DISSERTATION

Query Answering in Expressive Description Logics

Techniques and Complexity Results

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

O. Univ. Prof. Dipl.-Ing. Dr. Thomas Eiter
Institut für Informationssysteme 184/3
Abteilung für Wissenbasierte Systeme

und weiterer Betreuung durch

Prof. Dipl.-Ing. Dr. Diego Calvanese
Freie Universität Bozen

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von

María Magdalena Ortiz de la Fuente
Matrikelnummer 0527305
Lerchenfelder Straße 39/37
1070 Wien

Wien, am 29. April 2010

To my parents, Jesús and Rosa Elena,
to my sister, Rosa Elena,
and to my brother, Jesús.

Acknowledgments

I want to thank Thomas Eiter, my supervisor, for many reasons. I am grateful for all he has taught me, for his guidance, for how much time he has devoted to my research, and for all his support, encouragement and patience. My thanks extend to his wife, Katrin, not only for her patience and support despite so many overly-long meetings and tight deadlines, but also, and mostly, for sharing with us a bit of home and familiness, and for her courage when it is needed.

I also want to sincerely thank Diego Calvanese, for all his advise, for helping me learn so much, for being always willing to work with me—even when his schedule does not allow it—and for his friendship and trust.

Special thanks also to Carsten Lutz for the pleasant and motivating collaboration, and for contributing to making many parts of this thesis a bit more readable.

I am very grateful to all the people who made possible not only this thesis, but also the years that lead to it. I am most grateful to the Mexican Science and Technology Council (Conacyt), for its invaluable support. Thank you to Prof. Mauricio Osorio, my first advisor back in Mexico, for putting me on the track and giving me a good basis to build on.

Thank you to the Knowledge Base Systems group and to the Faculty of Informatics for letting me be part of the team. Special thanks to the people in and around the group, for all the coffee, sushi, and time together that made this important part of my life not only possible, but happy. Special thanks (and a little blame?) to Carlos Areces, without him I would most likely have landed elsewhere. Do you remember that I did not like Description Logics? And thanks to you and to Lu, for being just the right kind of friends, and doing it so well.

An enormous thanks to my parents, for all their love, for everything they have given me, and for being such wonderful parents. Gracias, los quiero muchísimo. To my brother and sister, for being always a fantastic part of my life, and for remaining close despite the distance that makes me miss them far too often. To my Šimkus family, for being my family, and to my big family, because it is so good to go back and be welcomed by such a great family. Thank you to those good friends—in Vienna, back in Mexico, and in a few other corners of the worlds—who always supply me with good times and memories, reasons to smile, encouragement and support, and thanks because you don't mind if I don't write up all the names.

And thank you, of course, to Mantas, for having so, so much to thank him for. What is there to say?

Abstract

Description Logics (DLs) are a family of logics specially tailored for knowledge representation and reasoning. They allow to model a domain of discourse by means of *knowledge bases*, which structure knowledge in terms of classes of objects (called *concepts*), and binary relations (called *roles*) between objects. DLs are appreciated for their ability to formally represent complex knowledge, but they generally do not provide means to *access* it. The traditional reasoning services are geared towards supporting domain conceptualization, and they are not sufficient in important applications where DLs are used to model *data repositories*. This has motivated research efforts aimed at extending DL reasoning services to support access by means of database-like query languages.

Query answering in DLs has received considerable attention in the last years. However, for *expressive DLs* that extend the full $\mathcal{ALC}$ with different combinations of additional concept and role constructors, few algorithms were available. Moreover, the existing approaches were computationally very demanding, and the precise computational costs of query answering remained unknown. In this thesis we study query answering in expressive DLs, explore novel reasoning techniques with the emphasis on worst-case optimal algorithms, and derive new computational complexity results. We explore two kinds of reasoning techniques.

First we use rich models of *automata on infinite forests* to develop a query answering algorithm that supports most of the popular constructs. This allows us to significantly push the frontier of decidability and complexity upper bounds, with respect to both the query language (two-way positive regular path queries) and the DLs (logics that support rich role assertions and different combinations of nominals, inverses and counting, such as $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, $\mathcal{ZOI}$, $\mathcal{SRIQ}$, $\mathcal{SROI}$ and $\mathcal{SROQ}$).

As second tool we employ *knots*, an instance of a technique known as *mosaics* in modal logic, to develop algorithms that allow for a more refined control of the sources of complexity, improving some existing upper bounds. For example, we show that for $\mathcal{ALCH}$ query answering is feasible in single exponential time—thus not harder than satisfiability testing—and that this positive result extends $\mathcal{SH}$ provided that the number of transitive roles involved in the query atoms is bounded. Our knot-based algorithms are optimal in data complexity, and support knowledge compilation and encodings into Datalog; this makes them appealing for implementation.

As an additional result we identify transitive roles and role inclusions as a combination of constructors that makes query answering 2ExpTime hard—that is, provably harder than satisfiability testing—in any extension of $\mathcal{ALC}$. This new source of complexity, in combination an analogous result for inverse roles due to Lutz, allows us to precisely characterize the precise computational complexity of query answering in a very wide range of expressive DLs.

Kurzfassung

Description Logics (DLs) sind eine Familie von Logiken die speziell entwickelt wurden, um Wissen zu repräsentieren und Schlüsse daraus zu ziehen. Sie erlauben es, einen bestimmten Wissensbereich mittels einer *Wissensbasis* zu modellieren. Diese strukturiert das Wissen in Klassen von Objekten (*Konzepte* genannt) und in binären Relationen zwischen Objekten (*Rollen* genannt). DLs werden vorallem geschätzt, weil sie es erlauben, komplexes Wissen formal darzustellen, wenn auch sie es für gewöhnlich nicht erlauben, auf dieses Wissen flexibel *zuzugreifen.* Die traditionellen Inferenzdienste sind auf die Domainkonzeptualisierung ausgerichtet, aber in wichtigen Anwendungen, in denen DLs für Datenzugriff verwendet werden, reichen diese nicht aus. Das hat eine Forschungsrichtung motiviert, die das Ziel hat, die traditionellen Inferenzdienste für DLs auf den Datenzugriff mittels Datenbankanfragesprachen auszuweiten.

Das Problem, Antworten auf Datenbankanfragen in DLs zu berechnen hat in den letzten Jahren großes Interesse geweckt. Allerdings sind noch wenige Algorithmen vorgeschlagen worden, die dieses Problem für ausdrucksstarke DLs lösen, die $\mathcal{ALC}$ mit verschiedenen Konzept- und Rollenkonstruktoren erweitern. Außerdem wären die bisher verfügbaren Techniken sehr komplex, und die genaue Komplexität des Problem war noch offen. In dieser Dissertation analysieren wir die Anfragebeantwortung in ausdrucksstarken DLs, wir schlagen neue worst-case optimale *Inferenztechniken* vor, und wir leiten neue Komplexitätsresultate für das Problem ab. Wir gehen das Problem mit zwei verschiedenen Techniken an.

Erstens verwenden wir ausdrucksstarke Automatenmodelle auf unendlichen Wäldern (eine Verallgemeinerung von Bäumen) und entwickeln einen neuen Algorithmus, der eine Vielzahl (tatsächlich fast alle) der verwendeten Konstrukte behandeln kann. Dieser ermöglicht es uns, die Entscheidbarkeitsgrenze und die oberen Komplexitätsschranken stark zu verbessern, sowohl bezüglich der Anfragesprache (two-way positive regular path queries), als auch bezüglich der DL (Logiken mit ausdrucksstarken Aussagen über Rollen, mit verschiedenen Kombinationen von Nominalkonzepten, inversen Rollen und beschränkten Rollen, wie zum Beispiel $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, $\mathcal{ZOI}$, $\mathcal{SRIQ}$, $\mathcal{SROI}$ und $\mathcal{SROQ}$).

Zweitens verwenden wir sogenannte *knots*, die eine Instanz der Mosaik-Technik aus der Modallogik sind, für Algorithmen. Die entwickelten Algorithmen erlauben es uns, einige der Komplexitätsquellen besser zu kontrollieren und somit obere Komplexitätsschranken zu verbessern. Wir zeigen zum Beispiel, dass für $\mathcal{ALCH}$ die Anfragebeantwortung in einfacher exponentieller Zeit möglich ist und damit nicht härter als das Erfüllbarkeitsproblem in dieser Logik. Weiters zeigen wir, dass sich dieses positive Resultat auch auf $\mathcal{SH}$ erweitern lässt unter der Voraussetzung, dass die Anzahl transitiver Rollen in der Anfrage von vornherein beschränkt ist. Unsere knot-basierten Algorithmen sind optimal bezüglich der Datenkomplexität und unterstützen Wissenskompilierung und eine Kodierung von Anfragen in Datalog, wodurch eine

unmittelbare praktische Implementierung möglich ist.

Als weiteres Resultat identifizieren wir transitive Rollen und Rolleninklusionen als eine Kombination von Konstrukten, welche die Anfragenbeantwortung 2ExpTime-hart macht, d.h., beweisbar härter als das Erfüllbarkeitsproblem. Dieses Resultat gilt für alle Erweiterungen von $\mathcal{ALC}$. Diese neue Komplexitätsquelle, in Kombination mit einem verwandten Resultat von Lutz für inverse Rollen, erlaubt es uns, die genaue Komplexität der Anfragenbeantwortung für eine grosse Anzahl von ausdrucksstarken DLs exakt zu bestimmen.

Contents

Chapter 1

Introduction

Representing knowledge about different domains and drawing inferences from this knowledge has been recognized a central challenge of Artificial Intelligence since its earliest days, and has evolved into the area that is now called *Knowledge Representation and Reasoning*. One of the fundamental goals of the field is to identify suitable formalisms, that on the one hand have sufficient expressive power to present the desired knowledge in a convenient form, and on the other hand allow for automatization of *reasoning services* that infer—as efficiently as possible—implicit information from the described knowledge More often than not, such formalisms are *logics*, broadly understood as formal languages with a precisely defined syntax and semantics. The identification of suitable logics, however, is a non-trivial process and a universal logic is elusive. Indeed, different application areas call for different levels of expressiveness and different reasoning services. Since, in general, more expressiveness comes at the price of less efficient reasoning, compromises must be made and the choice of a 'good trade-off' can vary from one setting to another.

