non-leaf node w of PT has exactly two children w_1 and w_2. If w is labeled by an atomic fact $X(v)$, then $\mathcal{P}$ contains a rule r whose head is of the form $X(x)$ and the following is true:
  (a) If the body of r is of the form $Y(x), Z(x)$, then w_1 is labeled $Y(v)$ and w_2 is labeled $Z(v)$.
  (b) If the body of r is of the form $\mathbf{lc}(x, y), Y(y)$, then node v of T has a left child v' and in PT the nodes w_1 and w_2 are labeled with the facts $\mathbf{lc}(v, v')$ and $Y(v')$.
  Accordingly, if the body of r is of the form $\mathbf{rc}(x, y), Y(y)$, then node v of T has a right child v' and in PT the nodes w_1 and w_2 are labeled with the facts $\mathbf{rc}(v, v')$ and $Y(v')$.
  (c) If the body of r is of the form $\mathbf{lc}(y, x), Y(y)$, then node v of T is the left child of its parent v' and in PT the nodes w_1 and w_2 are labeled with the facts $\mathbf{lc}(v', v)$ and $Y(v')$.
  Accordingly, if the body of r is of the form $\mathbf{rc}(y, x), Y(y)$, then node v of T is the right child of its parent v', and in PT the nodes w_1 and w_2 are labeled with the facts $\mathbf{rc}(v', v)$ and $Y(v')$.

We will build a 2ATA for which an accepting run R on an input tree T precisely corresponds to a proof tree PT for the fact $P(root^T)$. To better cope with technical details in the automaton construction, we will consider automata which receive input trees that are labeled by the extended alphabet $\hat{\Sigma}$, with

$$\hat{\Sigma} \;:=\; \Sigma \times 2^{\{\,\mathbf{root},\ \mathbf{has\_no\_lc},\ \mathbf{has\_no\_rc},\ \mathbf{is\_lc},\ \mathbf{is\_rc}\,\}}.$$

With every Σ-labeled binary tree T we associate a $\hat{\Sigma}$-labeled binary tree $\hat{T}$ that is obtained from T by replacing the label of each node v labeled $\alpha \in \Sigma$ with the label (α, I) where $I \subseteq \{\,\mathbf{root},\ \mathbf{has\_no\_lc},\ \mathbf{has\_no\_rc},\ \mathbf{is\_lc},\ \mathbf{is\_rc}\,\}$ is given as follows:

$$\begin{aligned}
\mathbf{root} \in I &\iff v \text{ is the root of } T,\\
\mathbf{has_no_lc} \in I &\iff v \text{ is a node of } T \text{ that has no left child},\\
\mathbf{has_no_rc} \in I &\iff v \text{ is a node of } T \text{ that has no right child},\\
\mathbf{is_lc} \in I &\iff v \text{ is the left child of its parent } v' \text{ in } T,\\
\mathbf{is_rc} \in I &\iff v \text{ is the right child of its parent } v' \text{ in } T.
\end{aligned}$$

We are now ready for this section's key result:

Proposition 5.14. *Let Σ be a finite alphabet and let Q be a Boolean mDatalog($\tau_{b,\Sigma}$)-query whose program is in TMNF. Within time polynomial in the size of Q and Σ, we can construct a 2ATA $\hat{\mathtt{A}}$ such that for all Σ-labeled binary trees T, the automaton $\hat{\mathtt{A}}$ accepts the tree $\hat{T}$ if, and only if, $Q(T) = \mathbf{yes}$.*

Proof. Let $\mathcal{P}$ and P be the program and the query predicate of Q. We construct the automaton $\hat{\mathtt{A}}$ in such a way that a proof tree PT for the fact $P(root^T)$ can easily be turned into an accepting run of $\hat{\mathtt{A}}$ on $\hat{T}$ (and vice versa).

The state set S of $\hat{\mathtt{A}} = (\hat{\Sigma}, S, s_0, \delta, F)$ is chosen as the set of all intensional predicates of $\mathcal{P}$, all unary relation symbols in $\tau_{b,\Sigma}$, and additionally, we use states called **is_lc**, **is_rc**, **accept**, and **reject**. Thus,

$$\begin{aligned} S \;=\; & \{\mathbf{accept},\ \mathbf{reject}\} \;\cup\; \mathrm{idb}(\mathcal{P}) \;\cup \\ & \{\mathbf{label}_\alpha \,:\, \alpha \in \Sigma\} \;\cup\; \{\,\mathbf{root},\ \mathbf{has\_no\_lc},\ \mathbf{has\_no\_rc},\ \mathbf{is\_lc},\ \mathbf{is\_rc}\,\}. \end{aligned}$$

The query predicate P is the initial state and **accept** is the only accepting state. I.e., $s_0 := P$ and $F := \{\mathbf{accept}\}$.
The transition function $\delta : S \times \hat{\Sigma} \to \mathcal{B}^+(S \times Op)$ is chosen as follows:
Let $\beta = (\alpha, I)$ be an arbitrary letter in $\hat{\Sigma}$. We let

$$\delta(\mathbf{accept}, \beta) := (\mathbf{accept}, \mathit{stay}) \qquad \text{and} \qquad \delta(\mathbf{reject}, \beta) := (\mathbf{reject}, \mathit{stay}).$$

For every $\alpha' \in \Sigma$ we let

$$\delta(\mathbf{label}_{\alpha'}, \beta) := \begin{cases} (\mathbf{accept}, \mathit{stay}) & \text{if } \alpha' = \alpha \\ (\mathbf{reject}, \mathit{stay}) & \text{otherwise.} \end{cases}$$

For every $X \in \{\,\mathbf{root},\ \mathbf{has\_no\_lc},\ \mathbf{has\_no\_rc},\ \mathbf{is\_lc},\ \mathbf{is\_rc}\,\}$ we let

$$\delta(X, \beta) := \begin{cases} (\mathbf{accept}, \mathit{stay}) & \text{if } X \in I \\ (\mathbf{reject}, \mathit{stay}) & \text{otherwise.} \end{cases}$$

For the case that $X \in \mathrm{idb}(\mathcal{P})$, the formula $\delta(X, \beta)$ is specified as follows. We let $\mathcal{P}_X$ be the set of all rules of $\mathcal{P}$ whose head is of the form $X(x)$ and we choose

$$\delta(X, \beta) := \bigvee_{r \in \mathcal{P}_X} \theta_r,$$

where the formula $\theta_r \in \mathcal{B}^+(S \times Op)$ is chosen as indicated in the following table:

rule r of the form	conjunction θ_r
$X(x) \leftarrow Y(x), Z(x)$	$(Y, \mathit{stay}) \ \wedge\ (Z, \mathit{stay})$
$X(x) \leftarrow \mathbf{lc}(x, y), Y(y)$	$(\mathbf{is\_lc}, \mathit{left}) \ \wedge\ (Y, \mathit{left})$
$X(x) \leftarrow \mathbf{rc}(x, y), Y(y)$	$(\mathbf{is\_rc}, \mathit{right}) \ \wedge\ (Y, \mathit{right})$
$X(x) \leftarrow \mathbf{lc}(y, x), Y(y)$	$(\mathbf{is\_lc}, \mathit{stay}) \ \wedge\ (Y, \mathit{up})$
$X(x) \leftarrow \mathbf{rc}(y, x), Y(y)$	$(\mathbf{is\_rc}, \mathit{stay}) \ \wedge\ (Y, \mathit{up})$

Clearly, this automaton $\hat{\mathtt{A}}$ can be constructed in time polynomial in the size of Σ and Q. Indeed, it remains to verify that for any Σ-labeled binary tree T we have $Q(T) = \mathbf{yes} \iff \hat{\mathtt{A}}$ accepts $\hat{T}$.

For the "$\Longrightarrow$"-direction, let PT be a proof tree for the the fact $P(root^T)$. We can transform PT into a run R of $\hat{\mathtt{A}}$ on $\hat{T}$ as follows: Assign the new label $(P, \mathit{stay}, root^T)$ to the root node of PT. For each non-leaf node w of PT note that w is originally labeled by an atomic fact $X(v)$ with $X \in \mathrm{idb}(\mathcal{P})$ and w has exactly two children w_1, w_2 in PT.

(a) If w_1 and w_2 are labeled $Y(v)$ and $Z(v)$, respectively, then assign to node w_1 the new label (Y, stay, v) and to node w_2 the new label (Z, stay, v).

(b) If w_1 and w_2 are labeled $\mathbf{lc}(v, v')$ and $Y(v')$, respectively, then assign to node w_1 the new label $(\mathbf{is\_lc}, \mathit{left}, v')$ and to node w_2 the new label (Y, left, v'). Furthermore, we add to w_1 a new child labeled $(\mathbf{accept}, \mathit{stay}, v)$.

We proceed analogously in the case that w_1, w_2 are labeled $\mathbf{rc}(v, v'), Y(v')$.

(c) If w_1 and w_2 are labeled $\mathbf{lc}(v', v)$ and $Y(v')$, respectively, then assign to w_1 the new label $(\mathbf{is\_lc}, \mathit{stay}, v)$, and to node w_2 the new label (Y, up, v'). Furthermore, we add to w_1 a new child labeled $(\mathbf{accept}, \mathit{stay}, v)$.

We proceed analogously in the case that w_1, w_2 are labeled $\mathbf{rc}(v', v), Y(v')$.

Finally, for each leaf w of PT that was originally labeled $X(v)$ for an $X \in \{\mathbf{root}, \mathbf{has\_no\_lc}, \mathbf{has\_no\_rc}\} \cup \{\mathbf{label}_\alpha : \alpha \in \Sigma\}$, we add a new child w_1 that receives the new label $(\mathbf{accept}, \mathit{stay}, v)$.

It is straightforward to verify that the obtained tree R is an accepting run of $\hat{\mathtt{A}}$ on $\hat{T}$.

For the direction "$\Longleftarrow$", let R be an accepting run of $\hat{\mathtt{A}}$ on $\hat{T}$. Along the definition of δ it is straightforward to see that we can assume w.l.o.g. that each node of R has at most two children.

The run R can be turned into a proof tree PT for the fact $P(root^T)$ (i.e., witnessing that $Q(T) = \mathbf{yes}$) as follows: Consider each node w of R, and let (s, o, v) be the label of node w.

Since R is an accepting run and $\mathbf{accept}$ is the only accepting state, we know by the construction of δ that $s \neq \mathbf{reject}$ and that $v \neq \bot$ if $s \neq \mathbf{accept}$. In case that $s \in \tau_{b,\Sigma} \cup \mathrm{idb}(\mathcal{P})$, we assign to w the new label "$s(v)$".

In the case that $s = \mathbf{label}_{\alpha'}$ for an $\alpha' \in \Sigma$, we know by the construction of δ and the fact that R is an accepting run that node w has a unique child w_1 in R and this node w_1 is labeled with $(\mathbf{accept}, \mathit{stay}, v)$. Furthermore, we know by the construction of δ that $\alpha' = \alpha$ where $\beta = (\alpha, I)$ is the label of node v in $\hat{T}$. Thus, the statement "$\mathbf{label}_{\alpha'}(v)$" is true for node v in T. Hence, we delete the node w_1 (and all nodes in the subtree rooted at w_1).

In the case that $s \in \{\mathbf{root}, \mathbf{has\_no\_lc}, \mathbf{has\_no\_rc}, \mathbf{is\_lc}, \mathbf{is\_rc}\}$, we know by the construction of δ and the fact that R is an accepting run that node w has a unique child w_1 in R and this node w_1 is labeled with $(\mathbf{accept}, \mathit{stay}, v)$. Furthermore, we know by the construction of δ that $s \in I$, where $\beta = (\alpha, I)$ is the label of node v in $\hat{T}$. Thus, the statement "$s(v)$" is true for node v in T. Hence, we delete the node w_1 (and all nodes in the subtree rooted at w_1).

In case that $s \in \{\mathbf{root}, \mathbf{has\_no\_lc}, \mathbf{has\_no\_rc}\}$, the node w then is a leaf labeled with an atomic fact "$s(v)$" that is true in T.
In case that $s = \mathbf{is\_lc}$, the statement "$\mathbf{is\_lc}(v)$" is a true statement, but it is not suitable as label in a proof tree since the predicate **is_lc** does not belong to the schema $\tau_{b,\Sigma}$. Therefore, we replace the label "$\mathbf{is\_lc}(v)$" by the label "$\mathbf{lc}(v', v)$" where v' is the parent of v in T. We proceed analogously in case that $s = \mathbf{is\_rc}$.

It is straightforward to verify that the obtained tree PT is a proof tree for $P(root^T)$. This completes the proof of Proposition 5.14. □

Finally, we are ready for establishing the second part of step (2) of the agenda described in the proof's sketch for Theorem 5.1.

Proposition 5.15. *Let Σ be a finite alphabet and let Q be a Boolean mDatalog($\tau_{b,\Sigma}$)-query whose program is in TMNF. Within time 1-fold exponential in the size of Q and Σ, we can construct an NBTA $\mathtt{A}^{\mathbf{yes}}$, which accepts exactly those ordered Σ-labeled binary trees T where $Q(T) = \mathbf{yes}$.*

Proof. First, we use Proposition 5.14 to construct, within polynomial time, a 2ATA $\hat{\mathtt{A}}$ such that for all ordered binary Σ-labeled trees T, the automaton $\hat{A}$ accepts the $\hat{\Sigma}$-labeled tree $\hat{T}$ if and only if $Q(T) = \mathbf{yes}$.

Now, we use Theorem 5.13 to construct, within time 1-fold exponential in $\|\hat{\mathtt{A}}\|$ (i.e., 1-fold exponential in the size of Q and Σ), an NBTA $\mathtt{A}$ with $\mathcal{L}(\mathtt{A}) = \mathcal{L}(\hat{\mathtt{A}})$.

Note that $\mathtt{A}$ operates on $\hat{\Sigma}$-labeled trees, while we are looking for an NBTA $\mathtt{A}^{\mathbf{yes}}$ operating on Σ-labeled trees. To obtain such an automaton, we proceed as follows:

Let $\mathtt{B}$ be an NBTA of alphabet $\hat{\Sigma}$ which accepts exactly those $\hat{\Sigma}$-labeled trees T' for which there exists a Σ-labeled tree T such that $T' = \hat{T}$. Building such an NBTA is straightforward: the automaton just needs to check that the $\hat{\Sigma}$-labels correctly identify the root node, the nodes that are left (right) children, and the nodes that have no left (right) child.

Using the intersection construction mentioned in Proposition 5.9, we can build the intersection automaton $\mathtt{A}'$ of $\mathtt{B}$ and $\mathtt{A}$. Hence, $\mathtt{A}'$ accepts a $\hat{\Sigma}$-labeled tree T' if and only if there exists a Σ-labeled tree T such that $T' = \hat{T}$ and $\hat{T}$ is accepted by $\mathtt{A}$.

Finally, we use the projection construction described in Proposition 5.9 to obtain an NBTA $\mathtt{A}^{\mathbf{yes}}$ over alphabet Σ such that $\mathcal{L}(\mathtt{A}^{\mathbf{yes}}) = \{proj_\Sigma(T') : T' \in \mathcal{L}(\mathtt{A}')\}$. Thus,

$$\begin{aligned} \mathtt{A}^{\mathbf{yes}} \text{ accepts a tree } T \quad &\Longleftrightarrow \quad \hat{T} \text{ is accepted by } \mathtt{A} \\ &\Longleftrightarrow \quad Q(T) = \mathbf{yes}. \end{aligned}$$

As the intersection and projection constructions can be performed within time polynomial in the size of its input NBTAs, the entire construction of $\mathtt{A}^{\mathbf{yes}}$ takes time at most 1-fold exponential in the size of Q and Σ. □

5.5 Finishing the Proof of Theorem 5.1

As noted on page 69 we show the following proposition which leads to a short proof of the original query containment problem in Theorem 5.1.

Proposition 5.16. *The Boolean-TMNF-QCP for mDatalog(τ_b) on binary labeled trees belongs to* EXPTIME.

Proof. Let Σ and the Boolean mDatalog($\tau_{b,\Sigma}$)-queries Q_1 and Q_2 in TMNF be the input of the Boolean-TMNF-QCP.

By using Proposition 5.15, we can construct, within time 1-fold exponential in the size of Q_1 and Σ, an NBTA $\mathtt{A}_1^{\mathbf{yes}}$ which accepts exactly those Σ-labeled binary trees T where $Q_1(T) = \mathbf{yes}$.

By using Proposition 5.11, we can construct, within time 1-fold exponential in the size of Q_2 and Σ, an NBTA $\mathtt{A}_2^{\mathbf{no}}$ which accepts exactly those Σ-labeled binary trees T where $Q_2(T) = \mathbf{no}$.

Now, we use the intersection construction mentioned in Proposition 5.9 to build the intersection automaton $\mathtt{B}$ of $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$. Clearly, $\mathtt{B}$ accepts a Σ-labeled binary tree T if and only if $Q_1(T) = \mathbf{yes}$ and $Q_2(T) = \mathbf{no}$.

Finally, we use the emptiness test provided by Proposition 5.10 to check whether $\mathcal{L}(\mathtt{B})$ is the empty language. Clearly, this is the case if and only if $Q_1 \subseteq Q_2$.

Since the intersection construction and the emptiness test take only time polynomial in the size of the input automata, the entire algorithm for checking whether $Q_1 \subseteq Q_2$ runs in time 1-fold exponential in the size of Σ, Q_1, and Q_2. □

Proof of Theorem 5.1:
Our goal is to show that the QCP for unary mDatalog($\tau_{GK}^{\mathbf{child}}$) on unranked ordered labeled trees belongs to EXPTIME.

Let Σ, Q_1, and Q_2 be an input for the QCP. Let $\Sigma' := \Sigma \times \{0, 1\}$. By using Proposition 5.8 we obtain, within linear time, Boolean mDatalog($\tau_{b,\Sigma'}$)-queries Q_1' and Q_2' such that $Q_1 \subseteq Q_2$ if and only if $Q_1' \subseteq Q_2'$ and the programs of Q_1' and Q_2' are in TMNF.

It remains to decide the Boolean-TMNF-QCP on input Σ', Q_1', and Q_2' in 1-fold exponential time which can be done by Proposition 5.16.

Thus, the algorithm for checking whether $Q_1 \subseteq Q_2$ runs in time 1-fold exponential in the size of Σ, Q_1, and Q_2. This completes the proof of Theorem 5.1 □

5.6 Consequences of Theorem 5.1

This subsection's goal is to transfer the result of Theorem 5.1 to the case of ranked trees. In particular, we show the following.

Corollary 5.17.

(a) The query containment problem for unary mDatalog($\tau_{GK}^{\mathbf{child}}$) on ranked ordered labeled trees belongs to EXPTIME.

(b) The query containment problem for unary mDatalog($\tau_u^{\mathbf{root,leaf}}$) on ranked unordered labeled trees belongs to EXPTIME.

Proof. Assume (a) is true then (b) follows immediately since $\tau_u^{\mathbf{root,leaf}} \subseteq \tau_{GK}^{\mathbf{child}}$.

Let the ranked alphabet $\Sigma(\sigma, ar)$ and the mDatalog($\tau_{GK,\Sigma}^{\mathbf{child}}$)-queries $Q_1 = (\mathcal{P}_1, P_1)$ and $Q_2 = (\mathcal{P}_2, P_2)$ be the input for the QCP on ranked ordered labeled trees. It is to decide whether for every ranked ordered Σ-labeled tree T holds $Q_1(T) \subseteq Q_2(T)$.

By Lemma 3.22 we know there is an mDatalog($\tau^{\mathbf{child}}_{GK,\Sigma}$)-query $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ such that the query predicate P_{rk} yields true for every node of an unranked ordered σ-labeled tree T if and only if T respects the ranked alphabet that means there is a ranked ordered Σ-labeled tree T' such that $S^M_o(T) = S^M_o(T')$. If there is not such a ranked tree the query predicate is false for every node of T.

W.l.o.g. we have $\text{idb}(Q_1) \cap \mathcal{P}_{rk} = \text{idb}(Q_2) \cap \mathcal{P}_{rk} = \varnothing$. Now, for the given query Q_1 we construct the query $Q'_1 = (\mathcal{P}'_1, P'_1)$ where

$$\mathcal{P}'_1 = \mathcal{P}_1 \cup \mathcal{P}_{rk} \cup \{\ P'_1 \ \leftarrow \ P_1(x), P_{rk}(x)\ \}$$

and analog for the given query Q_2 construct the query $Q'_2 = (\mathcal{P}'_2, P'_2)$ where

$$\mathcal{P}'_2 = \mathcal{P}_2 \cup \mathcal{P}_{rk} \cup \{\ P'_2 \ \leftarrow \ P_2(x), P_{rk}(x)\ \}.$$

By using the algorithm provided by Theorem 5.1, we return **yes** to the $QCP(\Sigma, Q_1, Q_2)$ on ranked ordered Σ-labeled trees if and only if the $QCP(\sigma, Q'_1, Q'_2)$ for unary mDatalog(τ^M_o) on unranked ordered σ-labeled trees is evaluated to **yes** since we have:

$$\begin{aligned}
& Q_1 \not\subseteq Q_2 \\
\iff\ & \text{there is a ranked ordered } \Sigma\text{-labeled tree } T \text{ and a node } v \in T \\
& \text{such that } v \in \mathcal{T}^\omega_{\mathcal{P}_1}(T) \text{ and } v \notin \mathcal{T}^\omega_{\mathcal{P}_2}(T) \\
\iff\ & \text{there is a ranked ordered } \Sigma\text{-labeled tree } T, \\
& \text{an unranked ordered } \sigma\text{-labeled tree } T', \text{ and a node } v \in T' \text{ such that} \\
& \mathcal{S}^M_o(T) = \mathcal{S}^M_o(T'),\ v \in \mathcal{T}^\omega_{\mathcal{P}'_1}(T'), \text{ and } v \notin \mathcal{T}^\omega_{\mathcal{P}'_2}(T') \\
\iff\ & Q'_1 \not\subseteq Q'_2
\end{aligned}$$

The construction of $\mathcal{P}_{rk}$ can be done in time linear in the size of the input alphabet Σ (cf. Lemma 3.22) and by Theorem 5.1 the algorithm that decides $QCP(\sigma, Q'_1, Q'_2)$ terminates in time exponential in the size of the input σ, Q'_1, and Q'_2. Thus, the query containment problem for unary mDatalog($\tau^{\mathbf{child}}_{GK}$) on ranked ordered labeled trees belongs to Exptime. □

Now we are ready to formulate and prove the first completeness result of mDatalog on finite labeled trees in general.

Corollary 5.18. *Let τ be one of the schema τ^N_u and τ^M_o where N is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}\}$ and M is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}, \mathbf{child}, \mathbf{ls}\}$.*

The query containment problem for mDatalog(τ) on finite labeled trees is complete for Exptime.

Proof. We have to show that the problem is hard for Exptime and it belongs to Exptime for the different schemas and alphabets given.

By Corollary 4.9 we know that the query containment problem of mDatalog(τ^N_u) on (ranked and unranked) unordered labeled trees is Exptime-hard. Moreover, by Corollary 4.14 we know that the query containment problem of mDatalog(τ^M_o) on (ranked and unranked) ordered labeled trees is Exptime-hard. This implies that the query containment problem for mDatalog(τ) on finite labeled trees is Exptime-hard.

By Corollary 5.17 we know that the query containment problem for $mDatalog(\tau)$ on finite labeled trees belongs to EXPTIME in the ranked case as well as it belongs to EXPTIME by Theorem 5.1 and Corollary 5.2 in the unranked case.

Thus, the query containment problem for mDatalog(τ) on finite labeled trees is complete for EXPTIME. □

The used Theorems and Corollaries for Corollary 5.18 provide a 1-fold exponential algorithm that solves the query containment problem even for unary monadic datalog on labeled trees. Now, we can use this algorithm to solve the equivalence and the emptiness problem with the same time resources.

Corollary 5.19. *Let τ be one of the schema τ_u^N and τ_o^M where N is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}\}$ and M is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}, \mathbf{child}, \mathbf{ls}\}$.*

(a) The equivalence problem for unary mDatalog(τ) on labeled finite trees and

(b) the emptiness problem for unary mDatalog(τ) on labeled finite trees

belong to EXPTIME.

Proof. Let $\mathfrak{A}$ be the algorithm obtained by the used results of Corollary 5.18.
(a) Let Σ, Q_1 and Q_2 be an input for the equivalence problem. An algorithm $\mathfrak{A}_\equiv$ that decides the problem can proceed as follows.

(1) it starts $\mathfrak{A}(\Sigma, Q_1, Q_2)$,

(2) it starts $\mathfrak{A}(\Sigma, Q_2, Q_1)$,

(3) if $\mathfrak{A}(\Sigma, Q_1, Q_2)$ or $\mathfrak{A}(\Sigma, Q_2, Q_1)$ returns **no** then $\mathfrak{A}_\equiv$ returns **no**, otherwise $\mathfrak{A}_\equiv$ returns **yes**.

By definition of equivalence and query containment, this algorithm $\mathfrak{A}_\equiv$ decides correctly and $\mathfrak{A}_\equiv$ is a 1-fold exponential time algorithm since so is $\mathfrak{A}$.
(b) Let $Q_\tau^\varnothing$ be a query in monadic datalog of the schema τ that is unsatisfiable for example the query $Q_\tau^\varnothing$ from Example 2.10 and let Σ and Q be an input for the emptiness problem. An algorithm $\mathfrak{A}_\varnothing$ that decides the problem can proceed as follows.

(1) it starts $\mathfrak{A}(\Sigma, Q, Q_\tau^\varnothing)$,

(2) if $\mathfrak{A}(\Sigma, Q, Q_\tau^\varnothing)$ returns **yes** then $\mathfrak{A}_\varnothing$ returns **yes**, otherwise $\mathfrak{A}_\varnothing$ returns **no**.

By definition of emptiness and query containment, this algorithm $\mathfrak{A}_\varnothing$ decides correctly and $\mathfrak{A}_\varnothing$ is a 1-fold exponential time algorithm since so is $\mathfrak{A}$. □

We close this chapter by presenting completeness results for the remaining problems.

Corollary 5.20. *Let τ be one of the schema τ_u^N and τ_o^M where N is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}\}$ and M is equal to or a subset of $\{\mathbf{root}, \mathbf{leaf}, \mathbf{child}, \mathbf{ls}\}$.*

(a) The equivalence problem for unary mDatalog(τ) on labeled finite trees and

(b) the emptiness problem for unary mDatalog(τ) on labeled finite trees

are complete for Exptime.

Proof. By Corollary 5.19 both problems belong to Exptime. Now it suffices to refer to the corresponding hardness results.
(a) The equivalence problem is Exptime-hard by Corollary 4.9 in the (unranked and ranked) unordered case and by Corollary 4.14 for the ordered case.
(b) The emptiness problem is Exptime-hard by Proposition 4.4 and Proposition 4.8 in the unordered case as well as by Corollary 4.10 in the ordered case. □

Chapter 6

The Distant Relatives Point of View:

Dealing with the Descendant-Axis

In this chapter we consider the case that the descendant-axis is involved. We present algorithms solving the emptiness problem, the equivalence problem, and the query containment problem within 2-fold exponential time. After that, we identify classes of queries rewritable within polynomial time into queries that do not use the **desc** predicate. By using results of the previous chapter, we conclude EXPTIME-completeness for the aforementioned problems on these classes.

Finally, we will see that these problems are hard for 2EXPTIME in general. To achieve this, we foremost present an intuitive reduction from the validity problem of Boolean conjunctive queries with respect to a tree automaton to the query containment problem of Boolean mDatalog on ordered unranked trees using a maximal schema. This proof cannot be extended to unordered trees. And so, for the general case a reduction from the word problem of exponential space bounded alternating Turing machines will be presented.

6.1 On Membership

It is to clarify whether the EXPTIME-membership results of Theorem 5.1 and Corollary 5.2 can be generalized to queries that also use the descendant predicate **desc**. However, the first approach described in the proof of Theorem 5.1 yields a 3-fold exponential algorithm. We can improve this by using methods and results from [GK04] and [GKS06] to eliminate the **desc**-predicate at the expense of an exponential blow-up of the query size. Afterwards, we apply the algorithms provided by Theorem 5.1 and Corollary 5.2. This leads to the following:

Theorem 6.1. *The QCP for unary mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf},\mathbf{desc}\}}$) on unordered trees and for unary mDatalog($\tau_{GK}^{\{\mathbf{child},\mathbf{desc}\}}$) on ordered trees can be solved in 2-fold exponential time.*

Note that $\tau_u^{\{\mathbf{root},\mathbf{leaf},\mathbf{desc}\}} \subseteq \tau_{GK}^{\{\mathbf{child},\mathbf{desc}\}}$. Thus, to prove Theorem 6.1, it suffices to provide a 2-fold exponential algorithm for the QCP for unary mDatalog($\tau_{GK}^{\{\mathbf{child},\mathbf{desc}\}}$) on ordered trees.

Upon input of two mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-queries Q_1 and Q_2, our algorithm proceeds as follows: First, within exponential time, we transform Q_1 and Q_2 into equivalent queries

Q'_1 and Q'_2 that do *not* contain the **desc**-predicate. Afterwards, we use the algorithm provided by Theorem 5.1 for unranked trees or Corollary 5.17 for ranked trees to decide whether $Q'_1 \subseteq Q'_2$ in time exponential in the size of Q'_1 and Q'_2. Thus, Theorem 6.1 is an immediate consequence of Theorem 5.1, Corollary 5.17, and the following Lemma 6.2.