This perspective of logics as means for knowledge representation and reasoning, and the study of different logics in terms of their trade-off between expressiveness and efficient inference, is a hallmark of *Description Logics (DLs)*, the family of languages studied in this thesis. Description Logics emerged in the late 1980s as an attempt to use formal logic (understood as the study of formal languages with a precisely defined syntax and semantics) as a tool for identifying knowledge representation formalisms, and to overcome certain limitations of other non-logic formalisms that lacked well-defined semantics. In little more than 20 years, they have evolved into a rich family of logics, and are now recognized as one of the most prominent family of formalisms for Knowledge Representation and Reasoning. The study of Description Logics is an active field of research, marked by the steady emergence of new challenges resulting from their deployment in new application areas.

Description Logics were designed to model a domain of discourse by means of *concepts*, which are classes of objects sharing common properties, and *roles*, which describe binary relations between the objects in the classes. The domain of discourse is then described by means of a *knowledge base*, which has an *intensional component* that constrains the relations between concepts and roles, and an *extensional component* that asserts the participation of specific objects in the concepts and roles. DLs are accompanied by various *reasoning services* that are aimed at extracting implicit information from a knowledge base. We refer to [BCM$^+$03] for a more extensive introduction to DLs.

There is a wide range of Description Logics that differ in their set of *constructors* for concepts and roles, and in the kind of statements they allow in knowledge bases. They can be roughly classified into two main types: *light-weight DLs* whose syntax is rather limited, but for which most reasoning problems can be efficiently solved, and *expressive DLs* that support a rich set of constructors and a flexible syntax, at the expense of higher computational complexity. Perhaps the most prominent reasoning task in DLs is determining *concept subsumption*, that is, given a pair of concepts C_1 and C_2, checking whether the knowledge base implies that every object that is an instance of C_1 is also an instance of C_2. In the case of expressive logics, deciding *satisfiability* (i.e., *consistency*) of a knowledge base can be viewed as a core reasoning task as other traditional problems efficiently reduce to it.

In the last decade, Description Logics have made their way into a few novel application areas. In particular, DLs and their reasoning facilities have been applied for many data management problems, including conceptual modeling, integration of data sources, and access to data sources by means of queries (see, e.g., [BLR03] and its references). In such contexts, it is common to view the intensional component of a DL knowledge base as a rich schema describing the organization of the data, and the extensional component as a (partial) instance of the described schema. The intensional component then acts as a high-level conceptual view of the data repository, that can be exploited to access the data in it. This perspective has also been advocated in the Semantic Web, a favorite application area for DLs, where it has led to the emergence of Ontology Based Data Access [PLC+08]. This trend brought along some new research challenges and, in particular, *query answering in Description Logics* has become a topic of active research, cf. [CDGL08, CDGL+07, GR09, GHLS08, Gli07, Lut08a, EGOŠ08, ELOŠ09b].

1.1 Motivation

Expressive DLs provide very rich means to structure knowledge, but provide virtually no tools to *query* a knowledge base. In particular, they allow one to ask whether an object is an instance of a concept, or whether a pair of objects are related by a role. However, a query, even a rather basic one, is expected to allow to join pieces of information in the spirit of SQL or Datalog queries that are common in databases. Unfortunately, the latter query languages are orthogonal to DLs in terms of expressiveness, and thus cannot be (directly) expressed in DLs. Indeed, many DLs can be viewed as fragments of the two-variable fragment of first-order logic, while the above query languages and their extensions require an unbounded number of variables. Furthermore, DLs allow only for a limited form of universal and existential quantification, analogous to the *guarded quantification* present in modal logics, which results in a relatively restricted way of talking about the relations between objects. The restriction placed in DLs ensure some positive features. Most DLs have some form of *forest* or *tree model property*, which means that a satisfiable knowledge base always has a model that resembles a tree and enjoys certain regularity. This property is usually admitted to be the crucial reason why modal and description logics are computationally so robust, and can be enriched with so many constructors without compromising the decidability of the formalism [Var97].

These notions will be made precise along the thesis, but for now we illustrate them with a simple example. We build a knowledge base describing genealogical relations between characters of the Greek mythology. It contains the following intensional information: every mortal has a (female) mother and a (male) father, every hero has a divine ancestor, and every princess has an ancestor who is a hero. The ancestors of deities are also deities, and every deity that is not a primordial god descends from a primordial god. Furthermore, if a god is primordial, then he must be Chronos. We are also given some extensional information in the form of facts about some individuals: we know that Perseus is a hero and the father of Telephus, that Telephus is

$$
\begin{array}{rcl}
\mathsf{Male} & \equiv & \neg\mathsf{Female} \\
\mathsf{Mortal} & \sqsubseteq & \neg\mathsf{Deity} \\
\mathsf{Mortal} & \sqsubseteq & \exists\mathsf{hasFather}.\mathsf{Male} \\
 & & \sqcap\ \exists\mathsf{hasMother}.\mathsf{Female} \\
\mathsf{Hero} & \sqsubseteq & \exists\mathsf{hasAncestor}.\mathsf{Deity} \\
\mathsf{Princess} & \sqsubseteq & \exists\mathsf{hasAncestor}.\mathsf{Hero} \\
\mathsf{Deity} & \sqsubseteq & \forall\mathsf{hasAncestor}.\mathsf{Deity} \\
\mathsf{Deity} \sqcap \neg\mathsf{Primordial} & \sqsubseteq & \exists\mathsf{hasAncestor}.\mathsf{Primordial} \\
\mathsf{Primordial} & \sqsubseteq & \mathsf{Deity} \\
\mathsf{Primordial} & \sqsubseteq & \{\mathrm{chronos}\} \\
 & & \mathsf{trans}(\mathsf{hasAncestor})
\end{array}
\qquad\qquad
\begin{array}{r}
\mathsf{Male}\,(\mathrm{perseus}) \\
\mathsf{Hero}\,(\mathrm{perseus}) \\
\mathsf{Hero}\,(\mathrm{telephus}) \\
\mathsf{hasFather}\,(\mathrm{telephus}, \mathrm{perseus}) \\
\exists\mathsf{hasMother}.(\mathsf{Princess})\,(\mathrm{telephus})
\end{array}
$$

Figure 1.1: A simple example Knowledge Base

$$
q_1(x) = \exists v_1, v_2, v_3.\quad \mathsf{Hero}(x) \wedge \mathsf{hasMother}(x, v_1) \wedge \mathsf{hasFather}(x, v_2) \wedge \mathsf{hasAncestor}(v_1, v_3) \wedge \mathsf{hasAncestor}(v_2, v_3)
$$

Figure 1.2: A simple example query

also a hero, and that the mother of Telephus is a princess. This information is represented, using the DL syntax, in Figure 1.1. Using the standard reasoning services of DLs we can infer, for example, that Chronos is not mortal, or that the mother of Telephus has a divine ancestor. But suppose that we are given the following simple query: we want to know if there is some hero x, whose parents have a common divine ancestor. This query can be expressed via the first-order formula in Figure 1.2. It is not hard to see that there is a positive answer to this query, as Telephus is a hero as required by the query. His father is a hero, and hence has a divine ancestor who is in turn a descendant of Chronos, the only primordial god. His mother has some ancestor who is a hero, and we can reason analogously to infer that she is also a descendant of Chronos. This query can not be answered using the standard reasoning services, and illustrates the goal we want to achieve by enhancing them with query answering capabilities.

There are close parallels between query answering in DLs and the more traditional relational databases where one poses SQL or Datalog queries over relations, possibly in presence of database *integrity constraints*. Indeed, the extensional component of a DL knowledge base can be viewed as a database, and the intensional component as a set of expressive integrity constraints. However, there are also important differences (see, e.g., [Hus94]). As opposed to relational databases, in DL-based knowledge representation one makes the *open-world assumption*, which means that everything that does not contradict the statements in the knowledge base is possible. For example, a query asking whether all children of Perseus are heroes would be answered negatively in our example knowledge base, since the existence of other children of Perseus can not be ruled out with the information we have. Furthermore, unlike in databases, in our setting one does not rule out the existence of domain objects that are not explicitly named in the extensional component of the knowledge base, i.e. we make the *open-domain assumption*. Thus query answering must be more refined as we must also take into account unnamed objects whose existence may be implied the knowledge base. For instance, we do not know the name of the object to which the variable v_1 can be matched, or the name of the ancestor of Telephus' mother who is a hero and has a divine ancestor. We note that the terminological information

can easily induce infinite structures (for example, in the situation where every object must have an ancestor and nobody can be his own ancestor) which must be taken into account for query answering.

1.2 State of the Art

The idea of combining database-like queries with variables and Description Logics was first brought forward in 1998, in two independent papers and in rather different contexts. In the more database-oriented approach, the authors of [CDGL98] presented an algorithm for deciding query containment under expressive constrains expressed in DLs. Also in 1998, Levy and Rousset [LR98a] proposed a rich knowledge representation language called CARIN that couples DLs and rules, and whose reasoning tasks can be seen as a generalization of query answering over DLs. The work of Tessaris [Tes01] directly advocates the idea of posing queries with variables in expressive DLs, and developed a query answering algorithm in an extension of $\mathcal{ALC}$, but imposing some syntactic restrictions on queries. The technique used there, called *rolling-up*, is a variation of the one used in [CDGL98, CDGL08]. Towards the middle of the last decade query answering in DLs became a main-stream branch of research in DLs, cf. [CDGL$^+$05a, OCE06, GHS06]. Several algorithms have been proposed, and some effort has been spent towards understanding the computational complexity of the problem.

Since then research has evolved in two main directions. In applications where DLs are used to formalize static data models and to query data sources that can be very large, special attention must be payed to the *data complexity*, which measures the efficiency of algorithms in terms of the size of the *extensional component*. As data complexity is known to be intractable even for standard reasoning tasks and in rather weak DLs (e.g., $\mathcal{ALE}$ [Sch94a]), fragments of expressive DLs with tractable data complexity are of particular interest. Specially tailored DLs, like the ones in the DL-Lite family [CDGL$^+$07] aim at limiting the expressiveness to the constructors most relevant for data modeling, in such a way that query answering is not only tractable—in fact, it has very low data complexity (inside logarithmic space)—but is also reducible to first-order queries, which can be answered using existing database technology. A quite good understanding of the boundaries of tractability and first-order rewritability in light-weight DLs has been achieved [CDGL$^+$05b, CGL$^+$07, ACKZ09], and several DLs for which query answering is still tractable (but outside logarithmic space) have been identified. This includes $\mathcal{EL}$ and some extensions of it, such as $\mathcal{EL}^+$ and $\mathcal{EL}^{++}$ [Ros07, KRH07, KL07], as well as other *Horn* fragments obtained by disallowing disjunction from expressive DLs. For example, the logic Horn-$\mathcal{SHIQ}$ that disallows disjunction from the prominent $\mathcal{SHIQ}$ is also tractable in data complexity [Mot06, EGOŠ08].

For expressive DLs data complexity is always intractable, and all tight upper bounds obtained so far—even for very expressive logics such as $\mathcal{SHIQ}$ [GHLS08], $\mathcal{SHOQ}$ [GHS08], or $\mathcal{ALCHOI}$ [OCE08]—match the coNP lower bound known for $\mathcal{ALE}$ since 1994 [Sch94a]. In this setting, research has aimed mostly at developing algorithms that are optimal in *combined complexity*, i.e., the complexity measured in terms of the combined size of the extensional component, the intensional component, and the query. The original algorithm from [CDGL98, CDGL08] gives a 2ExpTime upper bound for a DL containing $\mathcal{ALCQI}_{reg}$ (for containment of conjunctive queries without regular expressions). The same bound was obtained more recently for other expressive logics. In particular, it was shown for $\mathcal{ALCHIQ}$ in [HMS04], for $\mathcal{SHIQ}$ in [GHLS08, CEO07], and for $\mathcal{SHOQ}$ in [GHS08]. It was a long standing open problem whether these upper bounds were tight. For the DLs that contain the extension of $\mathcal{ALC}$ with inverse roles ($\mathcal{ALCI}$), Lutz closed the gap by showing 2ExpTime-hardness [Lut07]. In the absence of inverses, the same lower bound applies for $\mathcal{SH}$ [ELOŠ09b]. The precise complexity of these logics will be discussed

in Chapters 6 and 7.

The identification of other sources of complexity and the development of algorithms of lower complexity was hindered by the lack of suitable techniques. The results in [LR98a] were obtained by a clever adaptation of the standard tableaux algorithm for satisfiability that uses the size of the query as a parameter. The technique can be adapted to several DL constructs [OCE08], but not to transitive roles, and it has a major draw-back: it does not yield tight complexity upper bounds because it builds on sub-optimal tableau algorithms. The resolution-based approach employed in [HMS04] gives a tight upper bound for $\mathcal{ALCHIQ}$, but its extension to other DLs is not apparent. Transitive roles are also problematic, and the addition of nominals seems to result in the loss of the optimal complexity bounds [KM08].

The algorithms in [GHLS08] and [GHS08] employ the rolling-up technique from [Tes01, CDGL98], on unrestricted queries, and it is now known that the upper bounds arising from them are tight. Roughly, the central idea of rolling-up is to consider all the ways in which a query can possibly be matched in the tree-shaped models of the knowledge base, and to express each of them using a concept (i.e., to 'roll it up' into a concept). To decide if the query evaluates to false, one searches for a model of the knowledge base where all the rolled-up concepts are not satisfied, which in turn amounts to finding a model of an exponentially larger knowledge base. Characterizing correctly all the ways in which the query needs to be rolled up can get quite involved, and it is not always easy to adapt to different logics. As it turns out, in the absence of inverses and transitive roles, it is possible to use rolling up avoiding the exponential blow-up. This will be discussed in Chapter 6.

For many interesting DLs no query answering algorithms had been obtained until now. For example, the most recent proposal for the Web Ontology Languages, OWL 2 (cf. [CGHM$^+$08]), is based on a relatively new family of DLs which provides a very rich language for describing properties of roles and interactions between them. They seem particularly hard to handle with the techniques mentioned above. Indeed, only rolling-up allows for transitivity, and its presence is already hard to handle in $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$. A simple way to accommodate even richer role implications is not apparent.

Another 2ExpTime upper bound, which is now known to be optimal, was obtained for the DL $\mathcal{ALCQIb}_{reg}$ using automata on infinite trees [CEO07]. This technique is quite flexible and seems more likely to be adaptable to the logics mentioned above. While tree automata are usually hard to deploy to DLs supporting nominals, this is made easier by the recent appearance of some classes of rich automata on infinite forests. We build on such automata to obtain the most general algorithm and upper complexity bounds of this thesis.

1.3 Goal of the Thesis and Main Results

The main goal of this thesis is *to explore novel reasoning techniques for query answering in expressive Description Logics, that allow us to develop worst-case optimal algorithms and to gain a better understanding of the computational complexity of the problem.*

In more concrete terms, the main aspects considered in the thesis are the following:

Reasoning Techniques We develop novel techniques for answering queries in expressive DLs. In particular, we present two kinds of techniques:

Automata-based techniques We combine some rich models of automata on infinite forests [BLMV08] with the ideas developed in [CDGLV00] to deal with query containment in traditional (finite) databases. These techniques allow us to obtain an algorithm for query

answering that accommodates most of the popular DL constructs, showing decidability and complexity results for some of the most expressive DLs studied so far.

Techniques based on knots Knots [ŠE07] are an instance of mosaics as known in modal logic [Ném86, MM07]. We develop novel techniques for query answering based on knots, which allow for a more refined control of the sources of complexity and, in some cases, improved complexity upper bounds.

Computational Complexity We obtain some new upper and lower bounds for the complexity of query answering in expressive DLs. We extend the frontier of membership in 2ExpTime to some new logics, and obtain some new 3ExpTime upper bounds (which are not known to be tight); for some extensions of $\mathcal{ALC}$ we improve existing upper bounds from 2ExpTime to a tight ExpTime. Finally, we identify a new combination of constructors that makes query answering harder than satisfiability testing: transitive roles and role inclusions. With this new source of complexity (and the 2ExpTime-hardness of $\mathcal{ALCI}$ [Lut07]) we can show that all the given 2ExpTime upper bounds are tight.

1.4 Structure of this Thesis

The remainder of this thesis is organized as follows. In Chapter 2 we give a formal introduction to query answering in description logics. We define the syntax and semantics of all the DLs considered in this thesis, and also formally define the query languages and reasoning problems to be studied. We also fix the general notation to be used throughout the thesis.

The main contributions, which are presented in Chapters 3 to 7, are divided into two main parts. In the first part, Chapters 3 to 5, we use automata theoretic techniques to develop very general algorithms, with the aim of pushing the decidability frontier as far as we can:

- In Chapter 3, we focus on the $\mathcal{Z}$ family of DLs, which are very expressive extensions of $\mathcal{ALC}_{reg}$ capable of simulating $\mathcal{SROIQ}$ and its sublogics. We use *fully enriched automata* to develop an algorithm for deciding knowledge base satisfiability and show that the problem is ExpTime-complete for $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$, the three maximal sublogics of $\mathcal{ZOIQ}$ that respectively disallow nominals, inverses, and counting.

- In Chapter 4, we extend the knowledge base satisfiability algorithm for the $\mathcal{Z}$ family into an algorithm for answering a very rich class of queries, called *positive two-way regular path queries (P2RPQs)*. The algorithm is developed for *query entailment* and, as we show, can also be exploited for P2RPQ containment (in the case of $\mathcal{ZIQ}$, restricting the query on the left to be a simpler *conjunctive query*). Our algorithm allows to decide both problems in deterministic double exponential time, thus significantly extending the expressive DLs for which query answering has been shown to be feasible in 2ExpTime.

- In Chapter 5, we present an (exponential) reduction from $\mathcal{SROIQ}$ to $\mathcal{ZOIQ}$, that allows us to exploit the results of the previous two chapters in the logics of the $\mathcal{SR}$ family. In this way, we prove that knowledge base satisfiability in $\mathcal{SRIQ}$, $\mathcal{SROQ}$ and $\mathcal{SROI}$ is decidable in 2ExpTime, and that P2RPQ entailment and containment (restricting the query on the left to be a simpler *conjunctive query* in the case of $\mathcal{SRIQ}$) are decidable in 3ExpTime. These are the first such bounds, and also the first results concerning the feasibility of query answering in the $\mathcal{SR}$ family.

In the second part, Chapters 6 and 7, we develop two *knot-based techniques* for query answering, which illustrate how the knot technique allows for a more refined control of the sources of complexity.

- In Chapter 6, we describe an algorithm that decides query entailment by trying to build a counter-model for the query out of *knots*, which are small labeled trees of depth ≤ 1 [EŠ10]. We use *marked knots* to store information about the model and about partial query matches, and the existence of a counter-model is equivalent to the existence of a *coherent* set of marked knots. The algorithm is described for $\mathcal{ALCH}$ and its extension with inverse roles, $\mathcal{ALCHI}$. It runs in 2ExpTime in general, and in single exponential time for $\mathcal{ALCH}$. Both bounds are worst-case optimal. Interestingly, this shows that $\mathcal{ALCH}$ can be equipped with expressive queries without increasing the complexity w.r.t. standard reasoning. We also show that the algorithm is worst-case optimal in data complexity.