Lemma 6.2. *For every mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query Q there is an equivalent mDatalog($\tau_{GK,\Sigma}^{\mathbf{child}}$)-query Q', which can be computed in 1-fold exponential time.*

The proof of Lemma 6.2 proceeds in three steps:

Step 1: Gottlob, Koch, and Schulz [GKS06, Theorem 6.6] showed that every conjunctive query using the axes **child**, **desc**, **ns** can be rewritten, in 1-fold exponential time, into an equivalent union of *acyclic* conjunctive queries. We extend their result to monadic datalog rules that may also contain the **fc**-relation (see Lemma 6.5 below).

Step 2: Afterwards, we use a result of [GK04] which shows that every *acyclic* conjunctive query can be rewritten, in linear time, into a monadic datalog program in TMNF (see Lemma 6.4 below).

Step 3: Finally, we observe that each TMNF-rule which uses the **desc**-relation can be replaced (in linear time) by two suitable rules using **child** (see Proposition 6.3).

Step 3 is established by the following obvious proposition:

Proposition 6.3.
Over trees, the rule $X(x) \leftarrow \mathbf{desc}(x,y), Y(y)$ *is equivalent to the rules*

$$X(x) \leftarrow \mathbf{child}(x,y), Y(y)$$
$$X(x) \leftarrow \mathbf{child}(x,y), X(y).$$

Similarly, the rule $X(x) \leftarrow \mathbf{desc}(y,x), Y(y)$ *is equivalent to the rules*

$$X(x) \leftarrow \mathbf{child}(y,x), Y(y)$$
$$X(x) \leftarrow \mathbf{child}(y,x), X(y).$$ ⌟

For Steps 1 and 2, let us recall the notion of *acyclic queries* considered in [GK04, GKS06]. Let τ be a schema consisting of relations of arity at most two. Let r be a rule of a monadic datalog query of schema τ. The *directed rule graph* G_r is the multigraph whose vertex set is the set of variables of r, and where for each binary atom of the form $R(x,y)$ occurring in the rule's body, there is a directed edge e_R from node x to node y. The *shadow* of G_r is the undirected multigraph obtained from G_r by ignoring the edge directions. We say that r contains a *directed cycle* if the multigraph G_r contains a directed cycle. Accordingly, r contains an *undirected cycle* if the shadow of G_r contains a cycle. A rule is called *acyclic* if it does not contain an undirected cycle; an mDatalog(τ)-program is *acyclic* if all its rules are acyclic.

Step 2 of our agenda is provided by the following lemma.

Lemma 6.4 ([GK04, Lemma 5.8])**.** *Let r be an acyclic monadic datalog rule over relations that are either unary or binary. Then, r can be decomposed in linear time into a monadic datalog program in which each rule is in one of the three forms*

$$X(x) \leftarrow R(y,x), Y(y) \qquad X(x) \leftarrow R(x,y), Y(y) \qquad X(x) \leftarrow Y(x), Z(z)$$

where x (resp., Y) may but does not have to be different from z (resp., Z).

Finally, Step 1 of our agenda is established by the following Lemma 6.5, which generalizes a result by Gottlob, Koch, and Schulz [GKS06, Theorem 6.6] to queries that may make use of the **fc**-predicate.

Lemma 6.5. *Every unary mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query Q can be rewritten, in 1-fold exponential time, into an equivalent mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query Q' such that each rule in the program of Q' is acyclic.*

Proof. Let $\mathcal{P}$ be the program of Q. We choose Q' to have the same query predicate as Q. The program $\mathcal{P}'$ of Q' is constructed as follows.

We initialize $\mathcal{P}'$ to be equal to $\mathcal{P}$. Then, while $\mathcal{P}'$ is *not* acyclic, do the following: Let r be a rule in $\mathcal{P}'$ that is not acyclic. Remove r from $\mathcal{P}'$.

Case 1: If r contains a directed cycle, note that r is not satisfiable (since the directed cycle is built from the axes **fc**, **ns**, **child**, **desc**).
Thus, we simply drop r.

Case 2: Otherwise, r must contain an undirected cycle, but no directed cycle. Then, the directed query graph G_r is a DAG, and there must exist a variable z of r which belongs to an undirected cycle, such that G_r contains no directed path from z to another variable that belongs to an undirected cycle. For this variable z, the rule's body must contain two atoms of the form $R(x,z)$ and $S(y,z)$ (where $R,S \in \{\mathbf{fc},\mathbf{ns},\mathbf{child},\mathbf{desc}\}$, and x,y are variables). We make the following case distinction:

(i) In case that $R = \mathbf{fc}$ and $S = \mathbf{ns}$ (or vice versa), note that the rule is unsatisfiable, and hence we simply drop r.

(ii) In case that $R = S \in \{\mathbf{fc},\mathbf{ns},\mathbf{child}\}$, note that $R(x,z) \wedge S(y,z)$ is equivalent to $R(x,z) \wedge y{=}x$. Thus, we let $\tilde{r}$ be the rule obtained from r by omitting the atom $S(y,z)$ and replacing all occurrences of y by x. We add $\tilde{r}$ to $\mathcal{P}'$.

(iii) In case that $R = S = \mathbf{desc}$, note that $R(x,z) \wedge S(y,z)$ is equivalent to $\varphi :=$

$$\big(\,\mathbf{desc}(x,y) \wedge \mathbf{desc}(y,z)\big) \vee \big(\,\mathbf{desc}(y,x) \wedge \mathbf{desc}(x,z)\big) \vee \big(\,\mathbf{desc}(x,z) \wedge y{=}x\big).$$

For each $i \in \{1,2,3\}$ we let $\tilde{r}_i$ be the rule obtained from r by replacing "$R(x,z), S(y,z)$" with the i-th clause of φ. Concerning $\tilde{r}_3$, we furthermore delete the atom $y{=}x$ and replace all occurrences of y by x. We add $\tilde{r}_1$, $\tilde{r}_2$, and $\tilde{r}_3$ to $\mathcal{P}'$.

(iv) In case that $R = \mathbf{fc}$ and $S = \mathbf{child}$ (or vice versa), note that $R(x,z) \wedge S(y,z)$ is equivalent to $R(x,z) \wedge y{=}x$. Hence, we proceed in the same way as in case (ii).

(v) In case that $R = \mathbf{ns}$ and $S \in \{\mathbf{child},\mathbf{desc}\}$ (or vice versa), note that $R(x,z) \wedge S(y,z)$ is equivalent to $R(x,z) \wedge S(y,x)$. We let $\tilde{r}$ be the rule obtained from r by replacing the atom $S(y,z)$ with the atom $S(y,x)$, and we add $\tilde{r}$ to $\mathcal{P}'$.

(vi) In case that $R \in \{\mathbf{fc},\mathbf{child}\}$ and $S = \mathbf{desc}$ (or vice versa), note that $R(x,z) \wedge S(y,z)$ is equivalent to $\varphi :=$

$$\big(R(x,z) \wedge \mathbf{desc}(y,x)\big) \vee \big(R(x,z) \wedge y{=}x\big).$$

For each $i \in \{1, 2\}$ we let $\tilde{r}_i$ be the rule obtained from r by replacing "$R(x, z), S(y, z)$" with the i-th clause of φ. Concerning $\tilde{r}_2$, we furthermore delete the atom $y{=}x$ and replace all occurrences of y by x. We add $\tilde{r}_1$ and $\tilde{r}_2$ to $\mathcal{P}'$.

Clearly, the obtained query Q' is equivalent to the original query Q. Furthermore, along the same lines as in the proof of [GKS06, Lemma 6.5], one can show that the algorithm terminates after a number of steps that is at most 1-fold exponential in the size of the input query Q. Of course, upon termination the program $\mathcal{P}'$ is acyclic. Thus, the proof of Lemma 6.5 is complete. □

Finally, we are ready for the proof of Lemma 6.2.

Proof of Lemma 6.2:
Let Q be the given mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query. Using Lemma 6.5 we construct, within 1-fold exponential time, an equivalent mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query Q_1 such that each rule in the program $\mathcal{P}_1$ of Q_1 is acyclic. By applying Lemma 6.4 to each rule of $\mathcal{P}_1$, we obtain an equivalent mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query Q_2 such that each rule in the program $\mathcal{P}_2$ of Q_2 is in one of the following forms:

$$X(x) \leftarrow R(y, x), Y(y) \qquad X(x) \leftarrow R(x, y), Y(y) \qquad X(x) \leftarrow Y(x), Z(z)$$

with $R \in \{\mathbf{fc}, \mathbf{ns}, \mathbf{child}, \mathbf{desc}\}$.
Applying Fact 6.3, we then replace every rule of $\mathcal{P}_2$ that contains the **desc**-relation by two rules that use the **child**-relation.
This leads to an mDatalog($\tau_{GK,\Sigma}^{\mathbf{child}}$)-query Q_3 that is equivalent to Q. Furthermore, Q_3 is computed in time 1-fold exponential in the size of Q. □

Transferring the results of Theorem 6.1 in the same way as in Corollary 5.19, we obtain the following.

Corollary 6.6.

(a) The equivalence and the emptiness problem for unary mDatalog($\tau_u^{\{\mathbf{root},\mathbf{leaf},\mathbf{desc}\}}$) on labeled unordered trees can be solved in 2-fold exponential time.

(b) The equivalence problem and the emptiness problem for unary mDatalog($\tau_{GK}^{\{\mathbf{child},\mathbf{desc}\}}$) on labeled ordered trees can be solved in 2-fold exponential time.

6.2 Omitting the descendant-axis

In this section, we identify classes of queries in monadic datalog on labeled trees that use the descendant-axis but can be rewritten with an efficient algorithm, i.e., an algorithm that uses time polynomial in the size of the query.

6.2.1 Easy Observations

Analyzing the proof of Theorem 6.1, the crucial point is the cyclicity of programs, more precisely, in compliance with Theorem 4.12, the cyclicity of rules that contain the **desc**-predicate. So, for the first, we note the following.

Corollary 6.7. *Let $Q = (\mathcal{P}, P)$ be an mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query such that every cyclic rule r in $\mathcal{P}$ does not contain the* **desc***-predicate. An equivalent acyclic query Q' in mDatalog($\tau_{GK,\Sigma}^{\mathbf{child}}$) can be computed within polynomial time.*

Proof. An algorithm that computes Q' on input Q can proceed rulewise, as follows. If a rule r contains the **desc**-predicate, by assumption r has to be acyclic. Now, we use Lemma 6.4 and Proposition 6.3 to rewrite the rule. Otherwise, r does not contain a **desc**-predicate, but, potentially, is a cyclic rule, we use Theorem 4.12. □

Corollary 6.7 immediately implies the following weaker statement.

Corollary 6.8. *Let $Q = (\mathcal{P}, P)$ be an acyclic mDatalog($\tau_{GK,\Sigma}^{\{\mathbf{child},\mathbf{desc}\}}$)-query Q. An equivalent acyclic query Q' in mDatalog($\tau_{GK,\Sigma}^{\mathbf{child}}$) can be computed within polynomial time.* ⌟

6.2.2 An Example

Thus, it remains to consider cyclic rules which contain the **desc**-predicate. We obtained the membership result of Theorem 6.1 by using a result of Gottlob, Koch, and Schulz in the context of unions of conjunctive queries. In the same publication Gottlob et al. proved that in general this bound is tight.

For their tightness proof, they considered the following *n-diamond Boolean conjunctive query* D_n.[1]

$$D_n \leftarrow Y_1(y_1) \wedge \bigwedge_{i=1}^{n} \big(\mathbf{desc}(y_i, x_i) \wedge X_i(x_i) \wedge \mathbf{desc}(x_i, y_{i+1}) \wedge \mathbf{desc}(y_i, x_i') \wedge X_i'(x_i') \wedge \mathbf{desc}(x_i', y_{i+1}) \wedge Y_{i+1}(y_{i+1}) \big)$$

The reason for the name's choice can be seen in Figure 6.1(a). In Theorem 7 of [GKS06] it is shown that there is no family $(Q_n)_{n\geqslant 1}$ of unions of acyclic conjunctive queries not using **desc** that is equivalent to D_n and of size polynomial in n.

Which trees fulfill this query? The query D_n asks for a path in the tree which starts in any node and contains nodes labeled by $Y_1, Y_2, \ldots, Y_{n+1}$ in ascending sequence in the direction away from the root. Additionally, for every $1 \leqslant i \leqslant n$ there have to be at least two nodes in any order where one is labeled by X_i and the other by X_i', within the path between Y_i and Y_{i+1}.

Now, let us consider this query in the context of mDatalog. To this end, we generalize this query to an $(m_1, \ldots, m_n)$-chain of diamonds and obtain a query depicted in Figure 6.1(c). A direct translation to a Boolean mDatalog($\tau_u^{\mathbf{desc}}$)-query leads to the query $Q_{(m_1,\ldots,m_n)} = (\mathcal{P}, P)$ with

$$\mathcal{P} = \{\ r, \quad P(x) \leftarrow \mathbf{child}(x, y), P(y)\ \}$$

[1] In [GKS06] the predicate **desc** is denoted by $\mathbf{child}^+$.

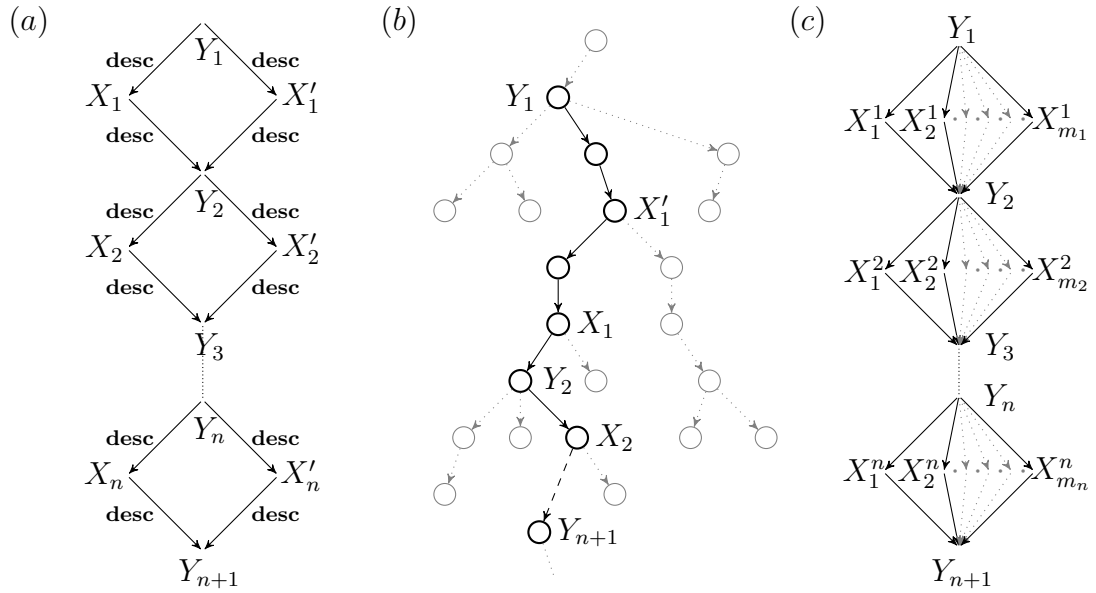

Figure 6.1: (a) The query graph G_{D_n} of D_n. (b) An unordered tree that fulfills the query D_n. The witness path is drawn in black as well as the crucial nodes and their labels. (c) The query graph $G_{D_{(m_1,\ldots,m_n)}}$ of $D_{(m_1,\ldots,m_n)}$ which is a chain of diamonds and in this way a generalization of the n-diamond query D_n.

$$
\begin{aligned}
\text{for } r := \quad & P(y_{n+1}) \leftarrow \mathbf{label}_{Y_1}(y_1), \\
& \qquad \mathbf{desc}(y_1, x_1^1), \mathbf{label}_{X_1^1}(x_1^1), \mathbf{desc}(x_1^1, y_2), \\
& \qquad \mathbf{desc}(y_1, x_2^1), \mathbf{label}_{X_2^1}(x_2^1), \mathbf{desc}(x_2^1, y_2), \\
& \qquad\quad \vdots \\
& \qquad \mathbf{desc}(y_1, x_{m_1}^1), \mathbf{label}_{X_{m_1}^1}(x_{m_1}^1), \mathbf{desc}(x_{m_1}^1, y_2), \\
& \quad \mathbf{label}_{Y_2}(y_2), \\
& \qquad \mathbf{desc}(y_2, x_1^2), \mathbf{label}_{X_1^2}(x_1^2), \mathbf{desc}(x_1^2, y_3), \\
& \qquad \mathbf{desc}(y_2, x_2^2), \mathbf{label}_{X_2^2}(x_2^2), \mathbf{desc}(x_2^2, y_3), \\
& \qquad\quad \vdots \\
& \qquad \mathbf{desc}(y_2, x_{m_2}^2), \mathbf{label}_{X_{m_2}^2}(x_{m_2}^2), \mathbf{desc}(x_{m_2}^2, y_3), \\
& \quad \mathbf{label}_{Y_3}(y_3), \\
& \qquad \vdots \\
& \quad \mathbf{label}_{Y_n}(y_n), \\
& \qquad \mathbf{desc}(y_n, x_1^n), \mathbf{label}_{X_1^n}(x_1^n), \mathbf{desc}(x_1^n, y_{n+1}), \\
& \qquad \mathbf{desc}(y_n, x_2^n), \mathbf{label}_{X_2^n}(x_2^n), \mathbf{desc}(x_2^n, y_{n+1}), \\
& \qquad\quad \vdots \\
& \qquad \mathbf{desc}(y_n, x_{m_n}^n), \mathbf{label}_{X_{m_n}^n}(x_{m_n}^n), \mathbf{desc}(x_{m_n}^n, y_{n+1}), \\
& \quad \mathbf{label}_{Y_{n+1}}(y_{n+1}).
\end{aligned}
$$

The second rule of $\mathcal{P}$ is only necessary because of the specific definition of a Boolean query. For $m_1 = m_2 = \ldots = m_n = 2$ the query is equivalent to the conjunctive query D_n in the sense that $Q_{(2,\ldots,2)}(T) = D_n(T)$ for every $n \in \mathbb{N}$ on every Σ-labeled unordered input tree T.

Let us use the following set $\mathcal{P}'$ of acyclic rules to replace the cyclic rule r in $\mathcal{P}$.

$$
\begin{aligned}
Acc_{Y_1}(x) &\leftarrow \mathbf{label}_{Y_1}(x)\\
Acc^{\mathbf{desc}}_{Y_1}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{Y_1}(y)\\
Acc_{X^1_1}(x) &\leftarrow \mathbf{label}_{X^1_1}(x), Acc^{\mathbf{desc}}_{Y_1}(x)\\
Acc^{\mathbf{desc}}_{X^1_1}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{X^1_1}(y)\\
Acc_{X^1_2}(x) &\leftarrow \mathbf{label}_{X^1_2}(x), Acc^{\mathbf{desc}}_{Y_1}(x)\\
Acc^{\mathbf{desc}}_{X^1_2}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{X^1_2}(y)\\
&\vdots\\
Acc_{X^1_{m_1}}(x) &\leftarrow \mathbf{label}_{X^1_{m_1}}(x), Acc^{\mathbf{desc}}_{Y_1}(x)\\
Acc^{\mathbf{desc}}_{X^1_{m_1}}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{X^1_{m_1}}(y)
\end{aligned}
$$

$$
\begin{aligned}
Acc_{Y_2}(x) &\leftarrow \mathbf{label}_{Y_2}(x), Acc^{\mathbf{desc}}_{X^1_1}(x), Acc^{\mathbf{desc}}_{X^1_2}(x), \ldots, Acc^{\mathbf{desc}}_{X^1_{m_1}}(x)\\
Acc^{\mathbf{desc}}_{Y_2}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{Y_2}(y)\\
&\vdots\\
Acc_{Y_n}(x) &\leftarrow \mathbf{label}_{Y_n}(x), Acc^{\mathbf{desc}}_{X^{n-1}_1}(x), Acc^{\mathbf{desc}}_{X^{n-1}_2}(x), \ldots, Acc^{\mathbf{desc}}_{X^{n-1}_{m_{n-1}}}(x)\\
Acc^{\mathbf{desc}}_{Y_n}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{Y_n}(y)\\
Acc_{X^n_1}(x) &\leftarrow \mathbf{label}_{X^n_1}(x), Acc^{\mathbf{desc}}_{Y_n}(x)\\
Acc^{\mathbf{desc}}_{X^n_1}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{X^n_1}(y)\\
Acc_{X^n_2}(x) &\leftarrow \mathbf{label}_{X^n_2}(x), Acc^{\mathbf{desc}}_{Y_n}(x)\\
Acc^{\mathbf{desc}}_{X^n_2}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{X^n_2}(y)\\
&\vdots\\
Acc_{X^n_{m_n}}(x) &\leftarrow \mathbf{label}_{X^n_{m_n}}(x), Acc^{\mathbf{desc}}_{Y_n}(x)\\
Acc^{\mathbf{desc}}_{X^n_{m_n}}(x) &\leftarrow \mathbf{desc}(y,x), Acc_{X^n_{m_n}}(y)
\end{aligned}
$$

$$
\begin{aligned}
Acc_{Y_{n+1}}(x) &\leftarrow \mathbf{label}_{Y_{n+1}}(x), Acc^{\mathbf{desc}}_{X^n_1}(x), Acc^{\mathbf{desc}}_{X^n_2}(x), \ldots, Acc^{\mathbf{desc}}_{X^n_{m_n}}(x)\\
P(x) &\leftarrow Acc_{Y_{n+1}}(x)
\end{aligned}
$$

We obtain an equivalent query Q' which is acyclic. It is easy to verify that Q' can be constructed in time polynomial in the size of $Q_{(m_1,\ldots,m_n)}$ and D_n, respectively. The idea for the construction of $\mathcal{P}'$ is to transport the information top-down, from the Y_1 colored node down to the Y_{n+1} colored node: For every node v of an input tree T colored by a color X in level i the fact $Acc_X(v)$ becomes true if and only if there is an ancestor v' colored Y_i and its ancestors fulfill the part from a Y_1 colored node in T to v'.

Finally, we translated (a generalization of) the n-diamond query D_n of [GKS06] into an acyclic query in monadic datalog within polynomial time. Recall, that the n-diamond query was used to prove a gap of exponential size between conjunctive queries using **desc** and unions of acyclic conjunctive queries (using **desc**).

6.2.3 Path Rules

Now, we use the observation of the previous subsection to identify a class of rules that can be handled in a similar way as the generalized n-diamond query. For this subsection let $\tau_{\mathbf{desc}}$ be a schema over finite labeled trees, such that the only binary relation symbol is **desc**.

Let r be a rule of a monadic datalog query of schema $\tau_{\mathbf{desc}}$. We assume that r has no directed cycles, otherwise it is unsatisfiable. The rule r is called a *path rule* if for every connected component of the directed rule graph there exists a variable x in r such that x is in every topological sort over the attendant component the maximum, and one of these maxima is the head variable. A *topological sort* is a linear order $\prec$ over the variables of r such that for every pair (x, y) of variables and every atom $\mathbf{desc}(x, y)$ in r it holds $x \prec y$.[2] For example, the Y_{n+1} colored node in Figure 6.1(a) is such a unique maximum.

Note, for every model of a path rule the value function maps the variables to nodes of one path in the tree. Therefore a path rule speaks about path properties.[3]

First, we observe that it is easily decidable if a given rule r in mDatalog($\tau_{\mathbf{desc}}$) is a path rule or not.

Lemma 6.9. *Let r be a rule in mDatalog($\tau_{\mathbf{desc}}$). There is an algorithm that decides on input r within polynomial time if r is a path rule.*

Proof. It is to verify if for every connected component of the rule graph of r there exists exactly one variable that does not have an outgoing edge, and one of them is the head variable.[4]

An algorithm deciding path rules can proceed as follows. We split the rule according to its connected components C_0, C_1, ... Now, we start with the empty sets $Cand_{C_i}$ and $nonCand_{C_i}$ for every component by reading the component C_i atom by atom. If an atom is binary, that is an atom of the form $\mathbf{desc}(x, y)$, we verify

(1) if the variable y is not element of one of the sets $Cand_{C_i}$ and $nonCand_{C_i}$, then we add y to $Cand_{C_i}$ and

(2) if $x \in Cand_{C_i}$, then we delete x from $Cand_{C_i}$ and insert it into $nonCand_{C_i}$.

Finally, if every component is read, we return **yes** if each of the sets $Cand_{C_i}$ consists of exactly one variable, and one of them is the head variable of r. Otherwise we return **no**. It is obvious that the algorithm has polynomial running time and it is easy to verify that the algorithm is correct. □

[2] Since r has no directed cycles at least one such order exists and as **desc** is not reflexive such an order is not reflexive.

[3] That's the reason for choosing the name *path rule*. These path rules are different from known regular path queries in the context of graph databases.

[4] An introduction into topological order and corresponding sorting algorithms can be found in various textbooks, for example Cormen, Leiserson, Rivest, and Stein [CLRS09].

Before we start to give a construction and prove its correctness, we need the meaning of *exactly colored* rules and queries. An associated lemma provides us to every mDatalog rule on finite labeled trees an equivalent query of exactly colored rules.

Let $\mathcal{P}$ be a program in mDatalog over finite labeled trees. Let r be a rule of $\mathcal{P}$ that contains at least one binary atom. We call r *exactly colored* if and only if for every variable x of r there exists exactly one idb predicate P_x of $\mathcal{P}$ and vice versa,[5] such that $P_x(x) \in \text{body}(r)$.

Let $S^{\mathcal{P}}_{bin} \subseteq \mathcal{P}$ be the set of all rules of $\mathcal{P}$ which have at least one binary atom in its body. Then the program $\mathcal{P}$ is called *exactly colored* if and only if every rule $r \in S^{\mathcal{P}}_{bin}$ is exactly colored, this means for every variable x of a rule r there exists exactly one idb predicate P^r_x of $\mathcal{P}$ and vice versa, such that $P^r_x(x) \in \text{body}(r)$ and $P^r_x(y) \notin \text{body}(r')$ for every rule $r' \neq r$ and every variable y.

Lemma 6.10. *Let Σ be an alphabet and let τ be a schema over finite labeled trees. For every query $Q = (\mathcal{P}, P)$ in mDatalog(τ) there exists a query $Q' = (\mathcal{P}', P')$, such that $\mathcal{P}'$ is exactly colored and for every Σ-labeled finite tree T and every node $v \in T$ we have*

$$P(v) \in \mathcal{T}^{\omega}_{\mathcal{P}}(T) \qquad \Longleftrightarrow \qquad P'(v) \in \mathcal{T}^{\omega}_{\mathcal{P}'}(T).$$

The program $\mathcal{P}'$ can be constructed in time polynomial in the size of $\mathcal{P}$.

Proof. If τ is a schema for unordered labeled trees, we start with

$$\mathcal{P}' := \{I_0(x) \leftarrow \mathbf{child}(x, y)\ , \qquad I_0(x) \leftarrow \mathbf{child}(y, x)\}$$

and if τ is a schema for ordered labeled trees, we initialize $\mathcal{P}'$ with

$$\mathcal{P}' := \{I_0(x) \leftarrow \mathbf{fc}(x, y)\ , \qquad I_0(x) \leftarrow \mathbf{fc}(y, x)\ , \qquad I_0(x) \leftarrow \mathbf{ns}(y, x)\}.$$

Now I_0 becomes true for every node of the tree.

From now, an algorithm that constructs $\mathcal{P}'$ proceeds rulewise. If the i-th rule r of $\mathcal{P}$ has no binary atom then it puts the rule r to $\mathcal{P}'$. Otherwise, let $\{x_1, \ldots, x_{|\text{var}(r)|}\}$ be the variables of r. We introduce $|\text{var}(r)|$ new unary idb predicates $I^i_1, \ldots, I^i_{|\text{var}(r)|}$. Let $Pred^i_1$ be the set of unary atoms of r and $Pred^i_2$ be the set of the binary atoms of r. Now, we add the rule r' with head $\text{head}(r') := \text{head}(r)$ and

$$\text{body}(r') = Pred^i_2 \cup \bigcup_{1 \leqslant j \leqslant |\text{var}(r)|} I^i_j(x_j),$$

this means we remove the old unary atoms and give to every variable a unique color. Additionally, we add for every variable $x_j \in \{x_1, \ldots, x_{|\text{var}(r)|}\}$ the following rule to $\mathcal{P}'$. If there is no predicate P such that $P(x_j) \in Pred^i_1$, then we add

$$I^i_j(x) \leftarrow I_0(x)$$

to $\mathcal{P}'$ and otherwise we add the rule with head $I^i_j(x)$ and the body that contains for every atom $P(x_j) \in Pred^i_1$ an atom $P(x)$ (for any predicate P). In this way the new color of the variable v in r' is the mix of all existing colors of v in r.

Finally, if the iteration is complete, we obtain the desired query in time polynomial in the size of $\mathcal{P}$. □

[5] This means there is no variable $y \neq x$, such that $P_x(y) \in \text{body}(r)$.

The next step of our agenda is to show that for every path rule r there exists an acyclic program that is equivalent and its size is polynomial in r. Therefore, we introduce the following set according to the definition of $F_{\mathcal{P},A}$ (c.f. page 13) for a finite labeled tree T and a program $\mathcal{P}$ in mDatalog.

$$F_{\mathcal{P},T} := \mathit{atoms}(T) \cup \{\ R(v) \ :\ R \in \mathrm{idb}(\mathcal{P}),\ v \in V^T\ \}$$

Proposition 6.11. *Let $\mathcal{P}$ be a monadic datalog program for finite labeled trees. Let $r \in \mathcal{P}$ be a path rule in mDatalog($\tau_{\mathbf{desc}}$) with head predicate h. There exists a set of acyclic rules $\mathcal{P}'$ in mDatalog($\tau_{\mathbf{desc}}$), such that for every finite labeled tree T, every node $v \in T$, and every set $P(T)$ with $\mathit{atoms}(T) \subseteq P(T) \subseteq F_{\mathcal{P},T}$ we have*

$$h(v) \in \mathcal{T}^{\omega}_{\{r\}}(P(T)) \qquad \Longleftrightarrow \qquad h(v) \in \mathcal{T}^{\omega}_{\mathcal{P}'}(P(T)).$$

The program $\mathcal{P}'$ can be constructed in time polynomial in the size of r.