- In Chapter 7, we describe another algorithm that is also based on knots but has different features. It is developed for $\mathcal{SH}$, which extends the basic $\mathcal{ALC}$ with role inclusions and transitive roles. It works in a more constructive way: instead of building a counter-model for a given query, the algorithm computes subqueries whose match is implied by the intensional component, and then uses this information to answer the query for a given extensional component. In this way, the algorithm is modular and employs knowledge compilation. It works in double exponential time for $\mathcal{SH}$. By providing a matching lower bound, we show that this is worst-case optimal. For queries over $\mathcal{ALCH}$ knowledge bases, and for all queries over $\mathcal{SH}$ knowledge bases in which the occurrences of transitive roles satisfy some syntactic restrictions, the algorithm run in single exponential time. It has worst-case optimal data complexity, and also facilitates an encoding into (disjunctive) Datalog.

Finally, we summarize our results and give conclusions in Chapter 8. We includes a summary of the current landscape of computational complexity of query answering in expressive Description Logics, considering the complexity bounds derived in the thesis and elsewhere.

Chapter 2

Query Answering in Description Logic Knowledge Bases

In this chapter, we lay down the formal framework and formally define the problems studied in this thesis. We introduce the syntax and semantics of expressive Description Logics in general, and the specific logics that are considered in the remaining chapters. We also introduce query languages, and define formally query answering and other related reasoning problems that we will treat. To make this thesis self contained, basic notions of computational complexity are provided, and we also introduce the general notation for trees and forests that will be used in all remaining chapters.

2.1 Expressive Description Logics

Description Logics (DLs) [BCM$^+$03] are languages specifically designed for representing structured knowledge in terms of *concepts*, denoting sets of objects of the domain of interest, and *roles*, denoting binary relations between the instances of concepts. Concepts and roles are built from a vocabulary of atomic names by applying concept and role *constructors*. The domain of interest is then modeled through a *knowledge base (KB)*, which comprises a set of *axioms* at the intensional level, specifying the properties of concepts and roles, and *assertions* at the extensional level, that specify the participation of specific individuals in concepts and roles. Diverse combinations of concept and role constructors, and of axioms in the intensional component, result in a wide range of DLs. In this thesis, we focus on some of the so called *expressive Description Logics*, which are defined next.

We start by defining the *vocabulary*, which contains three kinds of basic symbols: *concept names*, that denote primitive classes used to model the domain of interest, *role names*, that denote the basic binary relations in the domain, and *individual names*, that are used for referring to specific objects in the domain and specifying their properties.

Definition 2.1.1 (DL vocabulary) *A* DL vocabulary *is a triple* $\langle \mathsf{N_R}, \mathsf{N_C}, \mathsf{N_I} \rangle$ *of countably infinite, pairwise disjoint sets. The elements of* $\mathsf{N_R}$ *are called* role names*, the elements of* $\mathsf{N_C}$ *are called* concept names*, and the elements of* $\mathsf{N_I}$ *are called* individuals*. We assume that* $\mathsf{N_C}$ *contains the special concept names* $\top$ *and* $\bot$*, respectively called the* top *and* bottom *concepts,*

and that $\mathsf{N_R}$ *contains the special role names* T *and* B*, respectively called the* universal *and the* empty *role.*

Throughout the thesis, we assume a fixed DL vocabulary $\langle \mathsf{N_R}, \mathsf{N_C}, \mathsf{N_I} \rangle$. The semantics of DLs is given in terms of *interpretations*, which fix the meaning of the symbols in the vocabulary. An interpretation has a domain which can be any non-empty set, and an interpretation function, that gives meaning to the symbols in the vocabulary by associating them with the domain. Each individual name is interpreted as one object in the domain. Each concept name, which intuitively corresponds to a set of objects in a class, is naturally interpreted as a subset of the domain. Finally, each role name corresponds to a binary relation between objects.

Formally, an *interpretation* for our DL vocabulary is defined as follows.

Definition 2.1.2 (Interpretation) *An* interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a non-empty set* $\Delta^{\mathcal{I}}$*, called the* domain *of* $\mathcal{I}$*, and a* valuation function $\cdot^{\mathcal{I}}$ *that maps:*

1. *each individual* $a \in \mathsf{N_I}$ *to an element* $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$*,*
2. *each concept name* $A \in \mathsf{N_C}$ *to a set* $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$*,*
3. *each role name* $p \in \mathsf{N_R}$ *to a binary relation* $p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, and*
4. *for the special concepts and roles,* $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$*,* $\bot^{\mathcal{I}} = \emptyset$*,* $\mathsf{T}^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, and* $\mathsf{B}^{\mathcal{I}} = \emptyset$*.*

A DL vocabulary is in fact a first-order predicate logic signature, that contains no function symbols and no variables, but only constants and predicates of arities one and two. The constants are the individuals in $\mathsf{N_I}$, the unary predicates are the concept names in $\mathsf{N_C}$, and the binary predicates are the role names in $\mathsf{N_R}$. Interpretations are just standard Tarski-style interpretations as used in predicate logic.

Example 2.1.3 *In our running examples, we will write knowledge bases that describe some Greek mythological characters and genealogical relations between them. The common vocabulary that we use contains the following symbols:*

- *The role names* hasParent*,* hasFather*,* hasMother*,* hasAncestor*, ...*
- *The concept names* Parent*,* Mortal*,* Deity*,* Male*,* Female*,* Hero*, ...*
- *The individual names* heracles*,* zeus*,* alcmene*,* perseus*,* eros*,* gaia*, ...*

By convention, concept names start with an upper case and role names with an lower case letter, while individual names are written in lower case Roman font.

Now we introduce the syntax and semantics of the different DLs we consider in this thesis. For each DL $\mathcal{L}$, we define the syntax of concepts and roles that are supported in $\mathcal{L}$. Then we define *ABox assertions*, *TBox axioms* and *RBox axioms* in $\mathcal{L}$. *Knowledge bases* in $\mathcal{L}$ are then uniformly defined as a triple of an ABox, a TBox and an RBox, which are sets of the respective axioms and assertions. In some cases, additional constraints are imposed on knowledge bases that, for example, restrict the interaction between components. To give the semantics, we first define how an interpretation function is extended to all concepts and roles in $\mathcal{L}$, and then define the satisfaction relation for assertions and axioms. This defines when an interpretation is a *model* of an ABox, TBox, and RBox, and of a knowledge base in $\mathcal{L}$. When the specific DL $\mathcal{L}$ is clear from the context, we simply talk about concepts, roles, ABoxes, knowledge bases, etc.

2.1.1 The Basic Expressive Description Logics $\mathcal{ALC}$ and $\mathcal{ALCH}$

The DL known as $\mathcal{ALC}$ is considered the 'basic' expressive description logic because it is the minimal one that supports unrestricted use of the basic concept constructors: conjunction, disjunction, negation, and existential and universal restrictions. The term *expressive Description Logics* usually refers to $\mathcal{ALC}$ and its extensions. The first DL that we introduce in this thesis is in fact $\mathcal{ALCH}$, a minor extension of $\mathcal{ALC}$. The other DLs treated in this thesis can be defined in a simple and uniform way as variations of this basic one.

Concepts and roles in $\mathcal{ALCH}$

We start with the syntax of *concepts* and *roles*. $\mathcal{ALCH}$ does not support any role constructors, that is, only role names p are roles (except for the special role T, which we discuss later). On the other hand, it provides the five 'basic' concept constructors: negation $\neg C$, conjunction $C_1 \sqcap C_2$, disjunction $C_2 \sqcup C_2$, and existential and universal restrictions which are expressions of the form $\exists p.C$ and $\forall p.C$, respectively.

Definition 2.1.4 ($\mathcal{ALCH}$ concepts and roles) *Each role name $p \in \mathsf{N_R} \setminus \{\mathsf{T}\}$ is a* role. Concepts C *obey the following grammar, where $A \in \mathsf{N_C}$ and p is a role:*

$$C, C_1, C_2 ::= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_2 \sqcup C_2 \mid \exists p.C \mid \forall p.C$$

The subconcepts *of C are all the concepts that occur syntactically in C.*

Since the definition of subconcepts is equally natural in most of the DLs that we introduce in this thesis, we usually omit it.

Assertions and axioms in $\mathcal{ALCH}$

Using the concepts and roles, we can write statements of two different kinds:

- At the *extensional level*, we can state that a certain individual participates in some class, or that some relation holds between a pair of individuals; we call this kind of statements *ABox assertions*.
- At the *intensional level*, we can specify general properties of concepts and roles, constraining the way they are interpreted and defining new concepts and roles in terms of other ones. These kinds of statements are called *axioms* and classified into two kinds: the ones that constrain the interpretation of concepts are called *TBox axioms*, and the ones that refer to roles are called *RBox axioms*.

DLs can differ in their concept and role constructors, but also in the kinds of assertions and axioms they allow. We define now the assertions and axioms of the basic DL $\mathcal{ALCH}$, which are allowed in all the DLs that we consider in this thesis. We will later define other DLs where additional assertions and axioms are allowed.

Definition 2.1.5 ($\mathcal{ALCH}$ ABox assertions, TBox axioms, RBox axioms) *For $\mathcal{ALCH}$, assertions and axioms are defined as follows.*

ABox assertions*:*

- *If C is a concept and $a \in \mathsf{N_I}$ is an individual, then $C(a)$ is a* concept membership assertion.
- *If p is a role and $a, b \in \mathsf{N_I}$ are individuals, then $p(a, b)$ is a* role membership assertion.
- *If $a, b \in \mathsf{N_I}$ are individuals, then $a \not\approx b$ is an* inequality assertion.