Proof. Let r be the input rule with head variable x. By Lemma 6.10, we can assume that r is exactly colored.

An algorithm that computes an equivalent query can proceed as follows. In a first step it constructs the rule graph and obtains its set of components $\mathcal{C} = \{C_0, C_1, \ldots\}$ of size $c \in \mathbb{N}$. Analogously to Lemma 6.9, the algorithm determines to every component C_i the head variable x_i. For the beginning we add

$$h(x) \leftarrow C_0(x_0), C_1(x_1), \ldots, C_{c-1}(x_{c-1})$$

to the empty set $\mathcal{P}'$ of rules. Observe x is equal to x_i for some $0 \leqslant i < c$. Now, we proceed component wise and consider the rules r_i for $0 \leqslant i < c$

$$C_i(x_i) \leftarrow r_{|_{V(C_i)}}$$

where $r_{|_M}$ for a set $M \subseteq \mathrm{var}(r)$ of variables denotes the following set of body atoms of the rule r.

$$r_{|_M} := \{\ b \in \mathrm{body}(r)\ \mid \mathrm{var}(b) \subseteq M\ \}$$

For a given component rule $r_i := C_i(x_0) \leftarrow r_{|_{V(C_i)}}$ with $n = |\,\mathrm{var}(r_i)|$, let y_{n-1}, y_{n-2}, …, y_1, y_0 be the variables in an arbitrary topological sort $\prec^{r_i}$. Thus implies $y_0 = x_i$. Furthermore, for $0 \leqslant j < n$ let M_i^j be the following set

$$M_i^j := \left\{ x \;\middle|\; \begin{array}{l} x \in \mathit{var}(r_i), \\ x \text{ has at most } j \text{ successors concerning } \prec^{r_i} \end{array} \right\}.$$

Recursively over M_i^j , we build a query equivalent to $r_i^j := C_i(x_0) \leftarrow r_{|_{M_i^j}}$ and finally for M_i^{n-1}, we obtain a query equivalent to the component rule r_i. By repeating this approach for all components $C_i \in \mathcal{C}$ of r we obtain the desired query.

For M_i^0 the rule r_i^0 is of the form

$$C_i(x_0) \leftarrow X_0(x_0)$$

for some unique color X_0. Now, it is obvious that we obtain an equivalent query if the program $\mathcal{P}_i^0$ consists of the following rules.

$$Acc_{X_0}(x) \leftarrow X_0(x)$$
$$C_i(x) \leftarrow Acc_{X_0}(x)$$

For M_i^j with $j > 0$, we extend the program $\mathcal{P}_i^{j-1}$ obtained from r_i^{j-1} in the following way. Let x_j be the new variable such that $\{x_j\} = M_i^j \setminus M_i^{j-1}$. Furthermore, let X_j be the unique color of x_j. In a first step, we set $\mathcal{P}_i^j := \mathcal{P}_i^{j-1}$ and we add the rules

$$Acc_{X_j}(x) \leftarrow X_j(x)$$
$$Acc_{X_j}^{\mathbf{desc}}(y) \leftarrow \mathbf{desc}(x, y), Acc_{X_j}(x)$$

to $\mathcal{P}_i^j$. Let

$$\begin{aligned}\mathrm{Bin}_i^j &:= \{\mathbf{desc}(x_j, x) \in \mathrm{body}(r_i^j) \mid x \in \mathrm{var}(r_j)\} \\ &= r_{|_{M_i^j}} \setminus (r_{|_{M_i^{j-1}}} \cup \{X_j(x_j)\})\end{aligned}$$

be the set of all new binary predicates. Observe, Bin_i^j is the set of all binary predicates of r_i where x_j occurs in the first component. For every variable x with $\mathbf{desc}(x_j, x) \in \mathrm{Bin}_i^j$ and its unique color X there exists by construction a unique rule $r' \in \mathcal{P}_i^j$ with head predicate $Acc_X(x)$. Each body of such a rule will be extended by the atom $Acc_{X_j}^{\mathbf{desc}}(x)$. For $j = n - 1$ we set $\mathcal{P}_i' := \mathcal{P}_i^j$ and finally $\mathcal{P}' := \bigcup_{0 \leqslant i < c} \mathcal{P}_i'$.

As every binary atom of r occurs in exactly one set Bin_i^j for $0 \leqslant j < n$ and $0 \leqslant i < c$, the construction can be done within polynomial time. For the correctness, it suffices to show that for every unordered labeled tree T, every node $v \in T$, and every set $P(T)$ with $atoms(T) \subseteq P(T) \subseteq F_{P,T}$ it holds that

$$C_i(v) \in \mathcal{T}_{\{r_i\}}^{\omega}(P(T)) \qquad \Longleftrightarrow \qquad C_i(v) \in \mathcal{T}_{\mathcal{P}_i}^{\omega}(P(T)).$$

To establish the necessary condition, we prove the following invariant by induction over j for the rules r_i^j.

> *If there is a value function* $\beta : \mathrm{var}(r_i^j) \to V(T)$ *that satisfies* r_i^j*, then for every color* X *of a variable* $x \in \mathrm{var}(r_i^j)$ *we have* $Acc_X(\beta(x)) \in \mathcal{T}_{\mathcal{P}_i^j}^{\omega}(P(T))$.

As β satisfies r_i^j, it implies for $\beta(x_0) = v$ and $C_i(v) \in \mathcal{T}_{\{r_i^j\}}^{\omega}(P(T))$ that $Acc_{X_0}(v) \in \mathcal{T}_{\mathcal{P}_i^j}^{\omega}(P(T))$ and by the rule $C_i(x) \leftarrow Acc_{X_0}(x)$ the fact $C_i(v) \in \mathcal{T}_{\mathcal{P}_i^j}^{\omega}(P(T))$.

For the induction base the claim is clear since every satisfying value function is a function that assigns x_0 to an X_0 colored node v of the input tree T that fulfills the rule

$$Acc_{X_0}(x) \leftarrow X_0(x)$$

of $\mathcal{P}_0^i$. Let the claim be true for $j \geqslant 0$.

We assume the value function β satisfies the rule r_i^{j+1}. Let x_{j+1} be the new variable such that $\{x_{j+1}\} = M_i^{j+1} \setminus M_i^j$ and let X_{j+1} be the unique color of x_{j+1}. Since β satisfies the rule r_i^{j+1}, it also satisfies the rule r_i^j and by induction hypothesis it holds for every variable $x \in \mathrm{var}(r_i^j)$ and its color X that $Acc_X(\beta(x)) \in \mathcal{T}_{\mathcal{P}_i^j}^{\omega}(P(T))$. Now, $\mathcal{P}_i^{j+1}$ and $\mathcal{P}_i^j$ differ in only two points.

(1) The new rules

$$Acc_{X_{j+1}}(x) \leftarrow X_{j+1}(x)$$
$$Acc^{\mathbf{desc}}_{X_{j+1}}(y) \leftarrow \mathbf{desc}(x, y), Acc_{X_{j+1}}(x)$$

(2) In the bodies of all $Acc_X(x)$ headed rules of variables connected by an $\mathbf{desc}(x_j, x)$ predicate in r_i^{j+1}. These are extended by $Acc^{\mathbf{desc}}_{X_{j+1}}(x)$.

As β satisfies r_i^{j+1}, we know $X_{j+1}(\beta(x_{j+1}))$ is true and therefore $Acc_{X_{j+1}}(\beta(x_{j+1}))$ is in $\mathcal{T}^{\omega}_{\mathcal{P}_i^{j+1}}(P(T))$. Moreover, we have $Acc^{\mathbf{desc}}_X(v') \in \mathcal{T}^{\omega}_{\mathcal{P}_i^{j+1}}(P(T))$ for all descendants v' of $\beta(x_j)$. For the extended rules we know by the satisfying property of β that all variables x connected with $\mathbf{desc}(x_j, x)$ are actually descendants of $\beta(x_j)$ and therefore the extended rules in $\mathcal{P}_i^{j+1}$ where satisfied by β.

For the converse direction, the sufficient condition, we show by induction over the stages j of the fix point process $\mathcal{T}^{\omega}_{\mathcal{P}_i}(P(T))$ the following invariant.

> *If $Acc_X(v)$ for some node $v \in T$ is newly added during stage j of the fix point process then the path rule r_{i_X} is fulfilled by a value function $\beta : \mathrm{var}(r_{i_X}) \to V(T)$ with $\beta(x) = v$ where x is the variable of color X.*

The path rule r_{i_X} is defined as follows. Let G_i be the rule graph of r_i. Now the rule graph G_{i_X} of r_{i_X} is the induced sub graph of G_i where the set $V(r_{i_X})$ of nodes is defined as $V(r_{i_X}) := \{X\} \cup \{Y \mid X \text{ is reachable from } Y\}$ and the head variable of r_{i_X} is the variable colored by X.

Note, if $Acc_{X_0}(v) \in \mathcal{T}^{\omega}_{\mathcal{P}_i}(P(T))$ for some node $v \in T$ is true and therefore it holds that $C_i(v) \in \mathcal{T}^{\omega}_{\mathcal{P}_i}(P(T))$, it is implied that $C_i(v) \in \mathcal{T}^{\omega}_{\{r_i\}}(P(T))$.
For the inductive base we consider $j = 1$ since for no color X the predicate $Acc_X \in \mathrm{idb}(\mathcal{P}) \cup \mathrm{edb}(\mathcal{P})$ and so for node $v \in T$, we have $Acc_X(v) \in \mathcal{T}^0_{\mathcal{P}_i}(P(T))$. Let $Acc_X(v) \in \mathcal{T}^1_{\mathcal{P}_i}(P(T))$ be true for some color X and some node $v \in T$. As $Acc_X(v) \notin \mathcal{T}^0_{\mathcal{P}_i}(P(T))$, the fact $Acc_X(v)$ is provided by the rule

$$Acc_X(x) \leftarrow X(x).$$

Thus, v is colored by X. The induced path rule r_{i_X} of r_i is of the form $C_i^X(x) \leftarrow X(x)$ that is satisfied by β with $\beta(x) = v$.

For the inductive step, we assume $Acc_X(v) \in \mathcal{T}^j_{\mathcal{P}_i}(P(T))$ and $Acc_X(v) \notin \mathcal{T}^{j-1}_{\mathcal{P}_i}(P(T))$ is true for some node color X and some node $v \in T$. Now, $Acc_X(v) \in \mathcal{T}^j_{\mathcal{P}_i}(P(T))$ is provided by a rule of from

$$Acc_X(x) \leftarrow X(x), Acc^{Desc}_{X_1}(x), \ldots, Acc^{Desc}_{X_l}(x)$$

for some colors $X_1, \ldots, X_l$ and $l \in \mathbb{N}_{\geqslant 1}$.

By construction, this implies that $Acc_{X_1}(v_1), \ldots, Acc_{X_l}(v_l) \in \mathcal{T}^{j-1}_{\mathcal{P}_i}(P(T))$ for some predecessor $v_1, \ldots, v_l$ of v. Using the induction hypothesis, we know there are value functions $\beta_1, \ldots, \beta_l$ such that

(1) $\beta_1(x_1) = v_1,\ \beta_2(x_2) = v_2, \ldots,\ \beta_l(x_l) = v_l$ and

(2) $\beta_s(x_s)$ satisfies $r_{i_{X_s}}$ for $1 \leqslant s \leqslant l$.

For $l = 1$ it is obvious that the following extension β' of β with

$$\beta'(x') := \begin{cases} \beta_1(x') & x' \in \mathrm{var}(r_{i_{X_1}}) \\ v & x' = x \end{cases}$$

is satisfying r_{i_X}.

In the same way, we can determine a satisfying value function if the rule graphs $r_{i_{X_1}} \dots r_{i_{X_s}}$ have disjoint nodes. Then

$$\beta'(x') := \begin{cases} \beta_s(x') & x' \in \mathrm{var}(r_{i_{X_s}}) \text{ for } 1 \leqslant s \leqslant l \\ v & x' = x \end{cases}$$

fulfills the rule graph r_{i_X} as all predecessor $v_1, \dots, v_s$ of v are located of the same path from v to the root of T.

It remains the case that the colors of $r_{i_{X_1}} \dots r_{i_{X_s}}$ are not disjoint. Let $r_{i_{X_1}}$ and $r_{i_{X_2}}$ be two rules having the same color X'. Next, we observe that the rules $(r_{i_{X_1}})_{X'}$ and $(r_{i_{X_2}})_{X'}$ have the same rule graph. Recall, the nodes of the images of β_1 and β_2 are located on the same path and therefore the value function β' with

$$\beta'(x') := \begin{cases} \beta_1(x') & x' \in \mathrm{var}(r_{i_{X_1}}), x' \notin \mathrm{var}(r_{i_{X_2}}) \\ \beta_2(x') & x' \notin \mathrm{var}(r_{i_{X_1}}), x' \in \mathrm{var}(r_{i_{X_2}}) \\ \beta_1(x') & x' \in \mathrm{var}(r_{i_{X_1}}), x' \in \mathrm{var}(r_{i_{X_2}}), \beta_1(x') \text{ is a predecessor of } \beta_2(x') \text{ in } T \\ \beta_2(x') & \text{otherwise} \end{cases}$$

satisfies $r_{i_{X_1}}$ and $r_{i_{X_2}}$. Using this approach successively, a value function β' satisfying $r_{i_{X_1}} \dots r_{i_{X_s}}$ is defined and can be extended such that $\beta(x) = v$. □

We summarize the results of this section by the following proposition that is obtained immediately from the propositions and corollaries of this section and that leads to a class of monadic datalog queries efficiently rewritable into queries without **desc**-atoms. For this class of queries of schema $\tau_{GK}^{\mathbf{child},\mathbf{desc}}$, the emptiness problem, the equivalence problem, and the query containment problem can be solved within 1-fold exponential time.

Proposition 6.12. *Let τ be a schema over finite labeled trees such that $\tau \subseteq \tau_{GK}^{\mathbf{child},\mathbf{desc}}$. Any mDatalog($\tau_\Sigma$)-query $Q = (\mathcal{P}, P)$ can be rewritten into an equivalent monadic datalog query from schema $\tau_{GK}^{\mathbf{child}}$ in time polynomial in the size of the query if for every rule r of $\mathcal{P}$ at least one of the following conditions is satisfied:*

(a) r does not contain the **desc** *predicate, or*

(b) r is acyclic, or

(c) every binary predicate in the body of r is a **desc** *predicate and the rule r is a path rule.* ⌟

We close this section by considering a unary $(m_1, \ldots, m_n)$ chain of diamonds rule not asking for the Y_{n+1} colored node but for the Y_1 colored node. A rewriting of the last rule

$$P(x) \leftarrow Acc_{Y_{n+1}}(x)$$

of $\mathcal{P}'$ (presented on page 91) to

$$P(x) \leftarrow Acc_{Y_1}(x)$$

would not suffice. A program $\mathcal{P}^{\text{up}}$ constructed in an analogous way but giving the information bottom up will also not suffice. This is because a query Q' using $\mathcal{P}^{\text{up}}$ constructed in this way cannot distinguish between the tree T depicted in Figure 6.2(b) which is a model of the query and the tree T' (Figure 6.2(c)) which is not.

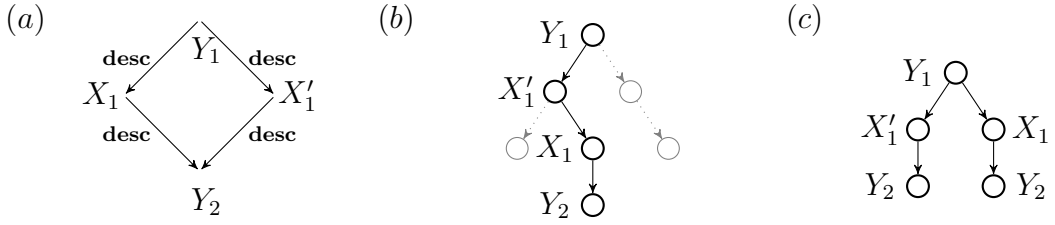

Figure 6.2: (a) The query graph G_Q of $Q = (\{r\}, Ans)$ where r is the following rule:

$$Ans(y_1) \leftarrow Y_1(y_1), \mathbf{desc}(y_1, x_1), X_1(x_1), \mathbf{desc}(y_1, x_1'), X_1'(x_1'), \mathbf{desc}(x_1, y_2), \mathbf{desc}(x_1', y_2), Y_2(y_2).$$

(b) An unordered tree T that fulfills the unary query Q and (c) an unordered tree T' that does not fulfill the unary query Q. The trees T and T' cannot be distinguished from each other by a unary mDatalog(τ_u)-program which is organized in a bottom-up scan.[6]

So, we realize: Important information speaking about properties of a node or a set of nodes on a path can be collected top-down. But merging information bottom-up seems to be unsafe in the sense that different paths can be involved.

6.3 On Hardness of the QCP on Ordered Unranked Trees Using the Descendant-Axis

In the previous section, we have seen that there are classes of queries using **desc** which can be efficiently translated into queries without **desc** that means the considered static analysis problems EXPTIME-complete for them. In general, this translation is not possible, more precisely, the problems are complete for 2EXPTIME if the descendant-axis is involved. The actual section's goal is to show that the query containment problem for unranked ordered trees using the schema $\tau_{GK}^{\mathbf{desc}}$ is hard for 2EXPTIME.

We use the following result by Björklund, Martens, and Schwentick published in 2008.

Theorem 6.13 (Björklund et al. [BMS08a])**.** *Validity of Boolean CQ*(**child**, **desc**) *with respect to a tree automaton is* 2EXPTIME*-complete.*

[6] As a solution for this special case, we can combine a top-down scan with a subsequent bottom-up scan.

This problem was considered in the context of conjunctive queries on trees, reflecting semi-structured data with respect to DTD and XML Schema Definitions, represented by a nondeterministic tree automaton,[7] which are formally defined as follows.[8]

A *nondeterministic (unranked) tree automaton* (NTA, for short) $\mathtt{A} = (\Sigma, S, \Delta, F)$ consists of an unranked alphabet Σ, a finite set of states S, $F \subseteq S$ the set of accepting (or final) states, and a set of transition rules Δ of the form $(s, \alpha) \to L$, where $s \in S$ is a state of $\mathtt{A}$, $\alpha \in \Sigma$, and L is a regular string language over the states of $\mathtt{A}$. The regular languages themselves are given by NFA's $\mathtt{A}_L = (\Sigma_L, Q_L, \delta_L, q_L, F_L)$, where the alphabet $\Sigma_L := S$ simply consists of the states of the NTA $\mathtt{A}$, Q_L is the set of states, $\delta_L \subseteq (Q_L \times \Sigma_L \times Q_L)$ is the transition relation, $q_L \in Q_L$ is the initial (or starting) state, and $F_L \subseteq Q_L$ is the set of accepting states. A run and the acceptance condition of the NFA is defined as usual. A run of the NTA $\mathtt{A}$ on an ordered unranked tree T is a mapping $\rho : V^T \to S$ such that the following is true for all nodes v of T where α is the label of v in T:

- If v has n children $u_1, \ldots, u_n$ (from left to the right) then there exists a rule $(s, \alpha) \to L$ in Δ such that $\rho(v) = s$ and $w_v = \rho(u_1) \cdots \rho(u_n) \in L$.
- If v is a leaf then there exists a rule $(s, \alpha) \to L$ in Δ such that $\rho(v) = s$ and $\varepsilon \in L$ where ε is the empty word (string).

A run is accepting if the root is labeled with an accepting state $s \in F$, a finite unranked ordered tree T labeled by Σ is accepted if there exists an accepting run of $\mathtt{A}$ on T, and finally, the set of all accepted trees is denoted by $L(\mathtt{A})$, the language of the automaton $\mathtt{A}$.

We define $\mathrm{strL}(\mathtt{A}) := \{L : \ \Delta \text{ contains a rule } (s, \alpha) \to L\}$ to be the set of all string languages occurring in the image of Δ. The size of the automaton $\mathtt{A}$ is the size of the NTA together with its NFA's $\mathtt{A}_L$, precisely

$$\|\mathtt{A}\| \ := \ |\Sigma| + |S| + |\Delta| + \sum_{L \in \mathrm{strL}(\mathtt{A})} \|\mathtt{A}_L\|, \quad \text{where } \mathtt{A}_L \text{ decides } L \text{ and } \|\mathtt{A}_L\| \ := \ |Q_L| + |\delta_L|.$$

Now, we are ready to prove the following theorem.

Theorem 6.14. *Let N be the following set $N := \{\mathbf{child}, \mathbf{desc}\}$. The QCP for Boolean mDatalog(τ^N_{GK}) on ordered unranked trees is hard for* 2Exptime.

Proof. The proof proceeds by a polynomial time reduction from the Validity Problem for Boolean CQ($\mathbf{child}, \mathbf{desc}$) with respect to a tree automaton that is 2Exptime-hard by Theorem 6.13.

Let the CQ($\mathbf{child}, \mathbf{desc}$)-query Q_{CQ} and the NTA $\mathtt{A}$ together with its NFA's $\mathtt{A}_L$ be the input for the Validity Problem where the unranked alphabet Σ is the input alphabet of $\mathtt{A}$. We construct from Q_{CQ} an mDatalog($\tau^N_{GK,\Sigma}$)-query $Q = (\mathcal{P}, P)$ and from $\mathtt{A}$ an mDatalog($\tau_{GK,\Sigma}$)-query $Q_{\mathtt{A}} = (\mathcal{P}_{\mathtt{A}}, P_{\mathtt{A}})$ within polynomial time in the size of Q_{CQ} and $\mathtt{A}$,[9] such that Q_{CQ} is valid w.r.t $\mathtt{A}$ if and only if $Q_{\mathtt{A}} \subseteq Q$.

[7] In fact, these automata represent much more expressive Relax NG schemas (cf. [CM01]).

[8] Note, the transition relation differs substantially from the transition relation of the nondeterministic bottom-up tree automaton introduced in section 5.3. In particular, this automaton receives unranked trees as input and not necessarily binary trees.

[9] Note, $Q_{\mathtt{A}}$ is an mDatalog($\tau^N_{GK,\Sigma}$)-query (that does not use the **desc** predicate) since $\tau_{GK,\Sigma} \subseteq \tau^N_{GK,\Sigma}$.

Using Lemma 2.9, we obtain an equivalent mDatalog($\tau^N_{GK,\Sigma}$)-query $Q_{\mathtt{A}}$ from Q_{CQ} within linear time. The following Lemma 6.15 provides us with a Boolean mDatalog($\tau_{GK,\Sigma}$)-query $Q_{\mathtt{A}}$ constructed from a NTA $\mathtt{A}$ working on unranked ordered trees labeled by Σ in time polynomial in the size of $\mathtt{A}$ such that for every unranked ordered tree T, $\mathtt{A}$ accepts T if and only if $Q_{\mathtt{A}} = \mathbf{yes}$.

Now, it holds that:

Q_{CQ} is valid w.r.t $\mathtt{A}$
$\iff$ for every unranked ordered tree $T \in L(\mathtt{A})$, we have $Q_{\text{CQ}}(T) = \mathbf{yes}$
$\iff$ for every unranked ordered tree T with $Q_{\mathtt{A}}(T) = \mathbf{yes}$, we have $Q(T) = \mathbf{yes}$
$\iff$ for every unranked ordered tree T, we have $Q_{\mathtt{A}}(T) \subseteq Q(T)$
$\iff$ $Q_{\mathtt{A}} \subseteq Q$. □

To establish the claim of Theorem 6.14, it remains to verify the following lemma.

Lemma 6.15. *Let $\mathtt{A} = (\Sigma, S, \Delta, F)$ be an NTA over the unranked alphabet Σ. There exists an mDatalog($\tau_{GK,\Sigma}$)-query $Q = (\mathcal{P}, P)$, such that for every unranked ordered tree T labeled by symbols from Σ it holds that*

$$\mathtt{A} \text{ accepts } T \qquad \Longleftrightarrow \qquad Q(T) = \mathbf{yes}.$$

Q is constructible from $\mathtt{A}$ in time polynomial in the size of $\mathtt{A}$.

Proof. Let $\mathtt{A} = (\Sigma, S, \Delta, F)$ be an NTA over the unranked alphabet Σ. The datalog program will compute for every node $v \in T$ the set of states the automaton $\mathtt{A}$ can receive in this node. So, we introduce for every state s of $\mathtt{A}$ an idb predicate s. The query Q will accept an input tree T if we have $s(root^T) \in \mathcal{T}^\omega_{\mathcal{P}}(T)$ for at least one state $s \in F$. In some sense the program will compute the well known power set construction.

Moreover, the program simulates the NFA $\mathtt{A}_L$ for every string language $L \in \text{strL}(\mathtt{A})$. To this end, let for every unordered unranked tree T the set

$$F_T \;:=\; \mathit{atoms}(T) \cup \{s(v) : v \in V^T, s \in S \text{ is a state of } \mathtt{A}\}$$

be the set consisting all atomic facts over T together with an assignment of all states of $\mathtt{A}$ to every node in T. For now, let $P(T)$ be a set with $\mathit{atoms}(T) \subseteq P(T) \subseteq F_T$. We start by constructing for every NFA $\mathtt{A}_L$ such a program $\mathcal{P}_L$.

For a given L let $\mathtt{A}_L = (\Sigma_L = S, Q_L, \delta_L, q_L, F_L)$ be the automaton given by the input, such that $L(\mathtt{A}_L) = L$. Let v be a node in T and $u_1, \ldots, u_n$ all its children where u_1 is the first child of v. We simulate $\mathtt{A}_L$ in the same way as above by computing for every node $u \in u_1, \ldots, u_n$ the set of states $q \in \mathtt{A}_L$ reachable by reading words on the nodes starting in u_1, going on from one sibling to the next. To do this, we introduce the idb predicates L_q and Acc_L, whereas a node $u \in u_1, \ldots, u_n$ is marked by L_q if the automaton $\mathtt{A}_L$ can reach the state $q \in Q_L$ by reading a word starting in u_1. Furthermore, a last sibling u_n is marked by Acc_L if this automaton can reach a final state in u_n, what means that $\mathtt{A}_L$ accepts the input. The possible input of $\mathtt{A}_L$ on a node u is given by the facts $s(u)$ in $P(T)$ for any state s of $\mathtt{A}$.

For the beginning let $\mathcal{P}_L$ be the empty set. For every entry (q_L, s, q) in δ_L where q_L is the initial state of $\mathtt{A}_L$, which says the automaton $\mathtt{A}_L$ starting in the first child x can reach state q by reading input s, we add the rule

$$L_q(x) \leftarrow \mathbf{fc}(y, x), s(x)$$

to $\mathcal{P}_L$.

Now, for every entry (q, s, q') in δ_L, we add the rule

$$L_{q'}(x) \leftarrow L_q(y), \mathbf{ns}(y, x), s(x)$$

that ensures that a node x is marked with the state q' of $\mathtt{A}_L$ if its predecessor sibling in respect to $order^T$ holds the state q and the input $s(x)$ is a fact in $P(T)$.

Finally, a last sibling can accept a language if it carries a final state of $\mathtt{A}_L$. So, we finish $\mathcal{P}_L$ by adding the rules

$$Acc_L(x) \leftarrow \mathbf{ls}(x), L_q(x)$$

for all final states $q \in F_L$ of $\mathtt{A}_L$.

Now, it is easy to verify that for the constructed program $\mathcal{P}_L$, every ordered unranked tree T, and every set $P(T)$ with $atoms(T) \subseteq P(T) \subseteq F_T$ the following is true:

> For all non leaves v in T and all its children $u_1, u_2, \ldots, u_n$ mentioned in the sibling order, such that u_1 is the first child of v and u_n is the last one, it holds that $Acc_L(u_n) \in \mathcal{T}^{\omega}_{\mathcal{P}_L}(P(T))$ if and only if there are $s_1, \ldots, s_n \in S$, such that the facts $s_1(u_1), s_2(u_2), \ldots, s_n(u_n)$ are elements of $P(T)$ and the word $s_1(u_1)\ s_2(u_2) \cdots s_n(u_n) \in L$.

Apparently, all the programs $\mathcal{P}_L$ for $L \in \mathrm{strL}(\mathtt{A})$ can be constructed in time linear in the size of $\mathtt{A}_L$ and have disjoint rules.