$\mathsf{Male}(\text{zeus})$	$\mathsf{hasFather}(\text{heracles}, \text{zeus})$
$\mathsf{Deity}(\text{zeus})$	$\mathsf{hasMother}(\text{heracles}, \text{alcmene})$
$\mathsf{Female}(\text{alcmene})$	$\mathsf{hasFather}(\text{alcmene}, \text{electryon})$
$\mathsf{Mortal}(\text{alcmene})$	$\mathsf{hasFather}(\text{electryon}, \text{perseus})$
$\mathsf{Hero}(\text{heracles})$	$\mathsf{hasFather}(\text{perseus}, \text{zeus})$

Figure 2.1: The Theogony ABox $\mathcal{A}_g$

TBox axioms*:*

- *If* C_1 *and* C_2 *are concepts, then* $C_1 \sqsubseteq C_2$ *is a* concept inclusion axiom (CIA).

RBox axioms*:*

- *If* p_1 *and* p_2 *are roles, then* $p_1 \sqsubseteq p_2$ *is a* role inclusion axiom (RIA).

Knowledge Bases in $\mathcal{ALCH}$

Now we can define *knowledge bases*, which are composed by ABox assertions, TBox axioms, and RBox axioms. They are naturally grouped into three sets: the ABox, the TBox, and the RBox. The definition of these three components is the same for all DLs.

Definition 2.1.6 (ABoxes, TBoxes, RBoxes) *For every DL* $\mathcal{L}$*, we define:*

- *An* ABox *in* $\mathcal{L}$ *is a finite set of* ABox assertions *in* $\mathcal{L}$*.*
- *A* TBox *in* $\mathcal{L}$ *is a finite set of* TBox axioms *in* $\mathcal{L}$*.*
- *An* RBox *in* $\mathcal{L}$ *is a finite set of* RBox axioms *in* $\mathcal{L}$*.*

The definition of $\mathcal{ALCH}$ knowledge bases is the basic one. It will remain essentially the same in all DLs we consider, modulo some additional constraints that will be imposed in some cases.

Definition 2.1.7 ($\mathcal{ALCH}$ knowledge bases) *In* $\mathcal{ALCH}$*, a* knowledge base *is a tuple* $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$*, where* $\mathcal{A}$*,* $\mathcal{T}$*, and* $\mathcal{R}$ *are, respectively, an ABox, a TBox, and an RBox in* $\mathcal{ALCH}$*.*

We remark that the distinction we make between TBox axioms and RBox axioms is not done in many Description Logics. It is a common practice to have one single intensional component in the knowledge base, possibly containing axioms about concepts and roles, and to call it TBox or *terminology*. We adopt separated TBox and RBox throughout the full thesis for uniformity, as some of the DLs we consider have rather complex RBoxes that can be more cleanly described this way.

We use the following notation throughout the thesis, for $\mathcal{ALCH}$ and for all other DLs.

Notation 2.1.8 *For a knowledge base, ABox, TBox, RBox, or concept* γ*, we respectively denote by* $\mathsf{N_C}(\gamma)$*,* $\mathsf{N_R}(\gamma)$*, and* $\mathsf{N_I}(\gamma)$ *the sets of concept names, role names, and individuals occurring in* $\mathcal{K}$*.*

Example 2.1.9 *Our first knowledge base describing genealogical relations between characters of the Greek mythology is the Theogony knowledge base* $\mathcal{K}_1^{theo} = \langle \mathcal{A}^{theo}, \mathcal{T}_1^{theo}, \mathcal{R}_1^{theo} \rangle$*, which is a*

$$\begin{aligned}
\mathsf{Male} &\equiv \neg\mathsf{Female} && (2.1)\\
\mathsf{Mortal} &\sqsubseteq \neg\mathsf{Deity} && (2.2)\\
\mathsf{Primordial} &\sqsubseteq \mathsf{Deity} && (2.3)\\
\neg\mathsf{Primordial} &\sqsubseteq \exists\mathsf{hasFather}.\mathsf{Male} \sqcap \exists\mathsf{hasMother}.\mathsf{Female} && (2.4)\\
\mathsf{Deity} &\sqsubseteq \forall\mathsf{hasParent}.\mathsf{Deity} && (2.5)\\
\mathsf{hasMother} &\sqsubseteq \mathsf{hasParent} && (2.6)\\
\mathsf{hasFather} &\sqsubseteq \mathsf{hasParent} && (2.7)
\end{aligned}$$

Figure 2.2: The $\mathcal{ALCH}$ TBox $\mathcal{T}_1^{theo}$ (2.1)–(2.5) and RBox $\mathcal{R}_1^{theo}$ (2.6)–(2.7)

knowledge base in $\mathcal{ALCH}$.[1] *The ABox $\mathcal{A}^{theo}$ is presented in Figure 2.1, the TBox $\mathcal{T}_1^{theo}$ and the RBox $\mathcal{R}_1^{theo}$ can be found in Figure 2.2. In the examples, we use $E_1 \equiv E_2$ (for E_i a concept or role) as a shortcut for the two axioms $E_1 \sqsubseteq E_2$ and $E_2 \sqsubseteq E_1$.*

Intuitively, the assertions in the ABox $\mathcal{A}^{theo}$ indicate that the individual named Zeus is a male deity, while the individual Alcmene is female and mortal. There is an individual named Heracles that is a hero, and that has Zeus as a father and Alcmene as a mother. Alcmene has a father named Electryon, Electryon has Perseus as a father, and Perseus has Zeus as a father.

The CIA (2.1) *in the TBox $\mathcal{T}_1^{theo}$ ensures that in the models of $\mathcal{K}_1^{theo}$ the domain is partitioned into males and females, and the CIA* (2.2) *indicates that mortals can not be deities. The concept 'primordial' is intended to contain the primordial deities who appeared at the beginning of the universe, and who are ancestors of all other deities; the CIA (2.3) asserts that they are deities. The CIA* (2.4) *indicates that everyone, except the primordial gods, must have a mother and a father. The last CIA* (2.5) *says that the parents of a deity must be deities, and the two RIAs in the RBox make sure that mothers and fathers are also parents.*

Semantics of $\mathcal{ALCH}$

The interpretation function, which fixes the meaning of the symbols in the vocabulary, is extended to all concepts and roles by means of an inductive definition for each of the concept and role constructors. The definition is such that each concept is mapped to a set of domain elements, and each role to a binary relation over the domain. For $\mathcal{ALCH}$, which only supports concept constructors, the interpretation of atomic concepts is extended to all concepts inductively.

Definition 2.1.10 (semantics of $\mathcal{ALCH}$ concepts) *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. The function $\cdot^{\mathcal{I}}$ is inductively extended to all $\mathcal{ALCH}$ concepts as follows:*

$$\begin{aligned}
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}\\
(C_1 \sqcap C_2)^{\mathcal{I}} &= {C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}}\\
(C_1 \sqcup C_2)^{\mathcal{I}} &= {C_1}^{\mathcal{I}} \cup {C_2}^{\mathcal{I}}\\
(\exists p.C)^{\mathcal{I}} &= \{x \mid \exists y.(x,y) \in p^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\\
(\forall p.C)^{\mathcal{I}} &= \{x \mid \forall y.(x,y) \in p^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}
\end{aligned}$$

Now that we have fixed the semantics of concepts and roles, we can define the satisfaction of assertions and axioms. This is done in a natural way. The symbol $\sqsubseteq$ in the TBox and RBox

[1] The examples are only for illustrative purposes and their contents is not necessarily accurate. Some information was taken from `http://www.pantheon.org` and `http://en.wikipedia.org`.

axioms is understood as an 'is-a' relation. That is, a concept inclusion $C_1 \sqsubseteq C_2$ indicates that every object that is C_1 is also C_2, or to be more precise, that every object that participates in the interpretation of concept C_1 also participates in the interpretation of concept C_2. Similarly, a role inclusion $p_1 \sqsubseteq p_2$ indicates that every pair of objects that participates in p_1 also participates in p_2. Concept and role membership assertions in the ABox simply state that (the interpretation of) an individual participates in (the interpretation of) a concept, and that a pair of individuals participates in a role, respectively. Finally, an assertion of the form $a \not\approx b$ states that the individuals a and b can not be interpreted as the same domain element. This is closely related to the *unique name assumption*, sometimes made in Description Logics and related formalisms. Under the unique name assumption, each interpretation $\mathcal{I}$ must be such that $a^{\mathcal{I}} = b^{\mathcal{I}}$ only if $a = b$, that is, one domain element can not be the interpretation of two different individuals. We do not make this assumption, but it can be simulated using assertions $a \not\approx b$ for each pair of individuals.

Definition 2.1.11 (satisfaction of ABox assertions, TBox axioms and RBox axioms) *Let $\mathcal{I}$ be an interpretation. We define the* satisfaction *relation $\mathcal{I} \models \gamma$ for γ an assertion or axiom as follows.*

- *For ABox assertions, we have* $\mathcal{I} \models C(a)$ *if* $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $\mathcal{I} \models p(a,b)$ *if* $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in p^{\mathcal{I}}$, *and* $\mathcal{I} \models a \not\approx b$ *if* $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.
- *For TBox axioms, we have* $\mathcal{I} \models C_1 \sqsubseteq C_2$ *if* $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- *For TBox axioms, we have* $\mathcal{I} \models p_1 \sqsubseteq p_2$ *if* $p_1^{\mathcal{I}} \subseteq p_2^{\mathcal{I}}$.

Naturally, the models of an ABox, TBox or RBox are defined as the interpretations that satisfy all the assertions or axioms it contains, and an interpretation is a model of a knowledge base if it is a model of each of its components. This definition is the same for all the DLs that we treat in this thesis.