Now, we start to construct the program $\mathcal{P}$ to simulate the complete entire $\mathtt{A}$. For the beginning let $\mathcal{P}$ consist of all the programs $\mathcal{P}_L$, so it is

$$\mathcal{P} \ := \bigcup_{L \in \mathrm{strL}(\mathtt{A})} \mathcal{P}_L.$$

The computation of $\mathtt{A}$ starts in the leaves and to initiate the simulation of $\mathtt{A}$ we add for every rule $(s, \alpha) \to L$ with $\varepsilon \in L$ the following rule[10]

$$s(x) \leftarrow \mathbf{label}_{\alpha}(x), \mathbf{leaf}(x)$$

to $\mathcal{P}$. The programs $\mathcal{P}_L$ ensure that every last sibling u_n is marked by $Acc_L(u_n)$ if the states of $\mathtt{A}$ are assigned to u_n and its siblings form a word in L. By adding for every language L the rule

$$\mathbf{child}_{Acc_L}(x) \leftarrow \mathbf{child}(x, y), \mathbf{ls}(y), Acc_L(y)$$

[10]This can be easily checked by verifying whether the initial state q_L of the NFA $\mathtt{A}_L$ is an element of its set of final states F_L.

to $\mathcal{P}$, we transport this information from the last sibling to its parent node. Now, for every rule $(s, \alpha) \to L$ in Δ we add the datalog rule

$$s(x) \leftarrow \mathbf{child}_{Acc_L}(x), \mathbf{label}_\alpha(x)$$

to $\mathcal{P}$ to push to computation of $\mathtt{A}$. Finally, we have to test the acceptance of $\mathtt{A}$ by accepting the computation if the root is marked by an final state of $\mathtt{A}$. So, we add

$$P(x) \leftarrow \mathbf{root}(x), s_{\mathtt{A}}(x)$$

to $\mathcal{P}$ for every state s in the set F of final states of $\mathtt{A}$ and now, the mDatalog($\tau_{GK,\Sigma}$)-query $Q = (\mathcal{P}, P)$ yields **yes** on the input of an unranked ordered tree T if and only if the automaton $\mathtt{A}$ accepts the same input T. It it easy to verify that Q can be computed in time polynomial in the size of $\mathtt{A}$. □

6.4 On Hardness on Ranked Trees

In the previous section, we have seen an intuitive and short proof that proceeds by a reduction from a known problem that is hard for 2EXPTIME. In this proof, we used the order on children during the simulation of the NFA intensely. So, we are not able to extend this proof for results on unordered trees and a more technical proof for this section's goal is necessary. By a reduction from the word acceptance problem of exponential space bounded alternating Turing machine, we establish the following theorem.

Theorem 6.16. *The emptiness problem for Boolean mDatalog($\tau_u^{\mathbf{desc}}$) on ranked unordered labeled trees is* 2EXPTIME*-hard.*

The proof idea of Theorem 6.16 is based on the proof of Theorem 6.13 presented in the full version of the MFCS publication by Björklund, Martens, and Schwentick [BMS08b]. The used alternating Turing machine was introduced at FOCS'76 by Chandra and Stockmeyer [CS76], as well as by Kozen [Koz76], and presented in a joint journal publication in 1981[CKS81].[11]

An *alternating Turing machine* (ATM, for short) $\mathtt{A} = (Q, \Sigma, \Gamma, \delta, q_0)$ consists of

- a finite set of *states* Q partitioned into *universal states* $Q_\forall$, *existential states* $Q_\exists$, an *accepting state* q_a, and a *rejecting state* q_r,
- the finite *input alphabet* Σ,
- the finite *tape alphabet* $\Gamma \supset \Sigma$, that contains the special *blank* symbol $\textvisiblespace \notin \Sigma$,
- the *initial* (or, starting) *state* q_0 and
- the *transition relation* $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{L, R, S\}))$.

[11] For more in-depth information about the topic of alternating Turing machines, we recommend chapter three of Balcázar, Díaz, and Gabarró [BDG90] or the textbooks by Sipser [Sip97], Kozen [Koz06], as well as Vollmer [Vol99] in the context of circuit complexity.

As usual the letters L, R, and S denote the directions *left*, *right*, and *stay* in which the head on the tape is moved.

A *configuration* c of $\mathtt{A}$ is given by specifying its state, the content of its tape together with the position of the tape head. Thus, we interpret a string of the form w_1qw_2 with $w_1, w_2 \in \Gamma^*$, $q \in Q$ as the configuration in which the tape contains the word w_1w_2, followed by blanks, the head's tape position is the first letter of w_2, and q is the current state of the machine. A transition rule $((q,a),(q',b,D)) \in \delta$ denotes a step of $\mathtt{A}$ by reading in state q the letter $a \in \Gamma$, overwriting a on the current position by $b \in \Gamma$, moving the head depending on $D \in \{L,R,S\}$ one position to the left, to the right, or stay, and finally, switching to state q'. A configuration c' obtained by applying a rule of δ to a given configuration c is called *successor configuration* of c. The configuration $w_1bq'w_2$, for example, is a successor configuration of w_1qaw_2 obtained by appling the transition rule $((q,a),(q',b,R))$. A configuration w_1qw_2 is a *halting configuration* if q is either the accepting state q_a or the rejecting state q_r. Without loss of generality, we can assume that there is no successor configuration of any halting configuration, and furthermore, before halting, the automaton moves its head to the left on the first non-blank symbol on the tape, so each halting configuration is of the form qw.

A *computation tree* $T_{\mathtt{A}}$ of the ATM $\mathtt{A}$ on input $w \in \Sigma^*$ is a tree labeled with configurations of $\mathtt{A}$, such that the root of $T_{\mathtt{A}}$ is labeled by q_0w, and for each node u of $T_{\mathtt{A}}$ labeled by w_1qw_2,

- if $q \in Q_\exists$, then u has exactly one child, and this child is labeled with a successor configuration of w_1qw_2,
- if $q \in Q_\forall$, then u has a child v for every successor configuration $w_1'q'w_2'$ of w_1qw_2, and v is labeled by $w_1'q'w_2'$
- if $q \in \{q_a, q_r\}$, then u is a leaf of $T_{\mathtt{A}}$.

Observe that $T_{\mathtt{A}}$ can be infinite, since $\mathtt{A}$ may have non-halting computation branches. A computation tree is *accepting* if all its branches are finite and all its leaves are labeled by configurations in state q_a. The language $L(\mathtt{A})$ of the ATM $\mathtt{A}$ is the set of words $w \in \Sigma^*$ for which there exists an accepting computation tree of $\mathtt{A}$ on w.

We say that an ATM is *normalized* if every non-halting universal configuration has precisely two different successor configurations, each universal step only affects the state of the machine, and additionally, the machine proceeds from a universal state to an existential state, and vice versa.

Lemma 6.17. *For every alternating Turing machine* $\mathtt{A}$ *there exists a normalized alternating Turing machine* $\mathtt{A}_n$ *with* $L(\mathtt{A}) = L(\mathtt{A}_n)$. $\mathtt{A}_n$ *can be constructed from* $\mathtt{A}$ *within polynomial time.*

Proof. To construct the normalized alternating Turing machine $\mathtt{A}_n$, we proceed in three steps in the following order.

(1) We ensure that every non halting universal step only affects the state of the machine.

(2) We introduce new states such that universal and existential configurations alternate.

(3) We ensure that every non halting universal state has exactly two different successor configurations.

Moreover, we ensure that no construction step destroys previously achieved properties.

For every universal state q that affects the head position or that writes on the tape there exists a tuple $((q,a),(q',b,D)) \in \delta$ such that $a \neq b$ or $D \neq S$. Now, we introduce a new state $q^{\exists} \in Q_{\exists}$ and replace $((q,a),(q',b,D))$ with the tuples $((q,a),(q^{\exists},a,S))$ and $((q^{\exists},a),(q',b,D))$.

After (1) is true for the entire automaton, we consider every tuple $((q,a),(q',b,D)) \in \delta$ where $q,q' \in Q_{\exists}$. Now, we introduce a new state $q^{\forall} \in Q_{\forall}$ and replace $((q,a),(q',b,D))$ with the tuples $((q,a),(q^{\forall},b,S))$ and $((q^{\forall},b),(q',b,D))$. For condition (2), it remains to consider successive universal configurations. Thus for every tuple $((q,a),(q',a,S)) \in \delta$ where $q,q' \in Q_{\forall}$, we introduce a new state $q^{\exists} \in Q_{\exists}$ and replace $((q,a),(q',a,S))$ with the tuples $((q,a),(q^{\exists},a,S))$ and $((q^{\exists},a),(q',a,S))$.[12] Note, condition (1) is still fulfilled.

The final step is to ensure that every universal non-halting node has exactly two different successors. Now, we have to consider three cases.

(a) The pair (q,a), with $q \in Q_{\forall}$ and $a \in \Gamma$ has $n > 2$ successor configurations. Let $\{((q,a),(q_i,a,S))\}_{0<i\leqslant n}$ be the tuples of δ leading to these successor configurations. (Recall, condition (1) is fulfilled.) Now, we introduce two new states $q^{\exists}$ and $q^{\forall}$ and replace the entries $((q,a),(q_n,a,S))$ and $((q,a),(q_{n-1},a,S))$ by the tuples $((q,a),(q^{\exists},a,S))$, $((q^{\exists},a),(q^{\forall},a,S))$, $((q^{\forall},a),(q_n,a,S))$, and $((q^{\forall},a),(q_{n-1},a,S))$.

Now, the original state q at reading a has $n-1$ successors and the newly introduced state $q^{\forall}$ has exactly two. This will be iterated $n-2$ times and finally the new universal states, as well as the state q, have at reading the letter a exactly two successor configurations.

(b) For (q,a) there exists exactly one successor state q', where $q' \neq q_a$ and $q' \neq q_r$. This means the successor state is neither the accepting state nor the rejecting state of the machine. Now, q' is an existential state. To this end, we introduce a new existential state $q^{\exists}$, and extend the transition relation by the tuple $((q,a),(q^{\exists},a,S))$, as well as by $((q^{\exists},a),(q'',b,D))$ for every tuple $((q',a),(q'',b,D))$ in δ.

(c) For (q,a) there exists exactly one successor state q' that is the accepting or the rejecting state. Now, we introduce a new existential state $q^{\exists}$ and extend the transition relation δ by the two tuples $((q,a),(q^{\exists},a,S))$ and $((q^{\exists},a),(q',a,S))$.

Note, there is exactly one accepting and exactly one rejecting state, and therefore a direct duplication is not possible.

It is easy to verify that the resulting automaton $\mathtt{A}_n$ can be constructed from $\mathtt{A}$ within polynomial time in the size of $\mathtt{A}$ and $L(\mathtt{A}) = L(\mathtt{A}_n)$. □

Now, we are ready to prove Theorem 6.16.

Proof of Theorem 6.16:
Our proof proceeds by a reduction from the word problem for exponential space bounded

[12] Observe, ensured by condition (1), it already holds for every universal state that $D = S$ and the tape content is untouched.

ATM $\mathtt{A}$. In this problem, the input consists of an exponential space bounded ATM $\mathtt{A}$ and an input word w for $\mathtt{A}$, and the task is to decide if $w \in L(\mathtt{A})$. In [CKS81], this problem was shown to be 2Exptime-complete.

Our reduction will be done from an ATM with empty input. Therefore, we construct for the given ATM $\mathtt{A}$ and the given word w an ATM $\mathtt{A}_w$ that works in space exponential in the size of w and accepts the empty input word if and only if $\mathtt{A}$ accepts w. To do this, we let $\mathtt{A}_w$ start by writing w on the empty tape, afterwards $\mathtt{A}_w$ returns to the leftmost tape position and finally, it starts to simulate the original machine $\mathtt{A}$. By Lemma 6.17, we can assume that $\mathtt{A}_w$ is normalized and since the computation is exponentially space bounded, the non blank portion of the tape during the computation of $\mathtt{A}_w$ on empty input is never longer than 2^n, where n is polynomial in the size $|w|$ of the original input word.

We will choose a suitable ranked alphabet Σ_T, independent from $\mathtt{A}_w$. Within polynomial time, we construct an mDatalog($\tau_{u,\Sigma_T}^{\mathbf{desc}}$)-query $Q = (\mathcal{P}, Ans)$ such that

$$\begin{aligned} Q \neq \varnothing \quad &\iff \quad \text{there is an accepting computation tree for } \mathtt{A}_w \\ &\iff \quad w \in L(\mathtt{A}). \end{aligned}$$

Since 2Exptime is closed under complement, this implies that the emptiness problem for Boolean mDatalog($\tau_u^{\mathbf{desc}}$) on ranked unordered labeled trees is 2Exptime-hard.

In the next paragraphs, we present the encoding of the computation tree that is basically taken from [BMS08b] and includes the encoding of the configuration tree, both are adapted to our problem. So, let $T_{\mathtt{A}_w}$ be a computation tree of $\mathtt{A}_w = (Q, \Sigma, \Gamma, \delta, q_0)$ (cf. Figure 6.3), we fix an arbitrary order of the children of each universal node such that every universal node has a left child and a right child.[13] Now the encoding $T := \text{enc}(T_{\mathtt{A}_w})$ can be obtained from $T_{\mathtt{A}_w}$ by replacing every node v labeled by $w_1 q w_2$ with a tree $\text{enc}(t_v)$, as follows

- if v is universal, then the root of $\text{enc}(t_v)$ is labeled with $\text{CT}_\forall$,
- if v is existential, and v is the root of $T_{\mathtt{A}_w}$ or v is the left child of a universal node, then the root of $\text{enc}(t_v)$ is labeled with $\text{CT}_\exists^{left}$,
- if v is existential, and v is the right child of a universal node, then the root of $\text{enc}(t_v)$ is labeled with $\text{CT}_\exists^{right}$,
- exactly one child of the root of $\text{enc}(t_v)$ is labeled by r (this will be the root of the subtree that encodes the configuration at v), and
- for each child u_i of v in $T_{\mathtt{A}_w}$, $\text{enc}(t_v)$ has a subtree $\text{enc}(t_{u_i})$, which is the encoded subtree of $T_{\mathtt{A}_w}$ obtained by the replacement of u_i.[14]

The set of subtrees denoted by their root label r encode the configurations that originally labeled the computation tree. We have to navigate through 2^n tape cells and we must be able to compare the i-th cell of one configuration with the i-th cell of the predecessor

[13] A node u is a universal node if it is labeled by a configuration $w_1 q w_2$ where q is a universal state. If q is an existential state then u is existential.

[14] In fact, for a non halting configuration there is exactly one child if v is existential. If v is universal, there are exactly two children, since $\mathtt{A}_w$ is normalized.

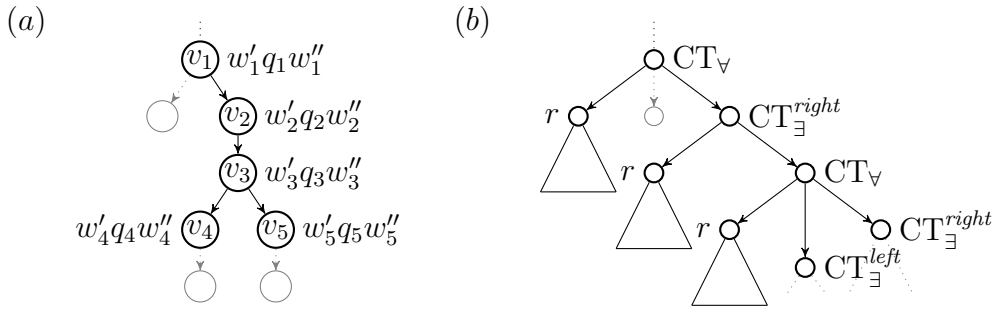

Figure 6.3: (a) A part of a computation tree $T_{\mathtt{A}_w}$ where the node v_1 is labeled by $w_1'q_1w_1''$ is universal, its children are existential. In particular, the node v_2 labeled by $w_2'q_2w_2''$ is the right children of v_1 and v_2 itself has one child, the universal node v_3, and so on. (b) The replacement of v_1 is a tree with a root node labeled by $\mathrm{CT}_\forall$ and with three children, the first is labeled by r and the root of a subtree encoding the configuration in v_1, the second is a replacement for its left child, and the third is the replacement for its right child. The obtained tree $T := \mathrm{enc}(T_{\mathtt{A}_w})$ is an unordered ranked tree.

configuration. Thus, the configuration tree is basically a binary tree of height n that has 2^n leaves to carry the information for the tape cells, together with the information of the current state of the machine and the position of the head. This sequence of 2^n *configuration cells* will carry the whole information about the configuration of the machine in this working step. To this end, the set of configuration cells is partitioned into three types.

- The set BCells of *basic cells* is equal to Γ. A basic cell represents a tape cell that is not currently visited by the head and also is not visited in the predecessor configuration.
- The set CCells of *current tape head cells* is equal to $\Gamma \times \delta$. The letter from Γ represents the tape content in the actual position that is currently visited by the head, while the transition from δ is the transition which leads to the actual configuration.
- The set PCells of *previous tape head cells* is equal to $\Gamma \times (Q \times \Gamma)$ and represents tape cells that were visited by the head in the predecessor configuration, but not in the current one. The first letter from Γ represents the actual content on the tape in this cell and the pair $(Q \times \Gamma)$ the previous state and tape content in the predecessor configuration.

Observe that the number k of all possible configuration cells for $\mathtt{A}_w$ is polynomial in the size of the automaton and so we can refer to each possible configuration cell by a natural number i in $\{1, \ldots, k\}$.

Next, we fix a set of constraints, that allows to decide if a sequence C_1 of 2^n configuration cells is a valid successor configuration of another sequence C_0. We start with constraints to ensure a certain degree of consistency inside a given sequence. The set $H(\mathtt{A}_w)$ of horizontal constraints consists of the following rules:

(H1) The only cell allowed to the left of a cell $(a, ((q_1, b), (q_2, c, R))) \in$ CCells is the cell $(c, (q_1, b)) \in$ PCells.

(H2) The only cell allowed to the right of a cell $(a, ((q_1, b), (q_2, c, L))) \in$ CCells is the cell $(c, (q_1, b)) \in$ PCells.

(H3) The only cell allowed to the right of the basic cell $\sqcup \in \Gamma$ is $\sqcup$ itself.

To fix the set $V(\mathtt{A}_w)$ of vertical constraints between two consecutive sequences C_0 and C_1, we imagine the predecessor is lying cell by cell on top of its successor such that the i-th configuration cell of C_0 is lying on top of the i-th cell of C_1.

(V1) If the i-th cell is a BCell $a \in \Gamma$ then the only allowed cells on the i-th tape position in a successor configuration are a itself and any CCell $(a, ((q_1, b), (q_2, c, m))$ where $m \in \{L, R\}$. The latter is the case that the automaton $\mathtt{A}_w$ just moved to this cell, coming from the left or the right. The letter on this position is currently untouched, but the letter in the left (right) neighbor is overwritten if $b \neq c$ and $m = L$ ($m = R$).

(V2) If the i-th cell is a CCell $(a, ((q_1, b), (q_2, c, m)))$ then the only allowed cells on the i-th tape position in a successor configuration are any $(d, (q_2, a)) \in$ PCells and any $(d, ((q_2, a), (q_3, d, m'))) \in$ CCells where $m' = S$.

(V3) If the i-th cell is a PCell $(a, (q, b))$ then the only allowed cells on the i-th tape position in a successor configuration are the BCell a and any CCell $(a, ((q_1, b), (q_2, c, m))$ where $m \in \{L, R\}$.

Figure 6.4 illustrates an example of valid transitions respecting these constraints. It is easy to verify that if C_0 is a valid encoding of a configuration, C_1 is a valid encoding of a successor configuration if and only if all horizontal and vertical conditions are satisfied.

Now, we are ready to describe the structure of the r-rooted subtrees that encode the configuration; that is the last remaining part of the whole encoding. We already noted that these configuration trees are based on binary trees of height n. Every non root node carries the label s and Björklund et al. called them *skeleton nodes*. Every skeleton node has an attached *navigation gadget*, that is a short path of four nodes labeled by p, 0, 1, $\perp$ for denoting any children as left children, and labeled by p, 1, 0, $\perp$ for right children in the sequence from the skeleton node to the leaf of the gadget (cf. Figure 6.5 (a)).

Each leaf skeleton node, that is a skeleton node that has no skeleton node as child, carries besides the navigation gadget, a *configuration cell gadget* that consists of a path of length $k+2$.[15] The root node of this path is labeled by m (for *me*) followed by k nodes labeled with digits 0 and 1, and the path ends in a leaf labeled with $\perp$. $k-1$ nodes on this path are labeled with 0, only the i-th node is labeled by 1, telling the current cell is the cell number i.

To finish the description of the encoding, for technical reasons, we start in the top of the computation tree with a node labeled with $\top$ that has exactly one child, the topmost configuration node. Now, we are ready to define the ranked alphabet Σ_T and afterwards, to construct the query. The alphabet consists of the following symbols:

$\top$ of arity $ar(\top) = 1$, that denotes the root node of the encoded computation tree.

[15] Recall, k is the number of all possible configuration cells of $\mathtt{A}_w$.

C_x

	BCell	CCell	PCell	
$\cdots$	e	b	f	$\cdots$
		$((q, c), (q', f, L))$	(q, c)	

C_{x+1}

	BCell	CCell	BCell	
$\cdots$	e	a	f	$\cdots$
		$((q', b), (q'', a, S))$		

Figure 6.4: This example shows the corresponding parts of a valid configuration C_x and its successor configuration C_{x+1}. The previous transition $((q, c), (q', f, L))$ leading to configuraion C_x was reading a $c \in \Gamma$ on the right cell, writing an $f \in \Gamma$, switching the state from q to q', and finally moving the head one position to the left. The changeover from C_x to C_{x+1} was done by using transition $((q', b), (q'', a, S))$, saying reading in state q' the letter b, write the letter a, switch to state q'', and stay with the head at the current position.

$\mathrm{CT}_\forall$ of arity $ar(\mathrm{CT}_\forall) = 3$, that denotes a universal configuration.

Leaf-$\mathrm{CT}_\forall$ of arity $ar(\text{Leaf-}\mathrm{CT}_\forall) = 1$, that denotes a halting configuration, that is a child of an existential configuration.[16]

$\mathrm{CT}_\exists^{\boldsymbol{left}}$ of arity $ar(\mathrm{CT}_\exists^{left}) = 2$, that denotes an existential configuration where the configuration itself is the left child of a universal configuration (or the initial configuration).

$\mathrm{CT}_\exists^{\boldsymbol{right}}$ of arity $ar(\mathrm{CT}_\exists^{right}) = 2$, that denotes an existential configuration where the configuration itself is the right child of a universal configuration.

Leaf-$\mathrm{CT}_\exists^{\boldsymbol{left}}$ of arity $ar(\text{Leaf-}\mathrm{CT}_\exists^{left}) = 1$, that denotes a halting configuration where the configuration itself is the left child of a universal configuration (or the initial configuration).

Leaf-$\mathrm{CT}_\exists^{\boldsymbol{right}}$ of arity $ar(\text{Leaf-}\mathrm{CT}_\exists^{right}) = 1$, that denotes a halting configuration where the configuration itself is the right child of a universal configuration.

r of arity $ar(r) = 2$, that denotes the root node of a configuration tree.

s of arity $ar(s) = 3$, that denotes a skeleton node of a configuration tree.

$s_{\mathbf{leaf}}$ of arity $ar(s_{\mathbf{leaf}}) = 2$, that denotes a skeleton leaf node that is a leaf of the configuration tree.

[16] To be precise, a halting configuration is neither an existential nor a universal configuration, but the labels tell us whose configuration child it is.

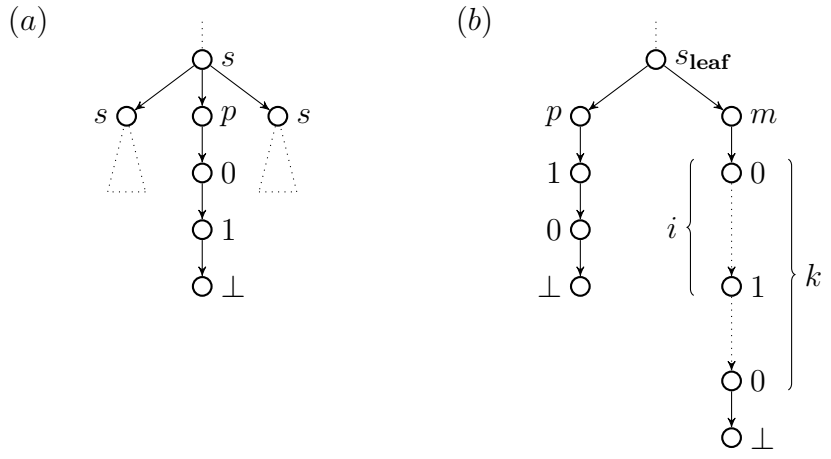

Figure 6.5: (a) A skeleton node that is an inner node of the configuration tree, since it has two children labeled with s. The navigation gadget denotes this node as the left child of its parent node. (b) A skeleton node that is a leaf of the configuration tree that carries a navigation gadget saying it is the right child of its parent node. Moreover, it is carrying a cell gadget m followed by k digits where the i-th one is labeled with 1 if the actual cell is cell $i \in \{1, \ldots, k\}$, while the other digits are 0.

p of arity $ar(p) = 1$, that denotes the root of an navigation gadget.

m of arity $ar(m) = 1$, that denotes the root of a cell gadget 'me'.

0 **and** 1 of arity $ar(0) = ar(1) = 1$, for the values of the gadgets.

$\perp$ the only symbol of Σ_T of arity $ar(\perp) = 0$. So every leaf of the encoding tree is labeled by $\perp$.

The construction of the demanded query $Q = (\mathcal{P}, Ans)$ starts with a program $\mathcal{P}_1$ that ensures the newly introduced idb predicate *structure* for the root if the input tree T is structured as an encoded computation tree. In particular, the input tree must fulfill the following conditions.

(1) The root of the tree is labeled with $\top$ and has exactly one child that represents the initial configuration.

(2) Each configuration node has exactly one child labeled with r.

(3) Every configuration cell gadget correctly encodes a configuration cell.

(4) Each encoded configuration tree is complete and has height n.

(5) Every skeleton node has exactly one correctly assigned navigation gadget.

(6) All horizontal constraints from $H(\mathtt{A}_w)$ are satisfied.

(7) The universal and existential configurations must alternate on the subtree of CT labeled nodes.

(8) For each non halting universal configuration, the two child configuration nodes represent two encoded configuration trees with two different CCells.

(9) The highest encoded configuration tree has the start configuration cell

$$(\llcorner, ((q_0, \llcorner), (q_0, \llcorner, S)))$$

as its leftmost configuration cell. Recall that q_0 is the initial state of $\mathtt{A}_w$ and the computation starts on an empty tape.

(10) Every configuration node that has no successor configuration encodes a final configuration, that implies the leftmost configuration cell is of the form

$$(a, ((q, b), (q_a, c, m))).$$

Recall q_a is the accepting state of the machine, the machine, upon accepting, moves its head to the leftmost tape cell, and finally, an input tree is accepted if every path in the computation tree leads to an accepting halting configuration.

The program $\mathcal{P}_1$ will start in the leaves of the encoded tree and verifies the structure step by step in the direction to the root node. For the beginning the program $\mathcal{P}_1$ is the empty set of rules and the first rule we add is to call leaves by what they are. Thus, we add

$$\mathbf{leaf}(x) \leftarrow \mathbf{label}_{\perp}(x)$$

In any case, a leaf belongs to a gadget, that is a cell configuration or a navigation gadget, and therefore we count the length of the digit path up to the length of k by the following rules.

$$\begin{aligned}
0(x) &\leftarrow \mathbf{label}_0(x)\\
1(x) &\leftarrow \mathbf{label}_1(x)\\
\mathit{digit}(x) &\leftarrow 0(x)\\
\mathit{digit}(x) &\leftarrow 1(x)\\
\mathit{digit}_0(x) &\leftarrow \mathbf{leaf}(x)\\
\mathit{digit}_1(x) &\leftarrow \mathit{digit}(x), \mathbf{child}(x, y), \mathit{digit}_0(y)\\
\mathit{digit}_2(x) &\leftarrow \mathit{digit}(x), \mathbf{child}(x, y), \mathit{digit}_1(y)\\
&\dots\\
\mathit{digit}_k(x) &\leftarrow \mathit{digit}(x), \mathbf{child}(x, y), \mathit{digit}_{k-1}(y)
\end{aligned}$$

Additionally, to ensure that a navigation gadget and the 'me' cell configuration gadget have exactly one node labeled with 1, we count the amount of 1-labeled nodes on every digit path by the following rules.

$$\begin{aligned}
\mathit{count}^1_{<1}(x) &\leftarrow \mathbf{leaf}(x)\\
\mathit{count}^1_{<1}(x) &\leftarrow 0(x), \mathbf{child}(x, y), \mathit{count}^1_{<1}(y)\\
\mathit{count}^1_{=1}(x) &\leftarrow 1(x), \mathbf{child}(x, y), \mathit{count}^1_{<1}(y)\\
\mathit{count}^1_{=1}(x) &\leftarrow 0(x), \mathbf{child}(x, y), \mathit{count}^1_{=1}(y)
\end{aligned}$$

We propagate these counting results to the gadget roots if they are labeled by m or p by adding the following rules to $\mathcal{P}_1$:

$$\begin{aligned}
\mathit{count}^1_{=1}(x) &\leftarrow \mathbf{label}_p(x), \mathbf{child}(x,y), \mathit{count}^1_{=1}(y)\\
\mathit{digit}_2(x) &\leftarrow \mathbf{label}_p(x), \mathbf{child}(x,y), \mathit{digit}_2(y)\\
p(x) &\leftarrow \mathbf{label}_p(x), \mathit{count}^1_{=1}(x), \mathit{digit}_2(x)\\
\mathit{count}^1_{=1}(x) &\leftarrow \mathbf{label}_m(x), \mathbf{child}(x,y), \mathit{count}^1_{=1}(y)\\
\mathit{digit}_k(x) &\leftarrow \mathbf{label}_m(x), \mathbf{child}(x,y), \mathit{digit}_k(y)\\
m(x) &\leftarrow \mathbf{label}_m(x), \mathit{count}^1_{=1}(x), \mathit{digit}_k(x)
\end{aligned}$$

Now, the predicate p becomes true for a node v of the input tree T if it is labeled with p and it is the starting node of a navigation gadget that actually denotes a direction. Similarly, m becomes true for a node v of the input tree T if it is labeled with m and it is the starting node of a 'me' cell configuration gadget that actually denotes a configuration cell.