Definition 2.1.12 (satisfaction of ABoxes, TBoxes, RBoxes, and KBs; models) *Let $\mathcal{I}$ be an interpretation. Then*

- *$\mathcal{I}$ satisfies an ABox $\mathcal{A}$, if it satisfies every assertion in $\mathcal{A}$.*
- *$\mathcal{I}$ satisfies a TBox $\mathcal{T}$, if it satisfies every axiom in $\mathcal{T}$.*
- *$\mathcal{I}$ satisfies an RBox $\mathcal{R}$, if it satisfies every axiom in $\mathcal{R}$.*
- *$\mathcal{I}$ satisfies a knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, if it satisfies $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$.*

Given an ABox, TBox, RBox or knowledge base γ, we write $\mathcal{I} \models \gamma$, if $\mathcal{I}$ satisfies γ. If $\mathcal{I} \models \gamma$, then $\mathcal{I}$ is called a model *of γ.*

Example 2.1.13 *Consider an interpretation $\mathcal{I}_1 = (\Delta^{\mathcal{I}_1}, \cdot^{\mathcal{I}_1})$ with the domain $\Delta^{\mathcal{I}_1} = \{1, \ldots, 8\}$. We describe the interpretation function only for the symbols that occur in $\mathcal{K}_1^{theo}$. In particular,*

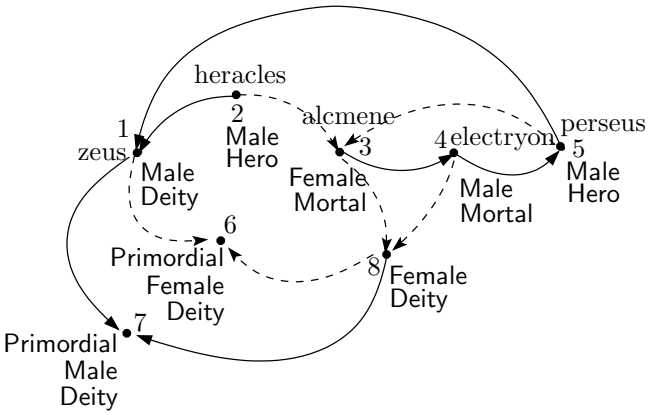

Figure 2.3: The interpretation $\mathcal{I}_1$

we have:

$$\begin{array}{rclrclrcl}
\mathsf{zeus}^{\mathcal{I}_1} & = & 1 & \mathsf{alcmene}^{\mathcal{I}_1} & = & 3 & \mathsf{perseus}^{\mathcal{I}_1} & = & 5 \\
\mathsf{heracles}^{\mathcal{I}_1} & = & 2 & \mathsf{electryon}^{\mathcal{I}_1} & = & 4 & & & \\
\mathsf{Male}^{\mathcal{I}_1} & = & \{1,2,4,5,7\} & \mathsf{Mortal}^{\mathcal{I}_1} & = & \{3,4\} & \mathsf{Hero}^{\mathcal{I}_1} & = & \{2,5\} \\
\mathsf{Female}^{\mathcal{I}_1} & = & \{3,6,8\} & \mathsf{Deity}^{\mathcal{I}_1} & = & \{1,6,7,8\} & \mathsf{Primordial}^{\mathcal{I}_1} & = & \{6,7\}
\end{array}$$

$$\begin{array}{rcl}
\mathsf{hasMother}^{\mathcal{I}_1} & = & \{(1,6),(2,3),(3,8),(4,8),(5,3),(8,6)\} \\
\mathsf{hasFather}^{\mathcal{I}_1} & = & \{(1,7),(2,1),(3,4),(4,5),(5,1),(8,7)\} \\
\mathsf{hasParent}^{\mathcal{I}_1} & = & \mathsf{hasMother}^{\mathcal{I}_1} \cup \mathsf{hasFather}^{\mathcal{I}_1}
\end{array}$$

The interpretation is depicted in Figure 2.3, where each node is labeled with the name of the individuals it interprets, and with the set of concepts in whose interpretation it participates. The hasFather *relation is represented by solid arrows, while* hasMother *is represented by dashed arrows, and* hasParent *comprises both kinds of arrows. The interpretation $\mathcal{I}_1$ satisfies $\mathcal{A}^{theo}$, $\mathcal{T}_1^{theo}$ and $\mathcal{R}_1^{theo}$, hence $\mathcal{I}_1 \models \mathcal{K}_1^{theo}$.*

$\mathcal{ALC}$, a sublogic of $\mathcal{ALCH}$

The basic DL $\mathcal{ALC}$ is defined like $\mathcal{ALCH}$, but it does not allow for role inclusions.

Definition 2.1.14 ($\mathcal{ALC}$ knowledge base) *$\mathcal{ALC}$ is the sublogic of $\mathcal{ALCH}$ that does not allow for any RBox axioms. That is, the only $\mathcal{ALC}$ RBox is $\emptyset$, and an $\mathcal{ALC}$ knowledge base is an $\mathcal{ALCH}$ knowledge base of the form $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \emptyset \rangle$.*

Example 2.1.15 *The RIAs* (2.6) *and* (2.7) *in Figure 2.2 are not allowed in $\mathcal{ALC}$, so we can not relate the roles* hasMother *and* hasFather *with* hasParent*. If we consider the knowledge base $\langle \mathcal{A}^{theo}, \mathcal{T}_1^{theo}, \emptyset \rangle$, the CIA* (2.5) *would not restrict the fillers of the* hasFather *and* hasMother *relation for a deity to be deities, Hence, in the absence of RIAs, the intended meaning of* (2.5) *would be captured better by a CIA* $\mathsf{Deity} \sqsubseteq \forall \mathsf{hasMother}.\mathsf{Deity} \sqcap \forall \mathsf{hasFather}.\mathsf{Deity}$ *instead.*

2.1.2 The $\mathcal{SH}$ Family and other Extensions of $\mathcal{ALCH}$

The DL $\mathcal{ALCH}$ can be extended with additional concept and role constructors, and by allowing other kinds of axioms. Some of the most prominent logics obtained this way are the ones in the so called $\mathcal{SH}$ family, which includes the DL $\mathcal{SHOIQ}$ and some of its sublogics. $\mathcal{SHOIQ}$ is a

very expressive DL that is closely related to the Web Ontology Language standard known as OWL-DL [PSHH04]. Most of the popular DL constructors are in $\mathcal{SHOIQ}$, and hence many of the DLs mentioned in the thesis can be defined as sublogics of it.

Concepts and roles in $\mathcal{SHOIQ}$

Definition 2.1.16 ($\mathcal{SHOIQ}$ concepts and roles) Atomic concepts B, concepts C *and* roles P, S, *obey the following grammar, where* $a \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, $p \in \mathsf{N_R} \setminus \mathsf{T}$, *and* $n \geq 0$*:*

$$\begin{aligned} B &::= A \mid \{a\} \\ C, C_1, C_2 &::= B \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists P.C \mid \forall P.C \mid \geqslant n\, S.C \mid \leqslant n\, S.C \\ P, S &::= p \mid p^- \end{aligned}$$

The inverse *of* $p \in \mathsf{N_R}$ *is* p^-*, and the inverse of* p^- *is* p*. To avoid expressions such as* $(p^-)^-$*, we denote by* $\mathsf{Inv}(P)$ *the inverse of the role* P*. Concepts of the form* $\{a\}$ *are called* nominals*, while concepts* $\geqslant n\, S.C$ *and* $\leqslant n\, S.C$ *are called* (qualified) number restrictions (QNRs)*.*

Assertions and axioms in $\mathcal{SHOIQ}$

In addition to the new role constructor p^- and the new concept constructors, $\mathcal{SHOIQ}$ extends $\mathcal{ALCH}$ with another kind of RBox axioms.

Definition 2.1.17 ($\mathcal{SHOIQ}$ ABox assertions, TBox axioms, RBox axioms) ABox assertions *and* TBox axioms *in* $\mathcal{SHOIQ}$ *are defined analogously to* $\mathcal{ALCH}$*, but allowing for* $\mathcal{SHOIQ}$ *concepts and roles where applicable. In addition to RIAs,* $\mathcal{SHOIQ}$ *RBoxes allow for* transitivity axioms (TAs)*, which are expressions* $\mathsf{trans}(P)$ *where* P *is a role.*

Knowledge bases in $\mathcal{SHOIQ}$

Knowledge bases in $\mathcal{SHOIQ}$ are defined essentially as for $\mathcal{ALCH}$, but must satisfy an additional constraint: the roles that occur in the number restrictions must be *simple*, which means that they can not be implied by roles whose extension has to be transitively closed. Intuitively, this allows us to count only the direct neighbors of a node, but not nodes that are further away in the models; it is well known that dropping this restriction results in an undecidable logic [HST00].

To formalize the notion of simple roles, we use the relation $\sqsubseteq^{\mathcal{R}}$, which will also be useful later. Intuitively $\sqsubseteq^{\mathcal{R}}$ relates each pair of roles P_1, P_2 such that $P_1^{\mathcal{I}} \subseteq P2^{\mathcal{I}}$ holds in every interpretation that satisfies $\mathcal{R}$.