For the rest of the section, we introduce a predicate $\mathbf{child}^i(x,y)$ for a natural number i as short hand for the set of atoms

$$\mathbf{child}(x,x_1), \mathbf{child}(x_1,x_2), \ldots, \mathbf{child}(x_{i-1},y)$$

where $\mathbf{child}^i(x,y)$ states the fact that y is a descendant of x in the i-th generation.

By the following rules, every m-marked node knows which configuration $i \in \{1,\ldots k\}$ it encodes.

$$\begin{aligned}
m_{k=1}(x) &\leftarrow m(x), \mathbf{child}(x,x_1), 1(x_1)\\
m_{k=2}(x) &\leftarrow m(x), \mathbf{child}^2(x,x_2), 1(x_2)\\
&\vdots\\
m_{k=i}(x) &\leftarrow m(x), \mathbf{child}^i(x,x_i), 1(x_i)\\
&\vdots\\
m_{k=k}(x) &\leftarrow m(x), \mathbf{child}^k(x,x_k), 1(x_k)
\end{aligned}$$

Now, we mark the leaves of the skeleton nodes with the idb predicate $s_{\mathbf{leaf}}$ that are leaves in the configuration tree considered without the gadgets.

$$s_{\mathbf{leaf}}(x) \leftarrow \mathbf{label}_{s_{\mathbf{leaf}}}(x), \mathbf{child}(x,x_m), m(x_m), \mathbf{child}(x,x_p), p(x_p)$$

Observe, the label $s_{\mathbf{leaf}}$ has arity two, so there cannot be further children the rule could work on. Now, for the subtrees rooted by nodes marked with $s_{\mathbf{leaf}}$ the condition (3) is fulfilled. By the next rules, we mark the nodes carrying the label s or $s_{\mathbf{leaf}}$ regarding their navigation gadget as left child using s_L or as right child by using s_R. Remember a correct navigation gadget is marked by the idb predicate p.

$$\begin{aligned}
s_L(x) &\leftarrow \mathbf{label}_{s_{\mathbf{leaf}}}(x), \mathbf{child}(x,x_p), p(x_p), \mathbf{child}(x_p,x_n), 0(x_n)\\
s_R(x) &\leftarrow \mathbf{label}_{s_{\mathbf{leaf}}}(x), \mathbf{child}(x,x_p), p(x_p), \mathbf{child}(x_p,x_n), 1(x_n)\\
s_L(x) &\leftarrow \mathbf{label}_s(x), \mathbf{child}(x,x_p), p(x_p), \mathbf{child}(x_p,x_n), 0(x_n)\\
s_R(x) &\leftarrow \mathbf{label}_s(x), \mathbf{child}(x,x_p), p(x_p), \mathbf{child}(x_p,x_n), 1(x_n)
\end{aligned}$$

We mark the entire configuration tree with the predicate s, that affects the nodes marked by $s_{\mathbf{leaf}}$ and every node labeled by s that have a correct navigation gadget, as well as left and right children.

$$\begin{aligned}
s(x) &\leftarrow s_{\mathbf{leaf}}(x)\\
s(x) &\leftarrow s_L(x), \mathbf{child}(x, x_l), s_L(x_l), s(x_l), \mathbf{child}(x, x_r), s_R(x_r), s(x_r)\\
s(x) &\leftarrow s_R(x), \mathbf{child}(x, x_l), s_L(x_l), s(x_l), \mathbf{child}(x, x_r), s_R(x_r), s(x_r)
\end{aligned}$$

Note that an inner node of the configuration tree is itself a left or right child, that implies there is such a navigation gadget and it gets the s predicate, if it has a left and a right child, marked with s_L and s_R. This implies, this node cannot own a second navigation gadget that claims the opposite of another navigation gadget since the arity of the symbol s enforces the limit of exactly three children. Remember, we have to ensure that the configuration tree is complete and has height n. This will be done if both children of the r labeled root of the configuration tree are marked by height $n-1$ and by s since s is only true for them if every s child itself has two s children downto the leaves of the configuration tree. So, up to $n-1$, we count the height of the configuration tree by adding the following rules to $\mathcal{P}_1$.

$$\begin{aligned}
s_{h=0}(x) &\leftarrow s_{\mathbf{leaf}}(x)\\
s_{h=1}(x) &\leftarrow \mathbf{child}(x, x_l), s_L(x_l), s_{h=0}(x_l), \mathbf{child}(x, x_r), s_R(x_r), s_{h=0}(x_r)\\
&\vdots\\
s_{h=n-1}(x) &\leftarrow \mathbf{child}(x, x_l), s_L(x_l), s_{h=n-2}(x_l), \mathbf{child}(x, x_r), s_R(x_r), s_{h=n-2}(x_r)
\end{aligned}$$

Finally we mark a node labeled by r with the predicate r_{nav} if it is the root of a navigable and complete configuration tree and add the rule

$$\begin{aligned}
r_{nav}(x) \leftarrow \mathbf{label}_r(x), \mathbf{child}(x, x_l), s_L(x_l), s(x_l), s_{h=n-1}(x_l),\\
\mathbf{child}(x, x_r), s_R(x_r), s(x_r), s_{h=n-1}(x_r)
\end{aligned}$$

to $\mathcal{P}_1$. Observe that during the computation of $\mathcal{P}_1(T)$ a node labeled by r gets marked with r_{nav} if it is a root of a complete configuration tree of height n where every skeleton node carries a correct navigation gadget and in the skeleton leaves a cell configuration is correctly encoded. So, the conditions (3) – (5) are fulfilled.

The next goal is to ensure condition (6) that stands for the horizontal constraints (H1)–(H3). This actually holds if the tuple (i, j) of two neighboring configurations cells is contained in the relation $H(\mathtt{A}_w)$. Remember, a node labeled by m is already marked by $m_{k=i}$ for its encoded configuration i. In a first step and for every $i \in \{1, \ldots, k\}$, we propagate this information to the skeleton leaves by the following rules.

$$(k = i)_{\mathbf{leaf}}(x) \leftarrow s_{\mathbf{leaf}}(x), \mathbf{child}(x, y), m_{k=i}(y)$$

Next, we propagate for a subtree of the configuration tree its leftmost and its rightmost configuration cell. Furthermore, it is to verify if the rightmost cell of the left child fits together with the leftmost cell of the right child. Therefore, we use the new predicates

$(k = i)_{\text{left}}$ and $(k = i)_{\text{right}}$ for every $i \in \{1, \ldots, k\}$ in the following rules

$$\begin{aligned}
(k = i)_{\text{left}}(x) &\leftarrow s(x), (k = i)_{\mathbf{leaf}}(x)\\
(k = i)_{\text{right}}(x) &\leftarrow s(x), (k = i)_{\mathbf{leaf}}(x)\\
(k = i)_{\text{left}}(x) &\leftarrow s(x), \mathbf{child}(x, x_l), s_L(x_l), (k = i)_{\text{left}}(x_l)\\
(k = i)_{\text{right}}(x) &\leftarrow s(x), \mathbf{child}(x, x_r), s_R(x_r), (k = i)_{\text{right}}(x_r),
\end{aligned}$$

as well as for every $(i, j) \in H(\mathtt{A}_w)$, the predicate H (if the nodes children fit together) in the following rules

$$\begin{aligned}
H(x) \leftarrow s(x), \mathbf{child}(x, x_l), s_L(x_l), s_{\mathbf{leaf}}(x_l), (k = i)_{\text{right}}(x_l),\\
\mathbf{child}(x, x_r), s_R(x_r), s_{\mathbf{leaf}}(x_r), (k = j)_{\text{left}}(x_r)\\
H(x) \leftarrow s(x), \mathbf{child}(x, x_l), s_L(x_l), H(x_l), (k = i)_{\text{right}}(x_l),\\
\mathbf{child}(x, x_r), s_R(x_r), H(x_r), (k = j)_{\text{left}}(x_r)\\
H(x) \leftarrow \mathbf{label}_r(x), \mathbf{child}(x, x_l), s_L(x_l), H(x_l), (k = i)_{\text{right}}(x_l),\\
\mathbf{child}(x, x_r), s_R(x_r), H(x_r), (k = j)_{\text{left}}(x_r).
\end{aligned}$$

Now, a node labeled by r is marked with H if its configuration tree satisfies all horizontal constraints from $H(\mathtt{A}_w)$. By the following rules, we ensure that in a configuration tree there do not exist two different CCells and use the idb predicate θ_i if the CCell i exists in a subtree and Non_θ if a cell does not belong to CCells. For all $i \in \{1, \ldots, k\}$ where $i \in$ CCells, we add the rule

$$\theta_i(x) \leftarrow (k = i)_{\mathbf{leaf}}(x)$$

and for all $j \in \{1, \ldots, k\}$ where $j \notin$ CCells, we add the rules

$$\text{Non}_\theta(x) \leftarrow (k = j)_{\mathbf{leaf}}(x)$$

to $\mathcal{P}_1$. This will be propagated by

$$\begin{aligned}
\theta_i(x) &\leftarrow \mathbf{child}(x, x_l), s_L(x_l), \theta_i(x_l), \mathbf{child}(x, x_r), s_R(x_r), \text{Non}_\theta(x_r)\\
\theta_i(x) &\leftarrow \mathbf{child}(x, x_l), s_L(x_l), \text{Non}_\theta(x_l), \mathbf{child}(x, x_r), s_R(x_r), \theta_i(x_r)
\end{aligned}$$

for every $i \in \{1, \ldots, k\}$ where $i \in$ CCells and finally, a node labeled by r carries the idb predicate θ_i for exactly one $i \in \{1, \ldots, k\}$ if its configuration contains exactly one CCell, that is the configuration cell i. Otherwise, the node is not marked by any θ_i predicate. Implied by the following rules

$$r(x) \leftarrow \mathbf{label}_r(x), H(x), r_{nav}(x), \theta_i(x) \qquad \text{for all } i \in \text{CCells}$$

every root node of a configuration tree is marked with r if its configuration tree satisfies the conditions (3)–(6).

Purposing the bottom-up analysis of the input tree, we have to verify that a configuration node labeled by Leaf-$\text{CT}_\forall$, Leaf-$\text{CT}_\exists^{left}$, or Leaf-$\text{CT}_\exists^{right}$ represents a halting configuration that is given as CCell in the leftmost cell of its configuration tree. So,

for all $i \in$ CCells representing a configuration cell with current state q_a that is the only accepting state of $\mathtt{A}_w$, we add the rules

$$\begin{aligned}
\text{Leaf-CT}_\forall(x) &\leftarrow \mathbf{label}_{\text{Leaf-CT}_\forall}(x), \mathbf{child}(x, x_r), r(x_r), \theta_i(x_r), (k = i)_{\text{left}}(x_r)\\
\text{Leaf-CT}_\exists^{left}(x) &\leftarrow \mathbf{label}_{\text{Leaf-CT}_\exists^{left}}(x), \mathbf{child}(x, x_r), r(x_r), \theta_i(x_r), (k = i)_{\text{left}}(x_r)\\
\text{Leaf-CT}_\exists^{right}(x) &\leftarrow \mathbf{label}_{\text{Leaf-CT}_\exists^{right}}(x), \mathbf{child}(x, x_r), r(x_r), \theta_i(x_r), (k = i)_{\text{left}}(x_r)
\end{aligned}$$

Recall that the rank of the symbols representing a halting configuration is $ar(\text{Leaf-CT}_\forall) = ar(\text{Leaf-CT}_\exists^{right}) = ar(\text{Leaf-CT}_\exists^{left}) = 1$ and so, for every subtree rooted by a node marked with the latter introduced idb predicates, we ensured conditions (2)–(6) and (10).

It remains to analyze the subtrees of the CT labeled nodes. Recall that an inner node of the CT tree will be positively marked if

(a) it is labeled as universal configuration and it has two existential configuration children (one or both can be a leaf configuration node) carrying different CCells, or

(b) it is labeled as existential configuration and it has exactly one universal configuration child (or one leaf configuration node).

Additionally, it has an r rooted configuration tree as child and the CCell on the r node denotes a state of the machine that is existential if the configuration node is labeled as existential or that is universal if the configuration node is labeled as universal.[17] So, we introduce predicates $\text{state}_\exists$ and $\text{state}_\forall$, as well as we extend the handling of the idb-predicates Leaf-$\text{CT}_\forall$, Leaf-$\text{CT}_\exists^{right}$, and Leaf-$\text{CT}_\exists^{left}$ by the following rules

$$\text{state}_\exists(x) \leftarrow r(x), \theta_i(x)$$

for all $i \in$ CCells where i is a configuration cell of an existential state, and

$$\text{state}_\forall(x) \leftarrow r(x), \theta_j(x)$$

for all $j \in$ CCells where j is a configuration cell of a universal state, and finally, we add

$$\begin{aligned}
\text{CT}_\exists^{left}(x) &\leftarrow \text{Leaf-CT}_\exists^{left}(x)\\
\text{CT}_\exists^{left}(x) &\leftarrow \text{state}_\exists(x), \mathbf{label}_{\text{CT}_\exists^{left}}(x), \mathbf{child}(x, x_r), r(x_r), \mathbf{child}(x, x_a), \text{CT}_\forall(x_a),\\
\text{CT}_\exists^{right}(x) &\leftarrow \text{Leaf-CT}_\exists^{right}(x)\\
\text{CT}_\exists^{right}(x) &\leftarrow \text{state}_\exists(x), \mathbf{label}_{\text{CT}_\exists^{right}}(x), \mathbf{child}(x, x_r), r(x_r), \mathbf{child}(x, x_a), \text{CT}_\forall(x_a)\\
\text{CT}_\forall(x) &\leftarrow \text{Leaf-CT}_\forall(x)\\
\text{CT}_\forall(x) &\leftarrow \text{state}_\forall(x), \mathbf{label}_{\text{CT}_\forall}(x), \mathbf{child}(x, x_r), r(x_r),\\
&\qquad \mathbf{child}(x, x_1), \text{CT}_\exists^{left}(x_1), \mathbf{child}(x_1, x_{1_r}), r(x_{1_r}), \theta_i(x_{1_r}),\\
&\qquad\qquad \mathbf{child}(x, x_2), \text{CT}_\exists^{right}(x_2), \mathbf{child}(x_2, x_{2_r}), r(x_{2_r}), \theta_j(x_{2_r})
\end{aligned}$$

for all $i \neq j \in \{1, \dots k\}$. Observe that a node v is marked with $\text{CT}_\exists^{left}$, $\text{CT}_\exists^{right}$, or $\text{CT}_\forall$ if its subtree rooted by v satisfies the conditions (2) – (8) and (10).

[17] Recall, the rank of $\text{CT}_\exists^{left}$, $\text{CT}_\exists^{right}$, and $\text{CT}_\forall$ is $ar(\text{CT}_\exists^{left}) = 2$, $ar(\text{CT}_\exists^{right}) = 2$, and $ar(\text{CT}_\forall) = 3$.

Now, to ensure condition (9) we fix $i \in$ CCells that represents the configuration $(\sqcup, ((q_0, \sqcup), (q_0, \sqcup, S)))$ and add the following rules

$$\text{Start-CT}(x) \leftarrow \text{CT}_\forall(x), \mathbf{child}(x, x_r), r(x_r), \theta_i(x_r)$$
$$\text{Start-CT}(x) \leftarrow \text{CT}^{\mathit{left}}_\exists(x), \mathbf{child}(x, x_r), r(x_r), \theta_i(x_r)$$

to $\mathcal{P}_1$. It is not forbidden that more than one node of the computation tree carries the marker as start configuration node, but the topmost configuration node has to be marked. And therefore, we add the rule

$$\mathit{structure}(x_\top) \leftarrow \mathbf{label}_\top(x_\top), \mathbf{child}(x_\top, x_{\text{CT}}), \text{Start-CT}(x_{\text{CT}})$$

and obtain a program $\mathcal{P}_1$ such that a query $Q' = (\mathcal{P}_1, \mathit{structure})$ yields **yes** on an input tree T if and only if T satisfies conditions (1)–(10), that is, if and only if it is structured as an encoded computation tree of $\mathtt{A}_w$.

To complete the demanded query $Q = (\mathcal{P}, \mathit{Ans})$, it remains to extend the program $\mathcal{P}_1$ in a way that Q accepts the tree if the *structure* predicate is true for its root and the encoded configurations do not violate the transition relation. For the beginning, let $\mathcal{P}$ consist of all rules of $\mathcal{P}_1$. To shorten the query program, we mark all configuration nodes with the predicate CT by adding the following rules.

$$\text{CT}(x) \leftarrow \text{CT}_\forall(x) \qquad \text{Leaf-CT}(x) \leftarrow \text{Leaf-CT}_\forall(x)$$
$$\text{CT}(x) \leftarrow \text{CT}^{\mathit{left}}_\exists(x) \qquad \text{Leaf-CT}(x) \leftarrow \text{Leaf-CT}^{\mathit{left}}_\exists(x)$$
$$\text{CT}(x) \leftarrow \text{CT}^{\mathit{right}}_\exists(x) \qquad \text{Leaf-CT}(x) \leftarrow \text{Leaf-CT}^{\mathit{right}}_\exists(x)$$

Since the upcoming rules are rather large, we introduce short hands as binary predicates.[18] First, we define a predicate $\text{Succ}(x_{r_1}, x_{r_2})$ that is true for two nodes x_{r_1} and x_{r_2} if they are root nodes of successive encoded configuration trees.

$$\text{Succ}(x_{r_1}, x_{r_2}) := \left\{ \begin{array}{c} r(x_{r_1}), r(x_{r_2}), \text{CT}(s_1), \text{CT}(s_2), \\ \mathbf{child}(s_1, s_2), \mathbf{child}(s_1, x_{r_1}), \mathbf{child}(s_2, x_{r_2}) \end{array} \right\}$$

The next predicate $\mathit{SameLevel}_i(x_{s_1}, x_{s_2})$ for an $i > 0$ states for two nodes x_{s_1} and x_{s_2} that they are on the same level i in the configuration tree of two successive encoded configuration trees.

$$\mathit{SameLevel}_i(x_{s_1}, x_{s_2}) := \{s(x_{s_1}), s(x_{s_2})\} \cup \text{Succ}(x_{r_1}, x_{r_2}) \cup \mathbf{child}^i(x_{r_1}, x_{s_1}) \cup \mathbf{child}^i(x_{r_2}, x_{s_2})$$

The predicate $\mathit{SameLevel}^{LR}_i(x_{s_1}, x_{s_2})$ extends the predicate $\mathit{SameLevel}_i(x_{s_1}, x_{s_2})$ by the following property: The nodes x_{s_1} and x_{s_2} have to be both the left or both the right child of their parent.

$$\mathit{SameLevel}^{LR}_i(x_{s_1}, x_{s_2}) := \begin{array}{c} \mathit{SameLevel}_i(x_{s_1}, x_{s_2}) \cup \\ \{\mathbf{child}(x_{s_1}, x_{p_1}), p(x_{p_1}), \mathbf{child}(x_{s_2}, x_{p_2}), p(x_{p_2}), \\ \mathbf{desc}(x_{p_1}, x_{t_1}), 1(x_{t_1}), \mathbf{desc}(x_{p_2}, x_{t_2}), 1(x_{t_2})\} \\ \cup\ \mathbf{child}^{i+4}(z, x_{t_1}) \cup \mathbf{child}^{i+5}(z, x_{t_2}) \end{array}$$

[18] This does not mean that our datalog program is no longer a monadic program, in fact, we use these predicates for replacements in the rule to increase the readability of the whole rule. Variables occurring in the definition of the predicate, but not in the head, have to be renamed in a later context if it is necessary.

Observe that the node z is the configuration node of the predecessor configuration or its parent node and so, for the initial configuration at the top of the encoded computation tree, the extra buffering node above is necessary. Furthermore, this is the only point during the reduction where the **desc** predicate is actually indispensable; we use it to guess whether the nodes are left or right children. In particular, if the nodes x_{t_1} and x_{t_2} do not indicate the same left- or right-orientation then the distance to z is not $i+4$ for the predecessor and $i+5$ for the successor and a valuation of the rule will not be possible. Even another labeling of the encoding tree that tells us directly whether a child is the left or the right one seems to be impossible because it implies a rule for every path through the configuration tree; that leads to 2^n rules and this would avoid a reduction in time polynomial in n and the size of the automaton.

Now, we are able to introduce a predicate $\mathit{SameCell}(x_{s_1}, x_{s_2})$ that states for two skeleton nodes x_{s_1} and x_{s_2} reflecting the same cell of successive encoded configuration cell sequences; those cells are at depth n of any configuration tree.

$$\mathit{SameCell}(x_{s_1}, y_{s_2}) := \bigcup_{1 \leqslant i \leqslant n-1} \big(\{\mathbf{child}(x_i, x_{i+1}), \mathbf{child}(y_i, y_{i+1})\} \cup \mathit{SameLevel}_i^{LR}(x_i, y_i)\big) \cup \{\mathbf{child}(x_{n-1}, x_{s_1}), \mathbf{child}(y_{n-1}, y_{s_2})\} \cup \mathit{SameLevel}_n^{LR}(x_{s_1}, y_{s_2})$$

Next, we use the idb predicate δ to denote that a configuration cell meshes with its predecessor configuration cell with respect to the transition relation. So, for every tuple $(i,j) \in V(\mathtt{A}_w)$ we add the following rule

$$\delta(x_{s_2}) \leftarrow \mathit{SameCell}(x_{s_1}, x_{s_2}), \mathbf{child}(x_{s_1}, x_{m_1}), m(x_{m_1}), m_{k=i}(x_{m_1}), \mathbf{child}(x_{s_2}, x_{m_2}), m(x_{m_2}), m_{k=j}(x_{m_2})$$

to $\mathcal{P}$. To verify the correctness of this rule, recall that the m-labeled node v of a 'me' cell configuration gadget is already marked with $m_{k=i}(v)$ if its gadget encodes the configuration cell i. Now, we have to verify that every configuration cell of the encoded sequence respects the transition relation regarding its predecessor configuration cell and propagate this information to the configuration node by the following rules.

$$\delta(x) \leftarrow \mathbf{child}(x, x_l), s_L(x_l), \delta(x_l), \mathbf{child}(x, x_r), s_R(x_r), \delta(x_r)$$
$$\delta(x) \leftarrow \mathrm{CT}(x), \mathbf{child}(x, x_r), r(x_r), \delta(x_r)$$

The next step is to collect the information that every configuration node is a valid successor up to the top of the tree and we obtain that a configuration node v is marked with Δ if the subtree rooted at v is a suffix of a valid computation tree.

$$\Delta(x) \leftarrow \text{Leaf-CT}(x)$$
$$\Delta(x) \leftarrow \mathrm{CT}_\exists^{\mathit{left}}(x), \mathbf{child}(x, x_a), \mathrm{CT}_\forall(x_a), \Delta(x_a), \delta(x_a)$$
$$\Delta(x) \leftarrow \mathrm{CT}_\exists^{\mathit{right}}(x), \mathbf{child}(x, x_a), \mathrm{CT}_\forall(x_a), \Delta(x_a), \delta(x_a)$$
$$\Delta(x) \leftarrow \mathrm{CT}_\forall(x), \mathbf{child}(x, x_1), \mathrm{CT}_\exists^{\mathit{left}}(x_1), \Delta(x_1), \delta(x_1), \mathbf{child}(x, x_2), \mathrm{CT}_\exists^{\mathit{right}}(x_2), \Delta(x_2), \delta(x_2)$$

Clearly, if the topmost configuration tree is an initial configuration and marked with Δ then we know that the input tree represents a valid accepting computation of $\mathtt{A}_w$. To

this end, we conclude the construction by adding the rule

$$Ans(x) \leftarrow structure(x), \mathbf{child}(x, x_{CT}), \Delta(x_{CT})$$

and obtain the demanded query $Q = (\mathcal{P}, Ans)$ within polynomial time; that finishes the proof of Theorem 6.16. □

The hardness of the emptiness problem implies the following corollary by using the same proof as for Corollary 4.9

Corollary 6.18.

(a) The equivalence problem for Boolean mDatalog($\tau_u^{\mathbf{desc}}$) on ranked labeled unordered trees is 2Exptime*-hard.*

(b) The query containment problem for Boolean mDatalog($\tau_u^{\mathbf{desc}}$) on ranked labeled unordered trees is 2Exptime*-hard.* ⌟

Finally, we obtain the following completeness result.

Corollary 6.19. *Let τ be a schema with $\tau_u^{\mathbf{desc}} \subseteq \tau \subseteq \tau_u^{\mathbf{root},\mathbf{leaf},\mathbf{desc}}$.*

(a) The emptiness problem for mDatalog(τ) on ranked labeled unordered trees is complete for 2Exptime*.*

(b) The equivalence problem for mDatalog(τ) on ranked labeled unordered trees is complete for 2Exptime*.*

(c) The query containment problem for mDatalog(τ) on ranked labeled unordered trees is complete for 2Exptime*.*

Proof. All problems are hard for 2Exptime by Theorem 6.16 and by Corollary 6.18. By Theorem 6.1 the QCP is solvable in 2Exptime and therefore complete for 2Exptime. Let $\mathcal{A}$ be the algorithm proposed by Theorem 6.1. Now, we solve the emptiness problem on input Q by solving $\mathcal{A}(Q, Q_\varnothing)$ where $Q_\varnothing$ is an unsatisfiable query (c.f. Example 2.10) and the equivalence problem on input (Q_1, Q_2) by solving the problem that asks: Hold $QCP((Q_1, Q_2))$ and $QCP((Q_2, Q_1))$? Thus, all problems are solvable in 2Exptime and in the end complete for 2Exptime. □

The next step is to extend the result to ordered ranked trees. By the aforementioned 2Exptime-membership, Corollary 6.19, and the fact that $\tau_u^{\mathbf{desc}} \subset \tau_o^{\mathbf{child},\mathbf{desc}}$, we can state

Corollary 6.20. *Let τ be a schema with $\tau_o^{\mathbf{child},\mathbf{desc}} \subseteq \tau \subseteq \tau_{GK}^{\mathbf{child},\mathbf{desc}}$.*

The emptiness problem, the equivalence problem, and the query containment problem for mDatalog(τ) on ranked labeled ordered trees is complete for 2Exptime*.* ⌟

Unfortunately, this does not help for the considered problems for mDatalog($\tau_o^{\mathbf{desc}}$), since for eliminating the **child** predicate, a rewriting similar to Gottlob and Koch (cf. Corollary 4.13) is not possible in the presence of **desc** predicates.

6.5 On Hardness on Unranked Trees

In this section, we consider the hardness of datalog queries using the descendant axis on unranked trees. For mDatalog on unordered trees, we prove the matching lower bound regarding the query containment and the equivalence problem. For ordered trees using the schema $\tau_{GK}^{\mathbf{desc}}$, we additionally establish the matching lower bound for the emptiness problem.

Theorem 6.21. *The query containment problem for Boolean mDatalog($\tau_u^{\mathbf{desc}}$) on unranked labeled unordered trees is* 2EXPTIME*-hard.*

Proof. We prove the theorem by using and extending the proof of Theorem 6.16, so we establish a reduction from the word acceptance problem of exponential space bounded alternating Turing machines to the QCP for mDatalog($\tau_u^{\mathbf{desc}}$) on unranked labeled unordered trees. More precisely, we give a polynomial time reduction to the complement of this QCP. For a given ATM $\mathtt{A}_w$ that is normalized and composed of the original ATM $\mathtt{A}$ and its input word w, we construct within polynomial time a finite unranked alphabet Σ_{ur} and two Boolean mDatalog($\tau_{u,\Sigma_{ur}}^{\mathbf{desc}}$)-queries Q_1 and Q_2, such that

$$
\begin{array}{lcl}
w \in L(A) & \iff & \text{there is an accepting computation tree for } \mathtt{A}_w \\
& \iff & \text{there exists an unordered } \Sigma_{ur}\text{-labeled tree } T \text{ such that} \\
& & \qquad Q_1(T) = \textbf{yes} \text{ and } Q_2(T) = \textbf{no} \\
& \iff & Q_1 \not\subseteq Q_2.
\end{array}
$$

Recall the reduction from Theorem 6.16, the utilized ranked alphabet Σ_T, and the obtained program $\mathcal{P}$ in mDatalog($\tau_u^{\mathbf{desc}}$) on ranked trees. We choose the unranked alphabet Σ_{ur} as the unranked version of Σ_T, to be precise we set $\Sigma_{ur} := \{\alpha |\, \alpha \in \Sigma_T\}$. Furthermore, we set $Q_1 := (\mathcal{P}, Ans)$, that is, the query constructed during the former reduction. So, Q_1 stands for the "necessary properties" of the encoded computation tree. Since the alphabet is no longer ranked, we cannot avoid that a node has more than the planned children, but we can forbid that the redundant children have other labels and falsify the computation. Therefore, all that remains is to construct a query Q_2 in mDatalog($\tau_u^{\mathbf{desc}}$) such that Q_2 describes "forbidden properties". A tree with such properties does not describe an encoded computation tree. To this end, we check for forbidden labels on child nodes, a child of an s-labeled node, for example, must not be labeled with $\mathrm{CT}_\forall$, and we have to test that there are no two paths encoding inconsistent information. Thus, the query $Q_2 = (\mathcal{P}_2, \textbf{reject})$ will yield to **yes** on an input tree if at least one of the following facts are true.

(1) A non root node is labeled by $\top$.

(2) The root has a child that is not labeled by a CT-label.

(3) A non halting existential configuration node has a child labeled with a symbol not in $\{r, \mathrm{CT}_\forall, \text{Leaf-CT}_\forall\}$.

(4) An non halting universal configuration node has a child labeled with a symbol not in $\{r, \mathrm{CT}_\exists^{right}, \mathrm{CT}_\exists^{left}, \text{Leaf-CT}_\exists^{right}, \text{Leaf-CT}_\exists^{left}\}$.