Definition 2.1.18 (simple roles, $\mathcal{SHOIQ}$ knowledge bases) *For an RBox* $\mathcal{R}$*, we denote by* $\sqsubseteq^{\mathcal{R}}$ *the reflexive transitive closure of* $\{(P_1, P_2) \mid P_1 \sqsubseteq P_2$ *or* $\mathsf{Inv}(P_1) \sqsubseteq \mathsf{Inv}(P_2)$ *is in* $\mathcal{R}\}$*; we usually write* $\sqsubseteq^{\mathcal{R}}$ *in infix notation. A role* S *is* simple w.r.t. $\mathcal{R}$*, if there is no* P *such that* $P \sqsubseteq^{\mathcal{R}} S$ *and* $\mathsf{trans}(P) \in \mathcal{R}$*.*

A knowledge base in $\mathcal{SHOIQ}$ *is a triple* $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *consisting of an ABox* $\mathcal{A}$*, a TBox* $\mathcal{T}$ *and an RBox* $\mathcal{R}$*, such that all roles* S *occurring in a number restriction* $\leqslant n\, S.C$ *or* $\geqslant n\, S.C$ *are simple w.r.t.* $\mathcal{R}$*.*

Example 2.1.19 *Consider the* $\mathcal{SHOIQ}$ *knowledge base* $\mathcal{K}_2^{theo} = \langle \mathcal{A}^{theo}, \mathcal{T}_2^{theo}, \mathcal{R}_2^{theo} \rangle$*, with the ABox* $\mathcal{A}^{theo}$ *of Figure 2.1. The TBox* $\mathcal{T}_2^{theo}$ *contains* $\mathcal{T}_1^{theo}$ *from Figure 2.2 and the CIAs* (2.8)–(2.10) *in Figure 2.4. The RBox* $\mathcal{R}_2^{theo}$ *contains* $\mathcal{R}_1^{theo}$*, and additionally the RIAs* (2.11) *and* (2.12) *and the transitivity axiom* (2.13)*. The first CIA* (2.8) *in the figure illustrates the usefulness of number restrictions, which allow us to restrict the number of parents of each mortal to two.*

$$\begin{aligned}
\mathsf{Mortal} &\sqsubseteq \leqslant 2\,\mathsf{hasParent}.\top && (2.8)\\
\mathsf{Primordial} &\equiv \{\text{chronos}\} \sqcup \{\text{gaia}\} && (2.9)\\
\mathsf{Deity} &\sqsubseteq \mathsf{Primordial} \sqcup \exists \mathsf{hasAncestor}.\mathsf{Primordial} && (2.10)\\
\mathsf{hasParent}^- &\equiv \mathsf{hasChild} && (2.11)\\
\mathsf{hasParent} &\sqsubseteq \mathsf{hasAncestor} && (2.12)\\
&\quad \mathsf{trans}(\mathsf{hasAncestor}) && (2.13)
\end{aligned}$$

Figure 2.4: TBox axioms (2.8)–(2.9) and RBox axioms (2.11)–(2.13) in $\mathcal{SHOIQ}$

Note that this CIA combined with (2.1), (2.4), (2.6) *and* (2.7) *actually ensures that each mortal has exactly two parents: one mother that is female and one father that is male. The CIA* (2.9) *illustrates the power of nominals: it says that the primordial gods are exactly Chronos and Gaia. The third CIA,* (2.10)*, says that all non-primordial deities descend from a primordial god. The* hasChild *relation is exactly the inverse of* hasParent*, that is d has a parent d' iff d' has a child d* (2.11)*. All parents are ancestors* (2.12)*, and the ancestor relation is transitive* (2.13)*. That is, the ancestors of the ancestors of d are ancestors of d, for every d.*

Semantics of $\mathcal{SHOIQ}$

To give semantics to $\mathcal{SHOIQ}$ knowledge bases, we need to define the semantics of the new concept and role constructors.

Definition 2.1.20 (semantics of concepts and roles in $\mathcal{SHOIQ}$) *For every interpretation $\mathcal{I}$, we define:*

$$\begin{aligned}
(p^-)^{\mathcal{I}} &= \{(y,x) \mid (x,y) \in p^{\mathcal{I}}\}\\
\{a\}^{\mathcal{I}} &= \{a^{\mathcal{I}}\}\\
(\geqslant n\, S.C)^{\mathcal{I}} &= \{x \mid |\{y \mid (x,y) \in S^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}| \geq n\}\\
(\leqslant n\, S.C)^{\mathcal{I}} &= \{x \mid |\{y \mid (x,y) \in S^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}| \leq n\}
\end{aligned}$$

We also need to define the semantics of assertions and axioms, on which the semantics of knowledge bases depends.

Definition 2.1.21 (satisfaction of ABox assertions, TBox axioms and RBox axioms) *Let $\mathcal{I}$ be an interpretation. The* satisfaction *relation $\mathcal{I} \models \gamma$ if γ is an ABox assertion, a TBox axiom, or a RIA, is as for $\mathcal{ALCH}$.*

For transitivity axiom $\mathsf{trans}(P)$*, we have that then $\mathcal{I} \models \mathsf{trans}(P)$ if $P^{\mathcal{I}}$ is transitively closed, that is, if for every d_1, d_2, d_3 in $\Delta^{\mathcal{I}}$, $(d_1,d_2) \in P^{\mathcal{I}}$ and $(d_2,d_3) \in P^{\mathcal{I}}$ implies $(d_1,d_3) \in P^{\mathcal{I}}$.*

Example 2.1.22 *Recall the interpretation $\mathcal{I}_1$ from Example 2.1.13, and additionally let*

$$\begin{aligned}
\text{gaia}^{\mathcal{I}_1} &= 6\\
\text{chronos}^{\mathcal{I}_1} &= 7\\
\mathsf{hasChild}^{\mathcal{I}_1} &= \{(d',d) \in \Delta^{\mathcal{I}_1} \times \Delta^{\mathcal{I}_1} \mid (d,d') \in \mathsf{hasParent}^{\mathcal{I}_1}\},
\end{aligned}$$

and let $\mathsf{hasAncestor}^{\mathcal{I}_1}$ *be the transitive closure of* $\mathsf{hasParent}^{\mathcal{I}_1}$*. Then $\mathcal{I}_1 \models \mathcal{K}_2^{theo}$.*

DL	TAs	RIAs	inverses	nominals	QNRs
$\mathcal{ALC}$					
$\mathcal{ALCI}$			✓		
$\mathcal{ALCH}$		✓			
$\mathcal{ALCHI}$		✓	✓		
$\mathcal{SH}$	✓	✓			
$\mathcal{SHIQ}$	✓	✓	✓		✓
$\mathcal{SHOQ}$	✓	✓		✓	✓
$\mathcal{SHOI}$	✓	✓	✓	✓	
$\mathcal{ALCHOIQ}$		✓	✓	✓	✓
$\mathcal{SHOIQ}$	✓	✓	✓	✓	✓

Table 2.1: Some expressive DLs between $\mathcal{ALC}$ and $\mathcal{SHOIQ}$

Sublogics of $\mathcal{SHOIQ}$

There are many well known DLs that contain $\mathcal{ALC}$, and extend it with some of the features of $\mathcal{SHOIQ}$. The logic $\mathcal{S}$ is the extension of $\mathcal{ALC}$ with transitivity axioms. Both $\mathcal{ALC}$ and $\mathcal{S}$ can be extended with the additional features as follows: the presence of the letter $\mathcal{H}$ indicates that RIAs are allowed as RBox axioms, and the additional letters $\mathcal{I}$, $\mathcal{O}$ and $\mathcal{Q}$ respectively denote the presence of inverses as a role constructor, of nominals, and of number restrictions. Some of these extensions, which will be mentioned throughout the thesis, are listed in Table 2.1.

Example 2.1.23 *Recall the knowledge base $\mathcal{K}_2^{theo} = \langle \mathcal{A}^{theo}, \mathcal{T}_2^{theo}, \mathcal{R}_2^{theo} \rangle$ from the previous example. In DLs that do not have inverses, like $\mathcal{ALCH}$ and $\mathcal{SHQ}$, the RIA* (2.11) *can not be expressed, hence we can only use the relations* hasParent *and* hasChild *as two separate roles, and the intended relationship between them can not be enforced. In DLs that do not support transitivity axioms, like $\mathcal{ALC}$, we can only ensure that parents of an object d are its ancestors, but we can not relate d to, for example, the parents of its parents.*

2.1.3 The $\mathcal{Z}$ Family

Now we introduce the DL $\mathcal{ZOIQ}$ and its sublogics $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$. These logics are extensions of a rather well known DL, called $\mathcal{ALC}_{reg}$, obtained by adding regular expressions over roles to $\mathcal{ALC}$ [Sch91, Baa91]. $\mathcal{ALC}_{reg}$ is just a syntactic variation of Propositional Dynamic Logic (PDL) [HKT00], a prominent formalism for reasoning about the behavior of programs. The expressive power of $\mathcal{ALC}_{reg}$ can be further increased by adding some of the additional constructs mentioned above, like inverse roles $\mathcal{I}$, quantified number restrictions $\mathcal{Q}$, and nominals $\mathcal{O}$. Some of the extensions have a natural counterpart in the PDL setting. For example, inverse roles are the DL equivalent to converse programs in PDL, and adding them to $\mathcal{ALC}_{reg}$ results in a DL equivalent to *converse PDL*. Number restrictions are not usually considered in PDL, but they are closely related to *functional programs*. Nominals, on the other hand, have no natural equivalent in the context of PDL, but have been considered in extensions of $\mathcal{ALC}_{reg}$. Please see [CDG03] for references and discussion.

We consider extensions of $\mathcal{ALC}_{reg}$ with role inclusions, inverses, nominals and number restrictions as in $\mathcal{SHOIQ}$, and additionally allow to use conjunction, disjunction and difference of roles as role constructors. We refer to these combinations as *safe Boolean role expressions*. Finally, we include as an additional feature some special concepts of the form $\exists S.\mathsf{Self}$ that express a certain form of local reflexivity [HKS05]. This kind of concepts had not been considered before

in the extensions of $\mathcal{ALC}_{reg}$, but as we will see, their addition enables these logics to simulate some other prominent DLs. Note that transitivity can be expressed in $\mathcal{ALC}_{reg}$, hence it is not natural to add transitivity axioms.

$\mathcal{ZOIQ}$ is an alternative name for the DL $\mathcal{ALCHOIQ}b^{\mathsf{Self}}_{reg}$, which extends $\mathcal{ALC}$ with role inclusion axioms ($\mathcal{H}$), nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), qualified number restrictions ($\mathcal{Q}$), regular expressions over roles (reg), safe Boolean role expressions (b), and concepts of the form $\exists S.\mathsf{Self}$ (Self).[2] Its sublogics $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$, also known as $\mathcal{ALCHOIQ}b^{\mathsf{Self}}_{reg}$, $\mathcal{ALCHIQ}b^{\mathsf{Self}}_{reg}$, $\mathcal{ALCHOQ}b^{\mathsf{Self}}_{reg}$, and $\mathcal{ALCHOI}b^{\mathsf{Self}}_{reg}$, respectively, are introduced below.