(5) A halting configuration node has a child labeled with a symbol that is not r.

(6) An r labeled node has a child labeled with a symbol that is not s.

(7) An s labeled node has a child labeled with a symbol not in $\{p, s, s_{\mathbf{leaf}}\}$.

(8) An $s_{\mathbf{leaf}}$ labeled node has a child labeled with a symbol not in $\{p, m\}$.

(9) A p or m labeled node has a child labeled with a symbol not in $\{0, 1\}$.

(10) A 0 or 1 labeled node has a child labeled with a symbol not in $\{0, 1, \bot\}$.

(11) A $\bot$ labeled node has a child.

(12) A p (or an m) labeled node has a descendant that is labeled $\bot$ with distance not equal to three (not equal to $k+1$), or is not a prefix of a valid gadget.

(13) There exists a path in a configuration tree from the r labeled node to an $s_{\mathbf{leaf}}$ of length not equal to n.

(14) Any node has two children fulfilling the same role, but encoding different information.

Obviously, the conditions (1) – (13) reflect the underlying structure. Additionally, an illustration to condition (12) is given with Figure 6.6 (a). Condition (14) reflects the consistency of the encoding and enforces the following; if there are two configurations as children of a node in the computation tree, both universal, both left – or right – existential, then they have to provide exactly the same information during the computation. This includes the contained configuration trees, navigation gadgets, and so on, which can have different copies or copies of prefixes. Intuitively, it is clear that it does not matter if a node has additional children, but they must not provide wrong information; since every rule uses a maximum distance of $3 + n + k$, it suffices to have a fixed look ahead inside the encoded configuration (cf. Figure 6.6 (b)). Now, it is comprehensible that the query Q_2 fulfilling condition (1)–(14) yields **no** on a tree T and Q_1 yields **yes** on the same tree if and only if T is an encoded accepting computation of $\mathtt{A}_w$.

For the beginning, let $\mathcal{P}_2$ consist of all rules of $\mathcal{P}$. We only consider trees T with $Q_1(T) = \mathbf{yes}$, otherwise we have in any way $Q_1 \subseteq Q_2$, which is enough for the reduction. To propagate any detected violation to the root node of the input tree, we propagate the **reject** predicate from any node to the root by adding the following rule

$$\mathbf{reject}(x) \leftarrow \mathbf{child}(x, x_1), \mathbf{reject}(x_1)$$

to $\mathcal{P}_2$. We reflect condition (1) by adding the rule

$$\mathbf{reject}(x) \leftarrow \mathbf{child}(x, x_1), \mathbf{label}_{\top}(x_1).$$

Since $Q_1(T) = \mathbf{yes}$, we know the root is labeled with $\top$ and so, we mirror condition (2) by the rule

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{\top}(x), \mathbf{child}(x, x_1), \mathbf{label}_{\alpha}(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{\mathrm{CT}_{\forall}, \mathrm{CT}_{\exists}^{left}, \mathrm{CT}_{\exists}^{right}\}$.

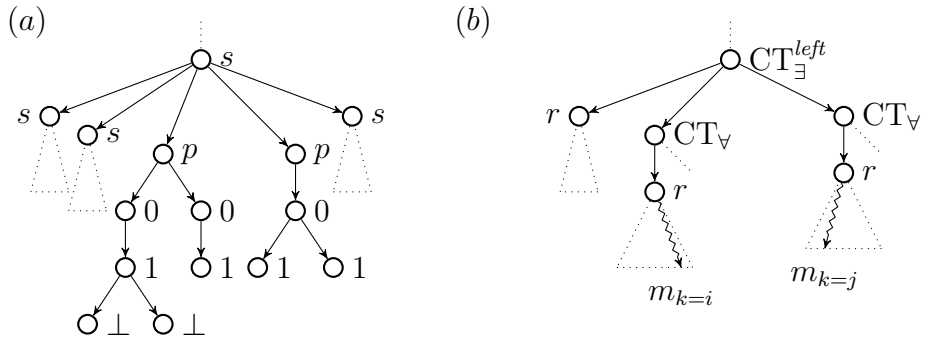

Figure 6.6: (a) An example of allowed "extentions" of the encoded computation tree, considered at a navigation gadget that can exist multiple times where a copy also can be reduced to a prefix. (b) If the nodes marked by $m_{k=i}$ and $m_{k=j}$ have the same path through their configuration tree, that is, the same sequence of left and right children, then i must be equal to j.

To verify condition (3) we add the rules

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{\mathrm{CT}_\exists^{left}}(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{\mathrm{CT}_\exists^{right}}(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{r, \mathrm{CT}_\forall, \text{Leaf-CT}_\forall\}$.

To verify condition (4) we add the rule

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{\mathrm{CT}_\forall}(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{r, \mathrm{CT}_\exists^{right}, \mathrm{CT}_\exists^{left}, \text{Leaf-CT}_\exists^{right}, \text{Leaf-CT}_\exists^{left}\}$.

To verify condition (5) we add the rules

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{\text{Leaf-CT}_\forall}(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{\text{Leaf-CT}_\exists^{left}}(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{\text{Leaf-CT}_\exists^{right}}(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{r\}$.

To verify condition (6) we add the rule

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_r(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{s\}$.

To verify condition (7) we add the rule

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_s(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{p, s, s_{\mathbf{leaf}}\}$.

To verify condition (8) we add the rule

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_{s_{\mathbf{leaf}}}(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{p, m\}$.

To verify condition (9) we add the rules

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_p(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_m(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{0, 1\}$.

To verify condition (10) we add the rules

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_0(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_1(x), \mathbf{child}(x, x_1), \mathbf{label}_\alpha(x_1)$$

for every $\alpha \in \Sigma_{ur} \setminus \{0, 1, \bot\}$.

To verify condition (11) we add the rule

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_\bot(x), \mathbf{child}(x, x_1)$$

to $\mathcal{P}_2$.

To verify condition (12), we assume that conditions (9) and (10) are not fulfilled. This implies, the only possible labels at nodes descending a node labeled with p or m are 0, 1, and $\bot$. So, we add the following rules that ensure that no **leaf** labeled path exists that is too short or too long, that is a path with a node labeled with 0 or 1 on position three for a navigation gadget and on position $k+1$ for a 'me' cell gadget. Thus, we add for the navigation gadget the following rules

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_p(x), \mathbf{child}(x, x_1), \mathbf{label}_\bot(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_p(x), \mathbf{child}^2(x, x_1), \mathbf{label}_\bot(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_p(x), \mathbf{child}^3(x, x_1), \mathbf{label}_0(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_p(x), \mathbf{child}^3(x, x_1), \mathbf{label}_1(x_1)$$

and for the 'me' gadget, we add

$$\mathbf{reject}(x) \leftarrow \mathbf{label}_m(x), \mathbf{child}(x, x_1), \mathbf{label}_\bot(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_m(x), \mathbf{child}^2(x, x_1), \mathbf{label}_\bot(x_1)$$
$$\vdots$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_m(x), \mathbf{child}^k(x, x_1), \mathbf{label}_\bot(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_m(x), \mathbf{child}^{k+1}(x, x_1), \mathbf{label}_0(x_1)$$
$$\mathbf{reject}(x) \leftarrow \mathbf{label}_m(x), \mathbf{child}^{k+1}(x, x_1), \mathbf{label}_1(x_1).$$

Recall that $\mathbf{child}^i(x, y)$ is a short hand for the set of atoms denoting y as a descendant of x in the i-th generation.

In the same way, we reflect condition (13) which says that there exists a path in a configuration tree from the r labeled node to an $s_{\mathbf{leaf}}$ of length not equal to n. We know by conditions (6), (7), (9) – (11) that it suffices to test if there is a shorter path ending

on an $s_{\mathbf{leaf}}$ labeled node, or if there exists a path of length n ending with an s labeled node. Therefore, we add the rules

$$\begin{aligned}
\mathbf{reject}(x) &\leftarrow \mathbf{label}_r(x), \mathbf{child}(x, x_1), \mathbf{label}_{s_{\mathbf{leaf}}}(x_1)\\
\mathbf{reject}(x) &\leftarrow \mathbf{label}_r(x), \mathbf{child}^2(x, x_1), \mathbf{label}_{s_{\mathbf{leaf}}}(x_1)\\
&\vdots\\
\mathbf{reject}(x) &\leftarrow \mathbf{label}_r(x), \mathbf{child}^{n-1}(x, x_1), \mathbf{label}_{s_{\mathbf{leaf}}}(x_1)\\
\mathbf{reject}(x) &\leftarrow \mathbf{label}_r(x), \mathbf{child}^{n}(x, x_1), \mathbf{label}_{s}(x_1)
\end{aligned}$$

to $\mathcal{P}_2$.

Finally, we consider condition (14) and we start by verifying all neighboring navigation gadgets. By condition (12) we already know every navigation gadget is a valid navigation gadget or the prefix thereof . By the following rules, we detect if they are in conflict.

$$\begin{aligned}
\mathbf{reject}(x_s) \leftarrow \mathbf{child}(x_s, x_{p_1}), \mathbf{child}(x_s, x_{p_2}), \mathbf{label}_p(x_{p_1}), \mathbf{label}_p(x_{p_2}),\\
\mathbf{child}(x_{p_1}, x_1), \mathbf{label}_1(x_1), \mathbf{child}(x_{p_2}, x_0), \mathbf{label}_0(x_0)\\
\mathbf{reject}(x_s) \leftarrow \mathbf{child}(x_s, x_{p_1}), \mathbf{child}(x_s, x_{p_2}), \mathbf{label}_p(x_{p_1}), \mathbf{label}_p(x_{p_2}),\\
\mathbf{child}^2(x_{p_1}, x_1), \mathbf{label}_1(x_1), \mathbf{child}^2(x_{p_2}, x_0), \mathbf{label}_0(x_0)
\end{aligned}$$

The same holds for the 'me' cell configuration gadget and therefore, we add the rules

$$\begin{aligned}
\mathbf{reject}(x_s) \leftarrow \mathbf{child}(x_s, x_{m_1}), \mathbf{child}(x_s, x_{m_2}), \mathbf{label}_m(x_{m_1}), \mathbf{label}_m(x_{m_2}),\\
\mathbf{child}(x_{m_1}, x_1), \mathbf{label}_1(x_1), \mathbf{child}(x_{m_2}, x_0), \mathbf{label}_0(x_0)\\
\mathbf{reject}(x_s) \leftarrow \mathbf{child}(x_s, x_{m_1}), \mathbf{child}(x_s, x_{m_2}), \mathbf{label}_m(x_{m_1}), \mathbf{label}_m(x_{m_2}),\\
\mathbf{child}^2(x_{m_1}, x_1), \mathbf{label}_1(x_1), \mathbf{child}^2(x_{m_2}, x_0), \mathbf{label}_0(x_0)\\
\vdots\\
\mathbf{reject}(x_s) \leftarrow \mathbf{child}(x_s, x_{m_1}), \mathbf{child}(x_s, x_{m_2}), \mathbf{label}_m(x_{m_1}), \mathbf{label}_m(x_{m_2}),\\
\mathbf{child}^k(x_{m_1}, x_1), \mathbf{label}_1(x_1), \mathbf{child}^k(x_{m_2}, x_0), \mathbf{label}_0(x_0)
\end{aligned}$$

Now, we compare the configurations; that is done analogously to the definition of the short hand predicate *SameCell* in the previous proof, but without the offset that was used to reach the successor configuration. So, we first define the predicates *EquiLevel*, $\mathit{EquiLevel}^{LR}$, and *EquiCell*, stating that two nodes are in the equivalent level, are both a left or both a right child, and, by the latter, denote equivalent cells.

$$\begin{aligned}
\mathit{EquiLevel}_i(x_{s_1}, x_{s_2}) := \quad & \mathbf{child}^2(x, x_{r_1}) \cup \mathbf{child}^2(x, x_{r_2}) \cup \{r(x_{r_1}), r(x_{r_2})\}\\
& \cup \mathbf{child}^i(x_{r_1}, x_{s_1}) \cup \mathbf{child}^i(x_{r_2}, x_{s_2}) \cup \{s(x_{s_1}), s(x_{s_2})\}
\end{aligned}$$

The predicate $\mathit{EquiLevel}_i^{LR}(x_{s_1}, x_{s_2})$ extends the predicate $\mathit{EquiLevel}_i(x_{s_1}, x_{s_2})$ by the following property: The nodes x_{s_1} and x_{s_2} have to be both the left or both the right child

of their parent.

$$\mathit{EquiLevel}_i^{LR}(x_{s_1}, x_{s_2}) := \begin{array}{c} \mathit{EquiLevel}_i(x_{s_1}, x_{s_2}) \cup \\ \{\mathbf{child}(x_{s_1}, x_{p_1}), p(x_{p_1}), \mathbf{child}(x_{s_2}, x_{p_2}), p(x_{p_2}), \\ \mathbf{desc}(x_{p_1}, x_{t_1}), 1(x_{t_1}), \mathbf{desc}(x_{p_2}, x_{t_2}), 1(x_{t_2})\} \\ \cup \mathbf{child}^{i+4}(z, x_{t_1}) \cup \mathbf{child}^{i+4}(z, x_{t_2}) \end{array}$$

And finally, we define *EquiCell* that is true for two nodes denoting configuration cells that encode the same cell of the automaton. Note, that the predicate is reflexive.

$$\begin{aligned} \mathit{EquiCell}(x_{s_1}, y_{s_2}) := \bigcup_{1 \leqslant i \leqslant n-1} & \left(\{\mathbf{child}(x_i, x_{i+1}), \mathbf{child}(y_i, y_{i+1})\} \cup \mathit{EquiLevel}_i^{LR}(x_i, y_i)\right) \\ & \cup \{\mathbf{child}(x_{n-1}, x_{s_1}), \mathbf{child}(y_{n-1}, y_{s_2})\} \cup \mathit{EquiLevel}_n^{LR}(x_{s_1}, y_{s_2}) \end{aligned}$$

To verify the value k, we utilize the predicate $m_{k=i}$ for every $i \in \{1, \ldots, k\}$ given by a positive evaluation of query Q_1, and compare them for all $i, j \in \{1, \ldots, k\}$ with $i \neq j$ by the following rules

$$\begin{aligned} \mathbf{reject}(x) \leftarrow & \mathbf{child}(x, x_{\mathrm{CT}_1}), \mathbf{child}(x, x_{\mathrm{CT}_2}), \\ & \mathit{type}(x_{\mathrm{CT}_1}), \mathit{type}(x_{\mathrm{CT}_2}), \mathbf{child}^{n+1}(x_{\mathrm{CT}_1}, x_{s_1}), \mathbf{child}^{n+1}(x_{\mathrm{CT}_2}, x_{s_2}), \\ & \mathit{EquiCell}(x_{s_1}, x_{s_2}), \mathbf{child}(x_{s_1}, x_{m_1}), m(x_{m_1}), m_{k=i}(x_{m_1}), \\ & \mathbf{child}(x_{s_2}, x_{m_2}), m(x_{m_2}), m_{k=j}(x_{m_2}) \end{aligned}$$

for every $\mathit{type} \in \{\mathrm{CT}_\forall, \text{Leaf-CT}_\forall, \mathrm{CT}_\exists^{\mathit{left}}, \text{Leaf-CT}_\exists^{\mathit{left}}, \mathrm{CT}_\exists^{\mathit{right}}, \text{Leaf-CT}_\exists^{\mathit{right}}\}$.

Now, it is ensured that two configurations in the same role, provide different information, so the demanded query is defined by $Q_2 = (\mathcal{P}_2, \mathbf{reject})$.

Observe, by Q_1 we evaluate the computation tree by starting in the halting configurations, so it does not matter if a configuration has a successor configuration twice or if these successor configurations themselves have different successor configurations. In this case it suffices if one subtree leads to accepting configurations on the leaves of an appropriate subtree. □

We use the result and a short reduction to extend the hardness of the query containment problem to the equivalence problem.

Corollary 6.22. *The equivalence problem for Boolean mDatalog($\tau_u^{\mathbf{desc}}$) on unranked labeled unordered trees is* 2Exptime*-hard.*

Proof. The proof proceeds by a reduction from the query containment problem for Boolean mDatalog($\tau_u^{\mathbf{desc}}$) on unranked labeled unordered trees which is shown to be 2Exptime-hard by Theorem 6.21.

The task is to decide for two given Boolean mDatalog($\tau_{u,\Sigma}^{\mathbf{desc}}$)-queries $Q_1 = (\mathcal{P}_1, \mathit{Ans}_1)$ and $Q_2 = (\mathcal{P}_2, \mathit{Ans}_2)$ whether it holds that $Q_1 \subseteq Q_2$. This will be done by verifying whether $(Q_1 \cup Q_2) \equiv Q_2$ holds. To this end, we define the query

$$(Q_1 \cup Q_2) := (\mathcal{P}_1 \cup \mathcal{P}_2 \cup \{\mathit{Ans}(x) \leftarrow \mathit{Ans}_1(x),\ \mathit{Ans}(x) \leftarrow \mathit{Ans}_2(x)\}, \mathit{Ans})$$

Now, the following is true.

- If $(Q_1 \cup Q_2) \equiv Q_2$ then there is no tree T such that $Q_1(T) = \textbf{yes}$ and $Q_2(T) = \textbf{no}$. This states $Q_1 \subseteq Q_2$.
- If $(Q_1 \cup Q_2) \not\equiv Q_2$ then there is a tree T such that $(Q_1 \cup Q_2)(T) = \textbf{yes}$ and $Q_2(T) = \textbf{no}$. Thus, $Q_1(T) = \textbf{yes}$ and therefore $Q_1 \not\subseteq Q_2$.

  The converse $(Q_1 \cup Q_2)(T) = \textbf{no}$ and $Q_2(T) = \textbf{yes}$ contradicts the fact that $Q_2(T) = \textbf{yes}$ implies $(Q_1 \cup Q_2)(T) = \textbf{yes}$.

Finally, a suitable reduction within polynomial time in the size of Q_1 and Q_2 has been presented finishing the proof of Corollary 6.22. □

With reference to the monotonicity of datalog, it seems to be difficult to check if an unordered tree respects the rank constraints. But for ordered trees and using the schema $\tau_{GK}^{\textbf{child},\textbf{desc}}$ we prove the following result.

Proposition 6.23. *The emptiness problem for Boolean mDatalog($\tau_{GK}^{\textbf{child},\textbf{desc}}$) on ordered unranked labeled trees is* 2EXPTIME*-hard.*

Proof. Again, we prove the theorem by using and extending the proof of Theorem 6.16; so we establish a reduction from the word acceptance problem of exponential space bounded alternating Turing machines to the complement of the emptiness problem for mDatalog($\tau_{GK}^{\textbf{child},\textbf{desc}}$) on ordered unranked labeled trees. For a given ATM $\mathtt{A}_w$ that is normalized and composed of the original ATM $\mathtt{A}$ and its input word w, we construct within polynomial time a finite unranked alphabet Σ_{ur} and a Boolean mDatalog($\tau_{GK}^{\textbf{child},\textbf{desc}}$)-query Q such that

$$w \in L(A) \qquad \Longleftrightarrow \qquad Q \neq \varnothing.$$

Recall the reduction from Theorem 6.16, the utilized ranked alphabet Σ_T, and the obtained query $Q' = (\mathcal{P}, Ans')$ in mDatalog($\tau_u^{\textbf{desc}}$) on ranked trees. We choose the unranked alphabet $\Sigma_{ur} := \{\alpha |\, \alpha \in \Sigma_T\}$.

Observe, it holds that $Q' \neq \varnothing$ if and only if there exists a ranked unordered tree T such that T represents an encoded accepting computation tree of $\mathtt{A}$. Since $\tau_u^{\textbf{desc}} \subset \tau_{GK}^{\textbf{child},\textbf{desc}}$, it implies that Q' is not empty on ordered ranked trees if and only if there exists a ranked ordered tree T that represents an encoded accepting computation tree of $\mathtt{A}_w$. Now, the alphabet is no longer ranked. Lemma 3.22 provides a unary mDatalog($\tau_{GK,\Sigma}$)-query $Q_{rk} = (\mathcal{P}_{rk}, P_{rk})$ such that for every unranked ordered σ-labeled tree T it holds that

$$\begin{array}{c} P_{rk}(v) \in \mathcal{T}^{\omega}_{\mathcal{P}_{rk}}(T) \\ \text{for every node } v \in T \end{array} \qquad \Longleftrightarrow \qquad \begin{array}{c} \text{there exists a ranked ordered } \Sigma\text{-labeled tree } T' \\ \text{such that } \mathcal{S}^M_o(T) = \mathcal{S}^M_o(T') \end{array} .$$

Moreover, we have for every node v in T that $P_{rk}(v) \notin \mathcal{T}^{\omega}_{\mathcal{P}_{rk}}(T)$ if such a tree T' does not exists.

To this end, we define the query

$$Q \quad := \quad (\mathcal{P} \cup \mathcal{P}_{rk} \cup \{\ Ans(x) \leftarrow P_{rk}(x), Ans'(x)\ \}, Ans)$$

that yields **yes**, evaluated on a ordered unranked tree T labeled with symbols from Σ if T respects the alphabet Σ_T and it represents an encoded accepting computation tree for $\mathtt{A}_w$. □

Unfortunately, this approach fails on unordered unranked trees, which is witnessed by Lemma 3.27. For ordered trees in general, we can take over the results from unordered trees since $\tau_u^{\mathbf{desc}} \subset \tau_o^{\mathbf{child},\mathbf{desc}}$. By using the membership results of Section 6.1, we close this chapter by concluding the results for unranked trees in the following corollary.

Corollary 6.24. *Let N be a set with $\{\mathbf{desc}\} \subseteq N \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{desc}\}$ and let M be a set with $\{\mathbf{child}, \mathbf{desc}\} \subseteq M \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}, \mathbf{child}, \mathbf{desc}\}$.*

(a) The emptiness problem for Boolean mDatalog($\tau_{GK}^{\mathbf{child},\mathbf{desc}}$) on ordered unranked labeled trees is 2Exptime*-complete.*

(b) The equivalence problem for Boolean mDatalog(τ_u^N) on unordered unranked labeled trees is 2Exptime*-complete.*

(c) The equivalence problem for Boolean mDatalog(τ_o^M) on ordered unranked labeled trees is 2Exptime*-complete.*

(d) The query containment problem for Boolean mDatalog(τ_u^N) on unordered unranked labeled trees is 2Exptime*-complete.*

(e) The query containment problem for Boolean mDatalog(τ_o^M) on ordered unranked labeled trees is 2Exptime*-complete.* ⌟

Chapter 7

Changing the Point of View:
Beyond Set Semantics

In the previous chapters, we considered structures under the so called *set semantics*. That means conjunctive queries as well as datalog queries can be seen as functions from sets into sets. The restriction to set semantics enables a lot of optimization – in particular minimization – and complexity results on various database languages – specifically for conjunctive queries – that are identified as important subclasses of `SQL`.

However, the practical use of `SQL` is defined as queries over multisets of tuples. Actually, in real-world databases, relations are viewed as multisets where a tuple may have multiple occurrences in a relation and in a query's result. These multiple occurrences can give further information evaluable with `SQL` aggregate functions like `Count` or `Sum`. Interpreting set semantics with `SQL` implies a permanent usage of the `distinct` operator to eliminate duplicates in each query and subquery.

In this chapter, we consider the notation of structures and relations under *bag semantics*. Afterwards, we propose a semantics for datalog in the bag theoretic context. We investigate the emptiness problem and the query containment problem for datalog under bag semantics in general as well as for monadic datalog on trees. To do this, we recall the semantics of conjunctive queries under bag semantics in style of the definitions used in [CV93], [IR95], and [JKV06]. We illustrate the examples in this chapter by corresponding statements in `SQL`. As in `SQL` the components of every tuple in a relation are addressed via attribute names, we identify the first column of any relation by the attribute name `A`, the second column by `B`, and so on. Queries in `SQL` are simple or composite statements in `SQL`; therefore, we usually denote a query in `SQL` by `S`, $\texttt{S}_1$ or likewise. The `SQL` statements are obtained from the `SQL` standard and they are tested on PostgreSQL.[1]

To define multisets in relations of relational structures (or databases), each relation $R^{\mathcal{A}}$ of a structure $\mathcal{A}$ over the universe A is defined by a function $f_{R^{\mathcal{A}}} : A^{ar(R)} \to \mathbb{N}$. The function $f_{R^{\mathcal{A}}}(\overline{a})$ denotes the *frequency* of a tuple $\overline{a} = (a_1, \ldots, a_{ar(R)}) \in A^{ar(R)}$ contained in the relation R of $\mathcal{A}$. As a shortcut, we write $(\overline{a})^{f_{R^{\mathcal{A}}}(\overline{a})}$ to denote a tuple and its frequency. Furthermore, we write $\overline{a}^l \in R^{\mathcal{A}}$ to indicate that $f_{R^{\mathcal{A}}}(\overline{a}) = l$.

For the remainder of this chapter we will call such finite structures allowing multiple tuples in the relations *databases*. The universe of a database is called *domain* and consists

[1] To be precise, the test suite was PostgreSQL in version 9.4 installed on Debian GNU/Linux 8.5

of all possible entries. We write $\mathrm{dom}(\mathcal{D})$ to denote the domain of the database $\mathcal{D}$. The subset of the domain that occurs in at least one tuple in the database $\mathcal{D}$ is called *active domain* and denoted by $\mathrm{adom}(\mathcal{D})$.

The function $f_{\mathcal{D}} : \mathrm{dom}(\mathcal{D}) \to \mathbb{N}$ denotes the frequency of the atomic fact $R(\overline{a})$ in $\mathcal{D}$ and is defined by $f_{\mathcal{D}}(R(\overline{a})) := f_{R^{\mathcal{D}}}(\overline{a})$.

Example 7.1. *We consider the database $\mathcal{D}$ with two relations R and S.*

R
a
a

S	
a	c
c	b
c	b

Now, we write $R(a)^2$ or $(a)^2 \in R$ to indicate that there are two rows in R containing a. To describe S, we write $\{(a,c)^1, (c,b)^2\} = S$. To denote that the tuple (a,a) is not contained in S, we write $(a,a) \notin S$, $(a,a)^0 \in S$, or $S(a,a)^0$.

7.1 Conjunctive Queries under Bag Semantics

Before defining the semantics of datalog under bag semantics, we briefly recall the semantics of conjunctive queries under bag semantics.

Let Q be the conjunctive query

$$Ans(u) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_n)$$

of schema τ where $n \in \mathbb{N}_{\geqslant 1}, \{R_1, \ldots, R_n\} \subseteq \tau$ (cf. Section 2.6). Furthermore, let $\mathcal{D}$ be a database of schema τ that is a τ-structure over the universe A allowing multiple occurrences of the same tuple in its relations.

The result of evaluating Q in database $\mathcal{D}$ under bag semantics is defined as follows (cf. [JKV06]):

$$Q(\mathcal{D}) = \left\{ \overline{a}^{f(\overline{a})} \;\middle|\; \begin{array}{c} \overline{a} = (a_1, \ldots, a_{ar(Q)}) \in A^{ar(Q)}, \text{ where } f(\overline{a}) = \sum_{\beta \in B} \prod_{i=1}^{n} f_{R_i^{\mathcal{D}}}(\beta(u_i)) \\ \text{and } B = \left\{ \beta : \mathrm{var}(Q) \to \mathrm{dom}(\mathcal{D}) \;\middle|\; \beta(u) = (a_1, \ldots, a_{ar(Q)}) \right\} \end{array} \right\}$$

Note that the result of Q in $\mathcal{D}$ defines a function $f_{Q(\mathcal{D})} : A^{ar(Q)} \to \mathbb{N}$ indicating the frequency of any tuple $\overline{a} = (a_1, \ldots, a_{ar(Q)})$ in $Q(\mathcal{D})$.

Example 7.2. *Consider the database of Example 7.1 and the following queries.*

Q_1 : $Ans(x) \leftarrow R(x)$. *There is only one valuation β from $\mathrm{var}(Q_1) = \{x\}$ into the domain of $\mathcal{D}$ such that the body atom has frequency bigger than zero. Therefore, the result of evaluating Q_1 in $\mathcal{D}$ under bag semantics is $Q_1(\mathcal{D}) = \{(a)^2\}$.*

The query Q_1 translated into an `SQL` *statement* $\mathtt{S}_1$*:*

```
select R.A from R.
```

Q_2 : $Ans(x) \leftarrow R(x) \wedge R(x)$. *Again, there is only one successful valuation β, but in contrast to set semantics, here the result differs from Q_1: The result of evaluating Q_2 in $\mathcal{D}$ under bag semantics is $Q_2(\mathcal{D}) = \{(a)^4\}$.*

The query Q_2 translated into an SQL *statement* $\mathtt{S}_2$*:*

$$\texttt{select R.A from R, R as R}_1\texttt{ where R.A} = \texttt{R}_1\texttt{.A}$$

or the equivalent statement $\mathtt{S}_{2\mathtt{b}}$*:*

$$\texttt{select R.A from R join R as R}_1\texttt{ on (R.A} = \texttt{R}_1\texttt{.A)}.$$

In any way, it can be seen that we use a temporary table $\mathtt{R}_1$ *to join the table* R *with itself.*

Q_3 : $Ans(x, y) \leftarrow S(x, z) \wedge S(z, y) \wedge S(v, w)$. *It can be easily verified that the result of evaluating Q_3 in $\mathcal{D}$ under bag semantics is $Q_3(\mathcal{D}) = \{(a, b)^6\}$.*

The query Q_3 can be translated into the following SQL *statement* $\mathtt{S}_3$*:*

$$\texttt{select S.A, S}_1\texttt{.B from S, S as S}_1\texttt{, S as S}_2\texttt{ where S.B} = \texttt{S}_1\texttt{.A}$$

7.2 Datalog Queries under Bag Semantics

In this section, we propose a semantics for datalog queries under bag semantics by extending the semantics of conjunctive queries and harmonizing it with the SQL statement WITH RECURSIVE.

In a first step, we extend datalog programs to multisets, saying that a rule r of $\mathcal{P}$ may have multiple occurrences. Similar to the definition of relations the frequency of a rule r is defined by a function $f_{\mathcal{P}}(r) : R \to \mathbb{N}$ where R is the set consisting of all possible rules. We write $r \in \mathcal{P}$ if $f_{\mathcal{P}}(r) > 0$ and $\mathcal{P}_{\text{set}} := \bigcup \{r \mid r \in \mathcal{P}\}$ is the *set reduct* of $\mathcal{P}$.

It is possible, but not intuitive, to define the semantics by using the operator $\mathcal{T}_{\mathcal{P}}$. Thereby, we have to distinguish between the newly and previously added tuples and their frequencies. To avoid this, we introduce the semantics by extending the proof trees of Section 2.5.

Let $\mathcal{P}$ be a datalog program. Furthermore, let $\mathcal{D}$ be a database. For $k, i \in \mathbb{N}$, $i \geqslant 1$, and $a_1, \ldots, a_k \in \text{dom}(\mathcal{D})$, a *proof tree* $T_{\mathcal{P},\mathcal{D}}$ of the fact $P(a_1, \ldots, a_k)^i$ from the datalog program $\mathcal{P}$ and database $\mathcal{D}$ is a finite ordered labeled tree where

- each vertex of the tree is labeled by a symbol of the form a_r^j where a is an atomic fact, $j \in \mathbb{N}_{\geqslant 1}$ and $r \in \mathbb{N}$,
- each leaf is labeled by a_0^j where a is an atomic fact from $atoms(\mathcal{D})$ with an superscript of its frequency in the database $f_{\mathcal{D}}(a) = j$,
- the root is labeled by $P(a_1, \ldots, a_k)^i_{f_r}$ for some $f_r \in \mathbb{N}$, and
- for each non-leaf vertex v with children $v_1, \ldots, v_l$ with $l \in \mathbb{N}$ there exists a rule $r : h(x) \leftarrow b_1(x_1), \ldots, b_l(x_l)$ in $\mathcal{P}$, $i_0, i_1, \ldots, i_l, f_r \in \mathbb{N}_{\geqslant 1}$, $f_1, \ldots, f_l \in \mathbb{N}$, and a valuation β such that:

(1) v is labeled by $h(\beta(x))^{i_0}_{f_r}$ with $f_{\mathcal{P}}(r) \geqslant f_r$ and

(2) the child v_j is labeled by $b_j(\beta(x_j))^{i_j}_{f_j}$ for every $1 \leqslant j \leqslant l$, and

(3) $i_0 = \prod_{j=1}^{l} i_j$.

Note that in contrast to proof trees under set semantics, these proof trees are ordered trees where the children are ordered in respect to the occurrence of their labeling in the datalog rule. Therefore, the ordering of the atoms in the body of every datalog rule is relevant under bag semantics. The subscript f_r denotes which occurrence of the rule r is used and the subscript is zero if the atomic fact occurs in the database. Furthermore note, there exists potentially more than one proof tree of one fact $P(a_1, \ldots, a_k)^i$ from the datalog program $\mathcal{P}$ and the database $\mathcal{D}$.

Now, let $Q = (\mathcal{P}, P)$ be a k-ary datalog query. Furthermore, let the forest $\mathcal{F}_{Q,\mathcal{D}}(\overline{a})$ for $\overline{a} = (a_1, \ldots, a_k) \in \mathrm{dom}(\mathcal{D})^k$ be the following set

$$\mathcal{F}_{Q,\mathcal{D}}(\overline{a}) := \left\{ T \;\middle|\; T \text{ is a proof tree for the fact } P(\overline{a})^i \text{ for an } i \in \mathbb{N}_{\geqslant 1} \right\}$$

of all proof trees for the fact $P(a_1, \ldots, a_k)$ of any frequency (greater than zero) from $\mathcal{P}$ and $\mathcal{D}$.

Finally the result of a datalog query $Q = (\mathcal{P}, P)$ evaluated on $\mathcal{D}$ under bag semantics is defined by

$$Q(\mathcal{D}) = \left\{ (\overline{a})^{f(\overline{a})} \;\middle|\; \begin{array}{c} \text{for } \overline{a} = (a_1, \ldots, a_{ar(Q)}) \in \mathrm{dom}(\mathcal{D})^{ar(Q)} \text{ and } f(\overline{a}) = \sum\limits_{T \in \mathcal{F}_{Q,\mathcal{D}}(\overline{a})} j_T \\ \text{where } T \text{ is a proof tree of the fact } P(a_1, \ldots, a_{ar(Q)})^{j_T} \text{ from } \mathcal{P} \text{ and } \mathcal{D} \end{array} \right\}$$

Again, the result of Q in $\mathcal{D}$ defines a function $f_{Q(\mathcal{D})} : A^{ar(Q)} \to \mathbb{N}$ indicating the frequency of any tuple $\overline{a} = (a_1, \ldots, a_{ar(Q)})$ in $Q(\mathcal{D})$.

Example 7.3. *Consider the database $\mathcal{D}$ of Example 7.1. Furthermore, let $Q = (\mathcal{P}, P)$ the datalog query where the datalog program $\mathcal{P}$ consists of the following rules:*

$$\begin{aligned} A(x) &\leftarrow R(x) \\ A(x) &\leftarrow R(x), R(x) \\ P(x, y) &\leftarrow A(x), S(x, y). \end{aligned}$$

each of frequency one.

The forest $\mathcal{F} := \bigcup_{a_1, a_2 \in \mathrm{dom}(\mathcal{D})} \mathcal{F}_{Q,\mathcal{D}}(a_1, a_2)$ consists only of the two proof trees depicted in Figure 7.1. Therefore, the result of Q in $\mathcal{D}$ is $Q(\mathcal{D}) = \{(a, c)^6\}$.

As the datalog query Q is not recursive, an equivalent SQL *statement* S *is the following:*

$$\texttt{select } \texttt{R}_2\texttt{.A, S.B from } (\texttt{S}_1 \texttt{ union all } \texttt{S}_2) \texttt{ as } \texttt{R}_2\texttt{, S where } \texttt{R}_2\texttt{.A} = \texttt{S.A},$$

where $\texttt{S}_1$ *and* $\texttt{S}_2$ *are the* SQL *statements from Example 7.2.* ⌟

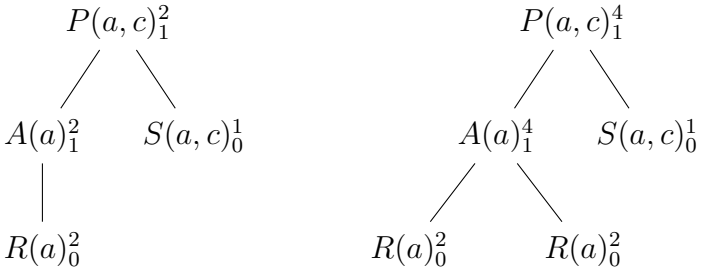

Figure 7.1: The forest $\mathcal{F} := \bigcup_{a_1,a_2 \in \mathrm{dom}(\mathcal{D})} \mathcal{F}_{Q,\mathcal{D}}(a_1, a_2)$

Example 7.4. *We consider the datalog query $Q = (\mathcal{P}, T)$ where the datalog program $\mathcal{P}$ over $\{E\}$ consists of the following rules:*

$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y), E(y,z)$$

each of frequency one. An equivalent SQL *statement* S *is the following:*

```
with recursive T(A, B) as (
    select E.A, E.B from E
union all
    select T.A, E.B from T, E where T.B = E.A
)
select T.A, T.B from T
```

The result of Q, and S *respectively, evaluated on a database $\mathcal{G}$ representing a directed graph yields under set semantics the transitive closure of the edge relation E. Under bag semantics the frequency of a tuple (a, b) corresponds to the count of different paths from a to b in the represented multigraph. Thus, for the result of Q in $\mathcal{G}$ with*

$$E^{\mathcal{G}} := \left\{ \begin{array}{c} (s,v_1)^1, (s,v_2)^1, (s,v_3)^2, (v_1,v_2)^1, \\ (v_2,t)^1, (v_3,v_4)^2, (v_4,t)^1 \end{array} \right\}$$

we have $(s,t)^6 \in Q(\mathcal{G})$. The graph represented in the database $\mathcal{G}$ is depicted in Figure 7.2.

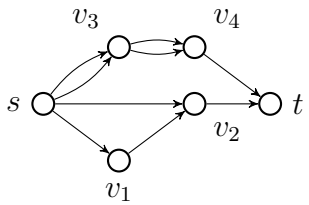

Figure 7.2: The graph represented in the database $\mathcal{G}$ of Example 7.4

Observe, adding an edge (v_2, v_1) makes the obtained directed graph $\mathcal{G}'$ cyclic. Now, there are infinitely many different paths from v_1 to v_2. Therefore, the result relation

defined by Q in $\mathcal{G}'$ under bag semantics is infinite and for the fact $E(v_1, v_2)$ of $\mathcal{P}$ and $\mathcal{G}'$ there exists an infinite forest of pairwise different proof trees. So in contrast to set semantics, we cannot be sure that the evaluation of the query Q, and the statement `S` *respectively, terminates.* ⌟

Now, let us consider an example of monadic datalog on trees under bag semantics. Since trees are given as structures without duplicates, we can expect that the complexity of our problems is located between the complexity of the problems under set semantics and the problems under bag semantics. In [CV93], [IR95], and[JKV06] the authors call such a setting *bag-set semantics*.

Example 7.5. *Let $\Sigma := \{Black, White\}$. We consider the monadic datalog query $Q = (\mathcal{P}, black_{labeled})$ over $\tau_{u,\Sigma}$ where $\mathcal{P}$ consists of the following rules (each of frequency one):*

$$black_{labeled}(x) \leftarrow \mathbf{child}(x, y), \mathbf{label}_{Black}(y)$$
$$black_{labeled}(x) \leftarrow \mathbf{child}(x, y), black_{labeled}(y)$$

Let T be the unordered Σ-labeled tree T presented in Example 2.1 and depicted in Figure 2.1, The result of Q evaluated on T under bag semantics is

$$Q(T) = \{(v_0)^5, (v_2)^1, (v_4)^1\}$$

as the query asks for all nodes having a descendant that is labeled by the symbol Black and the node is contained in the result for every such an occurrence.

An equivalent `SQL` *statement is the following:*

```
with recursive black_labeled(A) as (
    select child.A from child, label_Black where child.B = label_Black.A
union all
    select child.A from child, black_labeled where child.B = black_labeled.A
)
select black_labeled.A from black_labeled
```
⌟

7.3 Static Analysis of Datalog under Bag Semantics

In this section, we consider the emptiness problem and the query containment problem for (monadic) datalog under bag semantics on arbitrary databases.

A k-ary query Q is called *unsatisfiable under bag semantics* if for every database $\mathcal{D}$ and every tuple $\overline{a} = (a_1, \ldots, a_k) \in \mathrm{dom}(\mathcal{D})^k$ it holds that $f_{Q(\mathcal{D})}(\overline{a}) = 0$. We write $Q =_B \varnothing$ to indicate that Q is unsatisfiable under bag semantics and we write $Q \neq_B \varnothing$ to indicate that $Q =_B \varnothing$ does not hold.

Two k-ary queries Q_1 and Q_2 are called *equivalent under bag semantics* if for every database $\mathcal{D}$ and every tuple $\overline{a} = (a_1, \ldots, a_k) \in \mathrm{dom}(\mathcal{D})^k$ it holds that $f_{Q_1(\mathcal{D})}(\overline{a}) = f_{Q_2(\mathcal{D})}(\overline{a})$. We write $Q_1 \equiv_B Q_2$ to indicate that Q_1 and Q_2 are equivalent under bag semantics and we write $Q_1 \not\equiv_B Q_2$ to indicate that $Q_1 \equiv_B Q_2$ does not hold.

Let Q_2 be a k-ary query. A k-ary query Q_1 is called *contained in Q_2 under bag semantics* if for every database $\mathcal{D}$ and every tuple $\overline{a} = (a_1, \ldots, a_k) \in \mathrm{dom}(\mathcal{D})^k$ it holds

that $f_{Q_1(\mathcal{D})}(\overline{a}) \leqslant f_{Q_2(\mathcal{D})}(\overline{a})$. We write $Q_1 \subseteq_B Q_2$ to indicate that Q_1 is contained in Q_2 under bag semantics and we write $Q_1 \not\subseteq_B Q_2$ to indicate that $Q_1 \subseteq_B Q_2$ does not hold.

The corresponding decision problems are defined in the usual way.

7.3.1 The Emptiness Problem of Datalog under Bag Semantics

In this subsection, we will see that the complexity of the emptiness problem under bag semantics is the same as under set semantics.

Proposition 7.6. *Let $Q = (\mathcal{P}, P)$ be a k-ary datalog query of schema τ. Then it is equivalent to decide*

(a) $Q =_B \varnothing$, or

(b) $Q_{set} = \varnothing$ for $Q_{set} := (\mathcal{P}_{set}, P)$.

Proof. Assuming $Q \neq_B \varnothing$. Then, there exist a database $\mathcal{D}$, an $i \in \mathbb{N}_{\geqslant 1}$, and a tuple $\overline{a} = (a_1, \ldots, a_k) \in \mathrm{dom}(\mathcal{D})^k$ such that $f_{Q(\mathcal{D})}(\overline{a}) = i$. This implies that there exists a natural number $i' \in \mathbb{N}_{\geqslant 1}$ with $0 < i' \leqslant i$ and a proof tree T for the fact $P(\overline{a})^{i'}$ from $\mathcal{P}$ and $\mathcal{D}$. Let $\mathcal{D}_{set}$ be the τ-structure obtained from $\mathcal{D}$ where all duplicates in the relations are omitted. Now, it is easy to verify that T_{set} obtained from T by omitting the super- and subscripts from the labels is a proof tree for the fact $P(\overline{a})$ from $\mathcal{P}_{\text{set}}$ and $\mathcal{D}_{set}$ in the set semantics. That implies that $Q \neq \varnothing$.

The converse direction is easy to verify, as we assume that $Q_{set} \neq \varnothing$. Then there exists a database $\mathcal{D}$ (without duplicates) from schema τ such that $Q_{set}(\mathcal{D}) \neq \varnothing$. It is obvious, that $Q(\mathcal{D}) \neq_B \varnothing$ and therefore $Q \neq_B \varnothing$. □

By following the argumentation of Proposition 7.6, we immediately obtain the following corollary.

Corollary 7.7. *Let N be a set with $N \subseteq \{\mathbf{root}, \mathbf{leaf}\}$ and let M be a set with $M \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}, \mathbf{child}\}$.*

(a) The emptiness problem for mDatalog(τ_u^N) on finite unordered labeled trees under bag semantics is complete for Exptime.

(b) The emptiness problem for mDatalog(τ_o^N) on finite ordered labeled trees under bag semantics is complete for Exptime.

(c) The emptiness problem for mDatalog($\tau_u^{(N \cup \mathbf{desc})}$) on finite unordered unranked labeled trees under bag semantics is hard for Exptime *and solvable within 2-fold exponential time.*

(d) The emptiness problem for mDatalog($\tau_o^{(M \cup \{\mathbf{desc}, \mathbf{child}\})}$) on finite ordered unranked labeled trees under bag semantics is hard for Exptime *and solvable within 2-fold exponential time.*

(e) The emptiness problem for mDatalog($\tau_{GK}^{(\{\mathbf{desc}, \mathbf{child}\})}$) on finite ordered unranked labeled trees under bag semantics is complete for 2Exptime.

(f) The emptiness problem for mDatalog($\tau_u^{(N \cup \mathbf{desc})}$) on finite unordered ranked labeled trees under bag semantics is complete for 2EXPTIME.

(g) The emptiness problem for mDatalog($\tau_o^{(M \cup \{\mathbf{desc},\mathbf{child}\})}$) on finite ordered ranked labeled trees under bag semantics is complete for 2EXPTIME.

Proof. By Proposition 7.6, we can use the algorithms of the set semantics version of the problems and the hardness follows from the observation that solving the emptiness problems in bag semantics is at least as hard as solving the emptiness problems under set semantics. □

7.3.2 The Query Containment Problem of Datalog under Bag Semantics

Now, we investigate the query containment problem for datalog under bag semantics. Analogously to the argumentation of the proof for Proposition 7.6, we can see that $Q_1 \subseteq_B Q_2$ implies $Q_1 \subseteq Q_2$ for two datalog queries Q_1 and Q_2, but as Example 7.8 demonstrates, the converse implication does not hold.

Example 7.8. *We consider the monadic datalog queries $Q_1 = (\mathcal{P}_1, P)$ and $Q_2 = (\mathcal{P}_2, P)$ where*

$$\mathcal{P}_1 = \{P(x) \leftarrow R(x), S(x)\} \quad \text{and} \quad \mathcal{P}_2 = \{P(x) \leftarrow R(x)\}.$$

Clearly, we have $Q_1 \subseteq Q_2$ as the rule contained in $\mathcal{P}_1$ is a restriction of the rule contained in $\mathcal{P}_2$.

Now, let $\mathcal{D}$ be the database consisting of the following relations:

R
a

S
a
a

It holds that $Q_1(\mathcal{D}) = \{(a)^2\}$ and $Q_2(\mathcal{D}) = \{(a)^1\}$ and therefore, we have $Q_1 \not\subseteq_B Q_2$. ⌟

Now, it is clear that deciding $Q_1 \subseteq_B Q_2$ for (monadic) datalog on finite structures is harder than to decide $Q_1 \subseteq Q_2$. Moreover, section's main goal is to prove the following theorem stating that in general the containment problem for monadic datalog under bag semantics is undecidable.

Theorem 7.9. *For every $m \in \mathbb{N}_{\geqslant 1}$, let τ_m be the schema consisting of the unary relation symbols $X_1, \ldots, X_m$.*

There exists an $m \in \mathbb{N}_{\geqslant 1}$ such that

(a) the query containment problem for datalog queries over the schema τ_m under bag semantics, and

(b) the query containment problem for monadic datalog queries over the schema τ_m under bag semantics

is undecidable.

As the queries of Example 7.8 can be rewritten into conjunctive queries, it is nearby to consider at first the query containment problem for conjunctive queries. Unfortunately, it is not known whether the problem for conjunctive queries is decidable. So, let us mention two extensions of conjunctive queries for which the query containment problem under bag semantics is known to be undecidable.

The first extension is to allow inequality atoms in the body of the rule. Such an inequality atom forces any valuation to map the two variables of the inequality atom to different elements of the domain.

In 2006, Jayram et al. [JKV06] showed that the query containment problem for conjunctive queries with inequalities under bag semantics is undecidable.

The second extension under bag semantics is to allow a finite multiset of conjunctive queries using the same head atom. Similar to the definition of datalog programs in bag semantics, the frequency of a conjunctive query Q_{CQ} in a union Q of conjunctive queries is defined by a function $f_Q(r) : \text{CQ} \to \mathbb{N}$ where CQ is the set consisting of all possible conjunctive queries. We write $Q_{\text{CQ}} \in Q$ if $f_Q(Q_{\text{CQ}}) > 0$. Speaking about a union Q of conjunctive queries $Q_1, Q_2, \ldots, Q_n$ means a listing $Q_1, Q_2, \ldots, Q_n$ of every occurrence of a conjunctive query in Q, so two or more queries in the listing can be equal.

For an $n \in \mathbb{N}_{\geqslant 1}$, the result of a union Q of conjunctive queries $Q_1, Q_2, \ldots, Q_n$ evaluated in a database $\mathcal{D}$ of schema τ under bag semantics is defined as follows:[2]

$$Q(\mathcal{D}) = \left\{ (a_1, \ldots, a_{ar(Q)})^f \;\middle|\; \begin{array}{c} \text{for } a_1, \ldots, a_{ar(Q)} \in \text{dom}(\mathcal{D}), \text{ and} \\ f = \sum_{i=1}^{n} j \text{ , for } (a_1, \ldots, a_{ar(Q)})^j \in Q_i(\mathcal{D}) \end{array} \right\}$$

The frequency $f_{Q(\mathcal{D})}(\overline{a})$ of a tuple $\overline{a} \in \text{dom}(\mathcal{D})^{ar(Q)}$ in the result of a union Q of conjunctive queries evaluated in a database $\mathcal{D}$ is defined similar to $f_{Q'(\mathcal{D})}(\overline{a})$ for a datalog query Q'.

The `SQL` statement for bag union of two `SQL` statements $\texttt{S}_1$ and $\texttt{S}_2$ is given by:[3]

```
S1 UNION ALL S2
```

In 1993, Ioannidis and Ramakrishnan [IR95] investigated the query containment problem for unions of conjunctive queries under bag semantics. A special labeling system was used to obtain results for the query containment problem under several semantics. For the sake of completeness and to obtain a formulation of the result usable for a later reduction to (monadic) datalog, we reconstruct the proof of Ioannidis and Ramakrishnan and adopt it to our notation and context.

Starting point is the *m-variable Diophantine equation problem over the natural numbers* and Yuri Matiyasevich's Theorem.[4] The problem is defined as follows.

[2]It is equivalent to define the result of a union Q of conjunctive queries evaluated in a database by

$$Q(\mathcal{D}) = \left\{ (a_1, \ldots, a_{ar(Q)})^f \;\middle|\; \begin{array}{c} \text{for } a_1, \ldots, a_{ar(Q)} \in \text{dom}(\mathcal{D}), \text{ and} \\ f = \sum_{Q_{\text{CQ}} \in Q} j \cdot f_Q(Q_{CQ}) \text{ , for } (a_1, \ldots, a_{ar(Q)})^j \in Q_{\text{CQ}}(\mathcal{D}) \end{array} \right\}.$$

[3]Note, the statements $\texttt{S}_1$ and $\texttt{S}_2$ have to be compatible, meaning of the same arity and the corresponding attributes have to be type compatible.

[4] Actually, Юрий Владимирович Матиясевич. Some authors use the alternative transcription Jurij Matijasevič.

THE m-VARIABLE DIOPHANTINE EQUATION PROBLEM OVER THE NATURAL NUMBERS

Input: $s \in \mathbb{N}_{\geqslant 1}$,
$a_1, \ldots, a_s \in \mathbb{Z}$,
for every $i \in \{1, \ldots, s\}$: $\quad c_{i,1}, \ldots, c_{i,m} \in \mathbb{N}$

Question: Are there natural numbers $z_1, \ldots, z_m \in \mathbb{N}$ such that for the polynom

$$P(x_1, \ldots, x_m) := \sum_{i=1}^{s} a_i x_1^{c_{i,1}} \cdots x_m^{c_{i,m}}$$

it holds that $P(z_1, \ldots, z_m) = 0$?

In 1970, Matiyasevich proved the following theorem.[5]

Theorem 7.10 (Matiyasevich [Mat70])**.** *There exists an $m \in \mathbb{N}_{\geqslant 1}$ such that the m-variable Diophantine equation problem over the natural numbers is undecidable.* ⌟

It is worth to say that the m-variable Diophantine equation problem over the natural numbers is a variant of Hilbert's tenth problem that asks for a general algorithm deciding the solvability of Diophantine equations. Therefore, Matiyasevich's Theorem implies a negative solution of Hilbert's tenth problem.

Now, we use Matiyasevich's Theorem to show that the *m-variable Diophantine $\leqslant$ problem over $\mathbb{N}$* is undecidable. This problem is defined as follows.

THE m-VARIABLE DIOPHANTINE $\leqslant$ PROBLEM OVER $\mathbb{N}$

Input: $s_1, s_2 \in \mathbb{N}_{\geqslant 1}$,
$a_1, \ldots, a_{s_1+s_2} \in \mathbb{N}_{\geqslant 1}$,
for every $i \in \{1, \ldots, s_1 + s_2\}$:
$c_{i,1}, \ldots, c_{i,m} \in \mathbb{N}$
with $c > 0$ for at least one $c \in \{c_{i,1}, \ldots, c_{i,m}\}$

Question: Holds that

$$\sum_{i=1}^{s_1} a_i x_1^{c_{i,1}} \cdots x_m^{c_{i,m}} \leqslant \sum_{i=s_1+1}^{s_2} a_i x_1^{c_{i,1}} \cdots x_m^{c_{i,m}}$$

for all natural numbers $x_1, \ldots, x_m \in \mathbb{N}$?

Proposition 7.11 (implicit in [IR95])**.** *There exists an $m \in \mathbb{N}_{\geqslant 1}$ such that the m-variable Diophantine $\leqslant$ problem over $\mathbb{N}$ is undecidable.*

[5] As the result is a conjunction of Matiyasevich's 'Every computably enumerable set is Diophantine.' and a previous work of Martin Davis, Hilary Putnam, and Julia Robinson [DPR61], the theorem is also known as the DPRM-theorem (or the MRDP-theorem). It is an open problem to find the least number of variables needed to obtain an unsolvable result. The currently best-known bound is nine by Matiyasevich, presented in a publication of Jones [Jon82]. For further information, we refer to the textbook by Smoryński [Smo91].

Proof. We assume the m-variable Diophantine $\leqslant$ problem over $\mathbb{N}$ is decidable for every $m \in \mathbb{N}_{\geqslant 1}$ and show that this assumption contradicts Theorem 7.10.

Let $s \in \mathbb{N}_{\geqslant 1}$, $a_1, \ldots, a_s \in \mathbb{Z}$, and $c_{i,1}, \ldots, c_{i,n} \in \mathbb{N}$ for every $i \in \{1, \ldots, s\}$ be the input for the n-variable Diophantine equation problem over the natural numbers. This input defines a polynom

$$P(x_1, \ldots, x_n) := \sum_{i=1}^{s} a_i x_1^{c_{i,1}} \cdots x_n^{c_{i,n}}$$

and it is to decide whether there are natural numbers $z_1, \ldots, z_n \in \mathbb{N}$ such that

$$P(z_1, \ldots, z_n) = 0.$$

Obviously, the statement

$$\text{there exist natural numbers } z_1, \ldots, z_n \in \mathbb{N} \text{ such that } P(z_1, \ldots, z_n) = 0$$

is equivalent to the statement

$$\text{not for all } z_1, \ldots, z_n \in \mathbb{N} \text{ it holds that } P(z_1, \ldots, z_n) \neq 0.$$

As all parameters are integrals, it is equivalent to

$$\text{not for all } z_1, \ldots, z_n \in \mathbb{N} \text{ it holds that } P(z_1, \ldots .z_n) \leqslant -1 \text{ or } P(z_1, \ldots, z_n) \geqslant 1,$$

that is equivalent to the statement

$$\text{not for all } z_1, \ldots, z_n \in \mathbb{N} \text{ it holds that } P(z_1, \ldots, z_n)^2 \geqslant 1$$

or even

$$\text{not for all } z_1, \ldots, z_n \in \mathbb{N} \text{ it holds that } 1 - P(z_1, \ldots, z_n)^2 \leqslant 0.$$

Now, let x_{n+1} be a new variable such that $x_{n+1} \notin \{x_1, \ldots, x_n\}$ and consider the polynom $P'(x_1, \ldots, x_n, x_{n+1}) := x_{n+1}(1 - P(x_1, \ldots, x_n)^2)$. Then the aforementioned statements are equivalent to the statement

$$\text{not for all } z_1, \ldots, z_n, z_{n+1} \in \mathbb{N} \text{ it holds that } P'(z_1, \ldots, z_n, z_{n+1}) \leqslant 0.$$

Note, the polynom $P'(z_1, \ldots, z_n, z_{n+1})$ has no constant term. Therefore, there are $s_1, s_2 \in \mathbb{N}$, $a'_1, \ldots, a'_{s_1+s_2} \in \mathbb{Z}$, and $c'_{i,1}, \ldots, c'_{i,n+1} \in \mathbb{N}$ for every $i \in \{1, \ldots, s_1 + s_2\}$ such that

$$P'(x_1, \ldots, x_n, x_{n+1}) = \sum_{i=1}^{s_1+s_2} a'_i x_1^{c'_{i,1}} \cdots x_{n+1}^{c'_{i,n+1}}$$

where $a'_i > 0$ for all $i \in \{1, \ldots, s_1\}$ and $a'_i < 0$ for all $i \in \{s_1 + 1, \ldots, s_2\}$. Now, the statement

$$\text{not for all } z_1, \ldots, z_n, z_{n+1} \in \mathbb{N} \text{ it holds that } \sum_{i=1}^{s_1+s_2} a'_i z_1^{c'_{i,1}} \cdots z_{n+1}^{c'_{i,n+1}} \leqslant 0$$

is equivalent to

$$\text{not for all } z_1, \ldots, z_n, z_{n+1} \in \mathbb{N} \text{ it holds that } \sum_{i=1}^{s_1} a'_i z_1^{c'_{i,1}} \cdots z_{n+1}^{c'_{i,n+1}} - \sum_{i=s_1+1}^{s_2} |a'_i| z_1^{c'_{i,1}} \cdots z_{n+1}^{c'_{i,n+1}} \leqslant 0$$

that is equivalent to the statement

$$\text{not for all } z_1, \ldots, z_n, z_{n+1} \in \mathbb{N} \text{ it holds that } \sum_{i=1}^{s_1} a'_i z_1^{c'_{i,1}} \cdots z_{n+1}^{c'_{i,n+1}} \leqslant \sum_{i=s_1+1}^{s_2} |a'_i| z_1^{c'_{i,1}} \cdots z_{n+1}^{c'_{i,n+1}}.$$

Note, as P' has no constant term, there exists for every $i \in \{1, \ldots, s_1 + s_2\}$ a $c \in \{c'_{i,1}, \ldots, c'_{i,n+1}\}$ such that $c > 0$.

Now, an algorithm $\mathfrak{A}$ that has to decide the n-variable Diophantine equation problem over the natural numbers can proceed as follows. By assumption there exists for $m = n + 1$ an algorithm $\mathfrak{A}_{\leqslant}$ to decide the m-variable Diophantine $\leqslant$ problem over $\mathbb{N}$. $\mathfrak{A}$ computes on input $s \in \mathbb{N}_{\geqslant 1}$, $a_1, \ldots, a_s \in \mathbb{Z}$, and $c_{i,1}, \ldots, c_{i,n} \in \mathbb{N}$ for every $i \in \{1, \ldots, s\}$ the values of $s_1, s_2 \in \mathbb{N}$, $a'_1, \ldots, a'_{s_1+s_2} \in \mathbb{Z}$, and $c'_{i,1}, \ldots, c'_{i,n+1} \in \mathbb{N}$ for every $i \in \{1, \ldots, s_1 + s_2\}$ of the polynom P'.[6] Then $\mathfrak{A}$ returns **yes** if and only if $\mathfrak{A}_{\leqslant}$ on input $s_1, s_2, a'_1, \ldots, a'_{s_1}, |a'_{s_1+1}|, \ldots, |a'_{s_1+s_2}| \in \mathbb{N}_{\geqslant 1}$, and $c'_{i,1}, \ldots, c'_{i,n+1} \in \mathbb{N}$ for every $i \in \{1, \ldots, s_1 + s_2\}$ returns **no**, and $\mathfrak{A}$ returns **no**, otherwise. As seen above, this decides the n-variable Diophantine equation problem over the natural numbers for every $n \in \mathbb{N}$. This contradicts Theorem 7.10. Therefore the assumption must be false; that finishes the proof of Proposition 7.11. □

Now, we are ready to adapt the result of Ioannidis and Ramakrishnan.

Theorem 7.12 (implicit in [IR95])**.** *For every $m \in \mathbb{N}_{\geqslant 1}$ let τ_m be the schema consisting of the unary relation symbols $X_1, \ldots, X_m$.*

There exists an $m \in \mathbb{N}_{\geqslant 1}$ such that the query containment problem for union of unary conjunctive queries over the schema τ_m under bag semantics is undecidable.

Proof. We assume the problem is decidable for every $m \in \mathbb{N}_{\geqslant 1}$, hence there exists an algorithm deciding the problem. On the input of the m-variable Diophantine $\leqslant$ problem over $\mathbb{N}$ defining two polynomials $P_1(x_1, \ldots, x_m)$ and $P_2(x_1, \ldots, x_m)$, we construct two unions Q_1 and Q_2 of unary conjunctive queries of schema τ_m such that

$$Q_1 \subseteq_B Q_2 \qquad \Longleftrightarrow \qquad \begin{array}{c}\text{for all natural numbers } z_1, \ldots, z_m \text{ it holds that} \\ P_1(z_1, \ldots, z_m) \leqslant P_2(z_1, \ldots, z_m).\end{array}$$

Let $s_1, s_2, a_1, \ldots, a_{s_1+s_2}, c_{1,1} \ldots c_{s_1+s_2,m}$ be the input of the given problem instance of the m-variable Diophantine $\leqslant$ problem over $\mathbb{N}$. Then $P_1(x_1, \ldots, x_m)$ is defined as

$$P_1(x_1, \ldots, x_m) := \sum_{i=1}^{s_1} a_i x_1^{c_{i,1}} \cdots x_m^{c_{i,m}},$$

[6] These are simple multiplication and addition operations and a subsequent sorting.

where for every $i \in \{1, \ldots, s_1\}$ there is at least one $c > 0$ in $\{c_{i,1}, \ldots, c_{i,m}\}$.

Now, we construct the union Q_1 of unary conjunctive queries from P_1 as follows. Note, for every $l \in \{1, \ldots, m\}$ and every variable x_l there exists a unary relation X_l. For each term $a_i x_1^{c_{i,1}} \cdots x_m^{c_{i,m}}$, we define the conjunctive query $Q_{1,i}$ by

$$Ans(x) \leftarrow \underbrace{X_1(x) \wedge X_1(x) \wedge \ldots \wedge X_1(x)}_{c_{i,1} \text{ times}} \wedge \ldots \wedge \underbrace{X_m(x) \wedge X_m(x) \wedge \ldots \wedge X_m(x)}_{c_{i,m} \text{ times}}$$

that will occur with frequency a_i in the union Q_1. Recall, the polynom has no constant term, saying we have $c > 0$ for at least one $c \in \{c_{i,1}, \ldots, c_{i,m}\}$. This implies every query $Q_{1,i}$ is well defined and no conjunctive query has an empty body. Furthermore, if one exponent is zero, its factor can be dropped as for every natural number n it holds that $n^0 = 1$. Finally, the union Q_1 consist of $n_1 = \sum_{i=1}^{s_1} a_i$ conjunctive queries.

The union Q_2 of conjunctive queries is constructed from P_2 in the same way.

For the correctness of the construction, we consider an arbitrary database $\mathcal{D}$. Let $d \in \text{adom}(\mathcal{D})$ an arbitrary entry in the database. Then by construction of the union Q_1 of conjunctive queries and the defined semantics, we have

$$\begin{aligned}
f_{Q_1(\mathcal{D})}(d) &= \sum_{Q_{\text{CQ}} \in Q_1} f_{Q_{\text{CQ}}(\mathcal{D})}(d) \cdot f_{Q_1}(Q_{CQ}) \\
&= \sum_{i=1}^{s_1} f_{Q_{1,i}(\mathcal{D})}(d) \cdot f_{Q_1}(Q_{1,i}) \\
&= \sum_{i=1}^{s_1} \underbrace{f_{\mathcal{D}}(X_1(d)) \cdot \ldots \cdot f_{\mathcal{D}}(X_1(d))}_{c_{i,1} \text{ times}} \cdot \ldots \cdot \underbrace{f_{\mathcal{D}}(X_m(d)) \cdot \ldots \cdot f_{\mathcal{D}}(X_m(d))}_{c_{i,m} \text{ times}} \cdot a_i \\
&= \sum_{i=1}^{s_1} a_i \cdot f_{\mathcal{D}}(X_1(d))^{c_{i,1}} \cdots f_{\mathcal{D}}(X_m(d))^{c_{i,m}} \\
&= P_1(f_{\mathcal{D}}(X_1(d)), \ldots, f_{\mathcal{D}}(X_m(d))).
\end{aligned}$$

Analogously, we obtain $f_{Q_2(\mathcal{D})}(d) = P_2(f_{\mathcal{D}}(X_1(d)), \ldots, f_{\mathcal{D}}(X_m(d)))$ for Q_2.

We assume that for all natural numbers $z_1, \ldots, z_m$ it holds that

$$P_1(z_1, \ldots, z_m) \leqslant P_2(z_1, \ldots, z_m).$$

Then for every database $\mathcal{D}$ and for every element $d \in \text{adom}(\mathcal{D})$ it holds that $f_{Q_1(\mathcal{D})}(d) \leqslant f_{Q_2(\mathcal{D})}(d)$. This implies $Q_1(\mathcal{D}) \subseteq_B Q_2(\mathcal{D})$ and therefore $Q_1 \subseteq_B Q_2$.

If otherwise, we assume that not for all natural numbers $z_1, \ldots, z_m$ it holds that

$$P_1(z_1, \ldots, z_m) \leqslant P_2(z_1, \ldots, z_m)$$

then there exist natural numbers $z_1, \ldots, z_m \in \mathbb{N}$ such that

$$P_1(z_1, \ldots, z_m) > P_2(z_1, \ldots, z_m).$$

Consider the database $\mathcal{D}$ with $\text{adom}(\mathcal{D}) = \{d\}$ and $d^{z_i} \in X_i^{\mathcal{D}}$ for every $i \in \{1, \ldots, m\}$. Then $f_{Q_1(\mathcal{D})}(d) > f_{Q_2(\mathcal{D})}(d)$. This implies $Q_1(\mathcal{D}) \not\subseteq_B Q_2(\mathcal{D})$ and therefore, $Q_1 \not\subseteq_B Q_2$. □

Now, we are ready to prove Theorem 7.9.

Theorem 7.9 ***(restated)*** *For every $m \in \mathbb{N}_{\geqslant 1}$ let τ_m be the schema consisting of the unary relation symbols $X_1, \ldots, X_m$.*

There exists an $m \in \mathbb{N}_{\geqslant 1}$ such that

(a) the query containment problem for datalog queries over the schema τ_m under bag semantics, and

(b) the query containment problem for monadic datalog queries over the schema τ_m under bag semantics

is undecidable.

Proof. Note *(b)* implies *(a)*, so it suffices to show that claim *(b)* is true.

We assume that for every $m \in \mathbb{N}$ the query containment problem for monadic datalog over the schema τ_m under bag semantics is decidable. This implies an algorithm $\mathfrak{A}$ that decides for two given monadic datalog queries Q_1 and Q_2 from schema τ_m whether $Q_1 \subseteq_B Q_2$.

Let $m \in \mathbb{N}$ and Q_1^{CQ} and Q_2^{CQ} be two unions of conjunctive queries of arity one over the schema τ_m. By standard translation, we construct two monadic datalog queries Q_1 and Q_2 over τ_m such that for every database $\mathcal{D}$ of schema τ it holds that $Q_1^{CQ}(\mathcal{D}) = Q_1(\mathcal{D})$ and $Q_2^{CQ}(D) = Q_2(\mathcal{D})$.

Now, we decide $Q_1 \subseteq_B Q_2$ by using algorithm $\mathfrak{A}$. As $Q_1^{CQ} \equiv_B Q_1$ and $Q_2^{CQ} \equiv_B Q_2$, we decide $Q_1^{CQ} \subseteq_B Q_2^{CQ}$ that contradicts Theorem 7.12. So the assumption must be false. □

Finally, we consider the query containment problem under bag semantics on trees and we show that in general the problem is undecidable for monadic datalog.

Proposition 7.13. *For every $m \in \mathbb{N}_{\geqslant 1}$ let Σ_m be the alphabet $\Sigma_m := \{X_1, \ldots, X_m\}$.*

There exists an $m \in \mathbb{N}_{\geqslant 1}$ such that

(a) the query containment problem for mDatalog(τ_u) under bag-semantics on finite unordered Σ_m-labeled trees is undecidable.

(b) the query containment problem for mDatalog(τ_o) under bag-semantics on finite ordered Σ_m-labeled trees is undecidable.

Proof. We assume that the claim (b) is false; this implies that for every $m \in \mathbb{N}$ the query containment problem for mDatalog(τ_o) under bag semantics on finite ordered Σ_m-labeled trees is decidable. Therefore, there exists an algorithm $\mathfrak{A}$ that decides for a given alphabet Σ_m and two given monadic datalog queries Q_1 and Q_2 from schema τ_o whether for every ordered labeled tree T it holds that $Q_1(T) \subseteq_B Q_2(T)$.

For any $m \in \mathbb{N}$ let τ_m be the schema consisting of m unary relation symbols $X_1, \ldots, X_m$. Let $Q_1 = (\mathcal{P}_1, P)$ and $Q_2 = (\mathcal{P}_2, P)$ be two monadic datalog queries over τ_m. Now, we

set the alphabet $\Sigma_m := \{X_1, \ldots, X_m\}$ and define the monadic datalog program $\mathcal{P}_T$ using schema τ_{o,Σ_m} consisting of a unique occurrence of the following rules for every $1 \leqslant j \leqslant m$.

$$\begin{aligned} P_{X_j}(x) &\leftarrow \mathbf{label}_{X_j}(x) \\ P_{X_j}(x) &\leftarrow \mathbf{ns}(x, y), P_{X_j}(y) \\ X_j(x) &\leftarrow \mathbf{fc}(x, y), P_{X_j}(y) \end{aligned}$$

Furthermore, let Q'_1 and Q'_2 be the monadic datalog queries $Q'_1 := (\mathcal{P}_1 \cup \mathcal{P}_T, P)$ and $Q'_2 := (\mathcal{P}_2 \cup \mathcal{P}_T, P)$.

The result of evaluating the program $\mathcal{P}_T$ on an ordered Σ_m-labeled tree is a database instance $\mathcal{D}_T$ over the unary *idb*-relations $X_1, \ldots, X_m$ such that for every $i, j \in \mathbb{N}$, $1 \leqslant j \leqslant m$ and every non-leaf node v in T it holds:

$$(v)^i \in X_j \text{ if and only if } v \text{ has exactly } i \text{ children labeled by } X_j.$$

Therefore, it holds that if there is an ordered Σ_m-labeled tree T such that $Q'_1(T) \not\subseteq_B Q'_2(T)$, then there exists a database $\mathcal{D}$ (defined through T) such that $Q_1(\mathcal{D}) \not\subseteq_B Q_2(\mathcal{D})$.

If otherwise there exists a database $\mathcal{D}$ over τ_m such that $Q_1(\mathcal{D}) \not\subseteq_B Q_2(\mathcal{D})$, then there exists a tree $T_\mathcal{D}$ such that $Q'_1(T_\mathcal{D}) \not\subseteq_B Q'_2(T_\mathcal{D})$. This tree $T_\mathcal{D}$ can be obtained from the database $\mathcal{D}$, for instance, in the following way. We order the active domain $\mathrm{adom}(\mathcal{D})$ in any way. Let N be a set disjoint from $\mathrm{adom}(\mathcal{D})$. We start with a tree consisting only of the root that is an element of N and labeled by any symbol of Σ_m. Now, to construct $T_\mathcal{D}$ we proceed recursively over the elements of $\mathrm{adom}(\mathcal{D})$ by following the order. Let a be the actual element of $\mathrm{adom}(\mathcal{D})$. Then we choose any leaf of $T_\mathcal{D}$, replace it by a, and label the new inserted node a with the label of the omitted one. Now, we add for every occurrence of a in any relation X_i for $1 \leqslant i \leqslant m$ an element of N as child of $a \in T_\mathcal{D}$ labeled by X_i. Then, we continue with the next element of $\mathrm{adom}(\mathcal{D})$. Note that after every recursive step all leaf nodes are elements of N and all non-leaf nodes are elements of $\mathrm{adom}(\mathcal{D})$. Furthermore, the children of every non leaf node represent its occurrence and its frequency in every relation $X_1, \ldots, X_m$.

Finally, it holds that $Q_1 \subseteq_B Q_2$ if and only if for all ordered Σ_m-labeled trees T it holds that $Q'_1(T) \subseteq_B Q'_2(T)$ where $Q'_1 := (\mathcal{P}_1 \cup \mathcal{P}_T, P)$ and $Q'_2 := (\mathcal{P}_2 \cup \mathcal{P}_T, P)$. This implies that deciding the containment of Q'_1 in Q'_2 over trees under bag semantics by using algorithm $\mathfrak{A}$ decides $Q_1 \subseteq_B Q_2$. This contradicts Theorem 7.9 stating that $Q_1 \subseteq_B Q_2$ is undecidable.

The proof of claim (a) is similar to the proof of (b) but uses the monadic datalog program

$$\mathcal{P}_T := \left\{ X_j(x) \leftarrow \mathbf{child}(x, y), \mathbf{label}_{X_j}(y) \right\}_{1 \leqslant j \leqslant m}. \qquad \square$$

In the previous sections, we have investigated the emptiness problem and the query containment problem of (monadic) datalog under bag semantics. The equivalence problem under bag semantics is left open.

Surprisingly, the equivalence problem for conjunctive queries under bag semantics seems to be easier than under set semantics: Under set semantics equivalence for conjunctive queries is known to be NP-complete [CM77]. But in 1993 Chaudhuri and Vardi [CV93] stated that two conjunctive queries Q_1 and Q_2 are equivalent under bag semantics if and only if they are isomorphic.[7] This means that two queries are equivalent under bag

[7] The graph-isomorphism problem is known to be in NP but it is not known to be hard for NP.

semantics if and only if they are identical up to renaming and reordering. This would imply minimization by replacing queries by equivalent conjunctive queries with a smaller number of body atoms to find a minimally equivalent conjunctive query would fail.

However, to the best of the author's knowledge it is not known whether the equivalence problem under bag semantics for monadic datalog on finite labeled trees is decidable. So, it remains as an open question.

Chapter 8

Conclusion

In this thesis, we considered the emptiness problem, the equivalence problem, as well as the query containment problem of monadic datalog for various representations of finite labeled trees. These are fundamental problems of static analysis and thus, important tasks during query optimization.

We distinguished ordered and unordered trees, each labeled with symbols from an either ranked or unranked alphabet. As basic schema for unordered trees we employed a representation that uses the *child*-axis and for ordered trees we basically utilized the axes *firstchild* and *nextsibling*. Additionally, we considered schema extensions by the predicates denoting the root, leaves, and descendants, as well as last sibling and child on ordered trees. As seen in the first part of Chapter 3, the query languages using the various schemas over the miscellaneous representations differ strongly according to their expressive power. In the same chapter and by using results from logic, we established the decidability of the considered problems for all representations. Regarding the expressiveness, widely diversified complexity results could be expected. Surprisingly, they are relatively homogeneous.

In Chapter 4, we showed a lower bound for the aforementioned problems using the basic schemas. To obtain EXPTIME-hardness, we presented a reduction from the *two person corridor tiling problem* to the emptiness problem for Boolean monadic datalog on unordered unranked labeled trees within polynomial time. We extended the proof for ranked labeled trees and used the results to establish the lower bound even on ordered trees, as well as for the equivalence and the query containment problem on these structures.

For schemas and their extensions not including the descendant predicate, a matching upper bound is presented in Chapter 5. This chapter makes considerable use of automata theory. In a first step, we reduced the problems to the corresponding problems over *binary trees*. Then, we translated the given queries into *two-way alternating tree automata* whose principle of operation is very similar to the evaluation of monadic datalog queries on trees without descendant predicate. To decide the problems, we finally computed equivalent *nondeterministic bottom-up tree automata*.

While in Chapter 3 we have seen that adding the descendant predicate does not increase the expressive power, we realized in Chapter 6 that it enhances the complexity of the considered problems enormously. In the first part of Chapter 6, we presented an algorithm that solves our problems within 2-fold exponential time. This was achieved

	τ_u^N		τ_o^M		
Emptiness	EXPTIME-compl.	5.19(b)/4.4 5.19(b)/4.8	EXPTIME-compl.	5.19(b)/4.10 5.19(b)/4.10	*unranked* *ranked*
Equivalence	EXPTIME-compl.	5.19(a)/4.9(a) 5.19(a)/4.9(a)	EXPTIME-compl.	5.19(a)/4.14(a) 5.19(a)/4.14(a)	*unranked* *ranked*
Containment	EXPTIME-compl.	5.2/4.9(b) 5.17(b)/4.9(b)	EXPTIME-compl.	5.1/4.14(b) 5.17(a)/4.14(b)	*unranked* *ranked*

Table 8.1: Complexity results for monadic datalog on finite labeled trees not using **desc** obtained in this thesis. Here, N is a set with $N \subseteq \{\mathbf{root}, \mathbf{leaf}\}$ and M is a set with $M \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}, \mathbf{child}\}$. The pointers to the left refer to the membership results as the right ones point to the hardness result.

	$\tau_u^{N_d}$		$\tau_o^{M_{dc}}$		$\tau_{GK}^{M_{dc}}$		
Emptiness	EXPTIME-hard 2EXPTIME-compl.	4.4 6.16	EXPTIME-hard 2EXPTIME-compl.	4.10 6.20	2EXPTIME-compl.	6.23 6.20	*unranked* *ranked*
Equivalence	2EXPTIME-compl.	6.22 6.18(a)	2EXPTIME-compl.	6.24(c) 6.20		6.24(c) 6.20	*unranked* *ranked*
Containment	2EXPTIME-compl.	6.21 6.18(b)	2EXPTIME-compl.	6.24(e) 6.20		6.14 6.20	*unranked* *ranked*

Table 8.2: Complexity results for monadic datalog on finite labeled trees using the descendant-axis obtained in this thesis. 2EXPTIME-membership for all problems are given by Theorem 6.1 and Corollary 6.6. So, the pointers refer to the corresponding hardness results. N_d is a set with $\{\mathbf{desc}\} \subseteq N_d \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{desc}\}$ and M_{dc} is a set with $\{\mathbf{child}, \mathbf{desc}\} \subseteq M_{dc} \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}, \mathbf{child}, \mathbf{desc}\}$.

by rewriting the monadic datalog programs into equivalent programs in monadic datalog without descendant predicates. However, this led to an exponential blow up in the case of cyclic rules.

For ranked trees a matching lower bound on all ordered and unordered trees by a reduction from the *word problem of exponentially bounded alternating Turing machines* is presented. The results concerning the equivalence problem and the query containment problem are transferred to unranked trees. The matching lower bound for the emptiness problem on ranked trees is only established for ordered trees that use the whole schema. As the presented reduction from the word problem of exponentially bounded alternating Turing machines is technical, an alternative short and intuitive proof for the hardness of the query containment problem of ordered unranked labeled trees using the maximal schema is given. However, this proof cannot be extended to the other problems and structures.

In Chapter 7, we shifted the focus from the set semantics to the bag semantics. Bag semantics, reflects that real-world databases allow multiple occurrences of a tuple in a relation and in a query's result. We proposed a semantics for monadic datalog that

extends the known bag semantics for conjunctive queries. This semantics bases on the proof trees used in the set theoretic setting and agrees with the recursive evaluation of the SQL standard. We have seen that the complexity of the emptiness problem for monadic datalog under bag semantics on finite labeled trees complies with the complexity of the same problem under set semantics. For the query containment problem of monadic datalog under bag semantics on finite labeled trees we proved undecidability in general. This was done by a reduction from a variant of Hilbert's tenth problem. The complexity of the equivalence problem for monadic datalog under bag semantics on finite labeled trees remains as an open question.

Table 8.1 and Table 8.2 give an overview of the results under set semantics obtained in the thesis and point to the respective theorems. In summary, we achieved a virtually complete analysis of the problems entailed by static analysis of monadic datalog on finite labeled trees.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[Abi01] Serge Abiteboul. Semistructured Data: from Practice to Theory. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 379–386. IEEE Computer Society, 2001.

[ABMW13] Serge Abiteboul, Pierre Bourhis, Anca Muscholl, and Zhilin Wu. Recursive queries on trees and data trees. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 93–104, 2013.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[BBS12] Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic Datalog Containment. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 79–91. Springer, 2012.

[BDG90] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity II*, volume 22 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1990.

[BMS08a] Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing Conjunctive Queries over Trees using Schema Information. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2008.

[BMS08b] Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing Conjunctive Queries over Trees using Schema Information (full version). Available at `http://www8.cs.umu.se/~henrikb/papers/mfcs08full.pdf`, 2008. Full Version of [BMS08a], Accessed: 2016-03-05.

[CDG+08] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Christof Löding, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications. Available at `http://www.grappa.univ-lille3.fr/tata`, 2008. Release November, 18th 2008.

[CGKV88] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable Optimization Problems for Database Logic Programs (Preliminary Report). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 477–490. ACM, 1988.

[Chl86] Bogdan S. Chlebus. Domino-Tiling Games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM.

[CM01] James Clark and Makoto Murata. RELAX NG Specification. Available at `http://relaxng.org/spec-20011203.html`, (2001). *The Organization for the Advancement of Structured Information Standards* [OASIS].

[Cou90] Bruno Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 193–242. 1990.

[CS76] Ashok K. Chandra and Larry J. Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 98–108. IEEE Computer Society, 1976.

[CV93] Surajit Chaudhuri and Moshe Y. Vardi. Optimization of *Real* Conjunctive Queries. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 59–70. ACM Press, 1993.

[Don70] John Doner. Tree Acceptors and Some of Their Applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970.

[DPR61] Martin Davis, Hilary Putnam, and Julia Robinson. The decision problem for exponential diophantine equations. *Annals of Mathematics*, 74(3):425–436, 1961.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995.

[FFG01] Jörg Flum, Markus Frick, and Martin Grohe. Query Evaluation via Tree-Decompositions. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, volume 1973 of *Lecture Notes in Computer Science*, pages 22–38. Springer, 2001.

[FG02] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 215–224. IEEE Computer Society, 2002.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[FGS14] André Frochaux, Martin Grohe, and Nicole Schweikardt. Monadic Datalog Containment on Trees. In Georg Gottlob and Jorge Pérez, editors, *Proceedings of the 8th Alberto Mendelzon Workshop on Foundations of Data Management, Cartagena de Indias, Colombia, June 4-6, 2014*, volume 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

[FS13] André Frochaux and Nicole Schweikardt. A note on monadic datalog on unranked trees. *Technical Report, available at CoRR*, abs/1310.1316, 2013.

[FS16] André Frochaux and Nicole Schweikardt. Monadic Datalog Containment on Trees Using the Descendant-Axis. In Reinhard Pichler and Altigran Soares da Silva, editors, *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016*, volume 1644 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[GK04] Georg Gottlob and Christoph Koch. Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. *Journal of the ACM*, 51(1):74–113, 2004.

[GKS06] Georg Gottlob, Christoph Koch, and Klaus U. Schulz. Conjunctive Queries over Trees. *Journal of the ACM*, 53(2):238–272, 2006.

[HS65] Juris Hartmanis and Richard E. Stearns. ON THE COMPUTATIONAL COMPLEXITY OF ALGORITHMS. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[Imm86] Neil Immerman. Relational Queries Computable in Polynomial Time. *Information and Control*, 68(1-3):86–104, 1986.

[IR95] Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of Conjunctive Queries: Beyond Relations as Sets. *ACM Transactions on Database Systems (TODS)*, 20(3):288–324, 1995.

[JKV06] T. S. Jayram, Phokion G. Kolaitis, and Erik Vee. The Containment Problem for *REAL* Conjunctive Queries with Inequalities. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 80–89. ACM, 2006.

[Jon82] James P. Jones. Universal diophantine equation. *The Journal of Symbolic Logic*, 47(3):549–571, 1982.

[Koz76] Dexter Kozen. ON PARALLELISM IN TURING MACHINES. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 89–97. IEEE Computer Society, 1976.

[Koz06] Dexter Kozen. *Theory of Computation.* Texts in Computer Science. Springer, 2006.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[Lib06] Leonid Libkin. LOGICS FOR UNRANKED TREES: AN OVERVIEW. *Logical Methods in Computer Science*, 2(3), 2006.

[Löd12] Christof Löding. Basics on Tree Automata. In Deepak D'Souza and Priti Shankar, editors, *Modern Applications of Automata Theory.* World Scientific, 2012.

[Mat70] Yuri Matiyasevich. Enumerable sets are diophantine. *Doklady Akademii Nauk SSSR*, 191(2):279–282, 1970.

[MFS10] Sebastian Maneth, Sylvia Friese, and Helmut Seidl. Type-Checking Tree Walking Transducers. In Deepak D'Souza and Priti Shankar, editors, *Modern applications of automata theory*, volume 2 of *IISc Research Monographs.* World Scientific, 2010.

[Nev02] Frank Neven. Automata, Logic, and XML. In Julian C. Bradfield, editor, *Computer Science Logic, 16th International Workshop, CSL 2002, 11th Annual Conference of the EACSL, Edinburgh, Scotland, UK, September 22-25, 2002, Proceedings*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.

[NS02] Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.

[Pap94] Christos H. Papadimitriou. *COMPUTATIONAL COMPLEXITY.* Addison-Wesley, 1994.

[Sch07] Thomas Schwentick. Automata for XML - A survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.

[See91] Detlef Seese. The structure of the models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53(2):169–195, 1991.

[Shm87] Oded Shmueli. DECIDABILITY AND EXPRESSIVENESS ASPECTS OF LOGIC QUERIES. In Moshe Y. Vardi, editor, *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 23-25, 1987, San Diego, California, USA*, pages 237–249. ACM, 1987.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation.* Thomson Course Technology, second, international edition, 1997.

[Slu85] Giora Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305 – 318, 1985.

[Smo91] Craig A. Smoryński. *Logical Number Theory I.* Universitext. Springer-Verlag, Berlin, New York, 1991.

[SSS09] Nicole Schweikardt, Thomas Schwentick, and Luc Segoufin. Database theory: Query languages. In Mikhail J. Atallah and Marina Blanton, editors, *Algorithms and Theory of Computation Handbook*, volume 2: Special Topics and Techniques, chapter 19. CRC Press, second edition, Nov 2009.

[Tar55] Alfred Tarski. A LATTICE-THEORETICAL FIXPOINT THEOREM AND ITS APPLICATIONS. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[Tho97] Wolfgang Thomas. Languages, Automata, and Logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 389–455. Springer Berlin Heidelberg, 1997.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[Var82] Moshe Y. Vardi. THE COMPLEXITY OF RELATIONAL QUERY LANGUAGES (Extended Abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982.

[Var98] Moshe Y. Vardi. Reasoning about The Past with Two-Way Automata. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[Via01] Victor Vianu. A Web Odyssey: from Codd to XML. In Peter Buneman, editor, *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*. ACM, 2001.

[Vol99] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.

Erklärung

Ich erkläre hiermit, dass

- ich die vorliegende Dissertation *Static Analysis of Monadic Datalog on Finite Labeled Trees* selbstständig und ohne unerlaubte Hilfe angefertigt habe;
- ich mich weder bereits anderwärts um einen Doktorgrad im mathematisch-naturwissenschaftlichen Bereich beworben habe, noch einen solchen besitze;
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät der Humboldt-Universität zu Berlin vom 30. Juni 2014, veröffentlicht im Amtlichen Mitteilungsblatt Nr. 126/2014, bekannt ist.

Berlin, den 13. September 2016

André Frochaux