Concepts and roles in $\mathcal{ZOIQ}$

Definition 2.1.24 ($\mathcal{ZOIQ}$ concepts and roles) Atomic concepts B, concepts C, atomic roles P, simple roles S, *and* roles R, *obey the following grammar, where* $a \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, $p \in \mathsf{N_R}$, *and* $p \neq \mathsf{T}$*:*

$$\begin{aligned} B &::= A \mid \{a\} \\ C, C_1, C_2 &::= B \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geqslant n\, S.C \mid \leqslant n\, S.C \mid \exists S.\mathsf{Self} \\ P &::= p \mid p^- \\ S, S_1, S_2 &::= P \mid S_1 \cap S_2 \mid S_1 \cup S_2 \mid S_1 \setminus S_2 \\ R, R_1, R_2 &::= \mathsf{T} \mid S \mid R_1 \cup R_2 \mid R_1 \circ R_2 \mid R^* \mid id(C) \end{aligned}$$

We may refer to a simple role S as a Boolean role expression*, and to an arbitrary role R as a* regular role expression*.*

In $\mathcal{ZOIQ}$, we may use the term expression *to refer to a concept or a role. The* subconcepts*,* subroles*, and* subexpressions *of an expression E are the concepts, roles and expressions that occur syntactically in E.*

Note that in $\mathcal{ZOIQ}$, a concept may have roles as subexpressions, while a role can have concepts as subexpressions.

Recall the notation from 2.1.8, and in particular that $\mathsf{N_R}(\gamma)$ denotes the set of role names occurring in a knowledge base, ABox, TBox, or concept γ. For logics with inverse roles and for logics with nominals, we also use the following notation:

Notation 2.1.25 *For a knowledge base, ABox, TBox, or concept γ, we use the following notations for atomic roles and atomic concepts:*

$$\begin{aligned} \overline{\mathsf{N_R}}(\gamma) &= \mathsf{N_R}(\gamma) \cup \{p^- \mid p \in \mathsf{N_R}(\gamma)\} \\ \mathsf{N_{CI}}(\gamma) &= \mathsf{N_C}(\gamma) \cup \{\{a\} \mid \{a\} \textit{ occurs in } \gamma\} \end{aligned}$$

Assertions and axioms in $\mathcal{ZOIQ}$

Assertions and axioms in $\mathcal{ZOIQ}$ are very similar to those in $\mathcal{ALCH}$, but only simple roles are allowed in the ABox and in the RBox.

Definition 2.1.26 ($\mathcal{ZOIQ}$ ABox assertions, TBox axioms, RBox axioms) *In $\mathcal{ZOIQ}$,* ABox assertions *are inequality and concept membership assertions as in $\mathcal{ALCH}$ (but allowing for $\mathcal{ZOIQ}$ concepts), and* role membership assertions *$S(a, b)$ where S is a simple role and $a, b \in \mathsf{N_I}$ are individuals.*

TBox axioms *are CIAs defined in the usual way, and* RBox axioms *are defined as follows:*

[2] Following the notation conventions in [BCM+03], and using s to denote $\exists S.\mathsf{Self}$ as a concept constructor (to our knowledge, there is no standard symbol for it) the logic $\mathcal{ALCHOIQ}b^{\mathsf{Self}}_{reg}$ would be named $\mathcal{ALCHOQ}s^{-1,\cap,\cup,\setminus,\circ,*,id}$. However, this notational convention does not prescribe how to indicate that regular role expressions are not allowed in Boolean roles, Self concepts and number restrictions.

$$\mathsf{Deity} \sqsubseteq \exists(\mathsf{hasMother} \cup \mathsf{hasFather})^*.\mathsf{Primordial} \tag{2.14}$$
$$\{\text{narcissus}\} \sqsubseteq \exists\mathsf{isInLoveWith.Self} \tag{2.15}$$
$$\exists\mathsf{hasParent.Self} \sqsubseteq \bot \tag{2.16}$$
$$\mathsf{hasDirectRelative} \equiv \mathsf{hasParent} \cup \mathsf{hasChild} \tag{2.17}$$
$$\mathsf{hasParent} \cap \mathsf{hasChild} \sqsubseteq \mathsf{B} \tag{2.18}$$

Figure 2.5: TBox axioms (2.14)–(2.15) and RBox axioms (2.17)–(2.18) in $\mathcal{ZOIQ}$

- *if S_1 and S_2 are simple roles, then $S_1 \sqsubseteq S_2$ is a* (Boolean) role inclusion axiom (BRIA).

Note that all BRIAs are RIAs.

Knowledge bases in $\mathcal{ZOIQ}$

Knowledge bases in $\mathcal{ZOIQ}$ are as in $\mathcal{ALCH}$.

Definition 2.1.27 ($\mathcal{ZOIQ}$ knowledge bases) *In $\mathcal{ZOIQ}$, a knowledge base is a triple $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ consisting of an ABox $\mathcal{A}$, a TBox $\mathcal{T}$ and an RBox $\mathcal{R}$.*

Example 2.1.28 *To illustrate the expressiveness of $\mathcal{ZOIQ}$ we extend the first $\mathcal{ALCH}$ knowledge base $\mathcal{K}_1^{theo}$ from Example 2.1.9 with some additional axioms. We consider the $\mathcal{ZOIQ}$ knowledge base $\mathcal{K}^{theo} = \langle \mathcal{A}^{theo}, \mathcal{T}_3^{theo}, \mathcal{R}_3^{theo} \rangle$, where $\mathcal{A}^{theo}$ is the ABox in Figure 2.1, the TBox $\mathcal{T}_2^{theo}$ contains $\mathcal{T}_1^{theo}$ from Figure 2.2 and additionally the CIAs* (2.14)–(2.15) *in Figure 2.5, and the RBox $\mathcal{R}_2^{theo}$ contains $\mathcal{R}_1^{theo}$ and the BRIA* (2.17).

In $\mathcal{ZOIQ}$, instead of defining a role $\mathsf{hasAncestor}$ *that is transitive as we did in $\mathcal{SHOIQ}$, we can use regular expressions such as* $(\mathsf{hasMother} \cup \mathsf{hasFather}) \circ (\mathsf{hasMother} \cup \mathsf{hasFather})^*$. *Hence the CIA* (2.14) *ensures, similarly to* (2.10)*, that all deities descend from some primordial god. Using Self concepts we can express, for example,* (2.15) *that Narcissus is in love with himself, and* (2.16) *that no one can be his own parent. The BRIA* (2.17) *allows us to define 'direct relatives' as the union of parents and children, and the BRIA* (2.18) *says that somebody's child can not be also his parent.*

Semantics of $\mathcal{ZOIQ}$

To define the semantics, we first need to define how the interpretation function $\cdot^{\mathcal{I}}$ is extended to the concept and role constructors that are new in $\mathcal{ZOIQ}$.

Definition 2.1.29 (semantics of $\mathcal{ZOIQ}$ concepts and roles) *For every interpretation $\mathcal{I}$, we define:*

$$(\exists S.\mathsf{Self})^{\mathcal{I}} = \{x \mid (x,x) \in S^{\mathcal{I}}\}$$

$$\begin{aligned}
(S \cap S')^{\mathcal{I}} &= S^{\mathcal{I}} \cap S'^{\mathcal{I}} \\
(R \cup R')^{\mathcal{I}} &= R^{\mathcal{I}} \cup R'^{\mathcal{I}} \\
(S \setminus S')^{\mathcal{I}} &= S^{\mathcal{I}} \setminus S'^{\mathcal{I}} \\
(R \circ R')^{\mathcal{I}} &= R^{\mathcal{I}} \circ R'^{\mathcal{I}} \\
(R^*)^{\mathcal{I}} &= (R^{\mathcal{I}})^* \\
(id(C))^{\mathcal{I}} &= \{(x,x) \mid x \in C^{\mathcal{I}}\}
\end{aligned}$$

Here, we override the $\circ$ operator to denote the composition of two binary relations, and $\cdot^*$ to denote the reflexive transitive closure of a binary relation. That is, for binary relations rel_1 and rel_2 with $rel_i \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, we define $rel_1 \circ rel_2 = \{(d_1, d_3) \mid \exists d_2 \in \Delta^{\mathcal{I}} : (d_1, d_2) \in rel_1 \text{ and } (d_2, d_3) \in rel_2\}$, and ${rel_1}^*$ denotes the smallest relation containing rel_1 such that $(d, d) \in {rel_1}^*$ for every $d \in \Delta^{\mathcal{I}}$, and such that $(d_1, d_2) \in {rel_1}^*$ and $(d_2, d_3) \in {rel_1}^*$ imply $(d_1, d_3) \in {rel_1}^*$.

Given the semantics of concepts and roles, the satisfaction relation for assertions and axioms is as in $\mathcal{ALCH}$.

Sublogics of $\mathcal{ZOIQ}$

Finally, we introduce the sublogics of $\mathcal{ZOIQ}$ that we discuss in the thesis.

Definition 2.1.30 ($\mathcal{ZIQ}$, $\mathcal{ZOQ}$ and $\mathcal{ZOI}$) *We consider three sublogics of $\mathcal{ZOIQ}$ that result from disallowing different constructors in concepts and roles:*

- *$\mathcal{ZIQ}$ disallows nominals $\{a\}$;*
- *$\mathcal{ZOQ}$ disallows inverse roles p^-;*
- *$\mathcal{ZOI}$ disallows number restrictions $\geqslant n\, S.C$, $\leqslant n\, S.C$.*

2.1.4 The $\mathcal{SR}$ Family

$\mathcal{SROIQ}$ is a rather well known DL, because it was proposed as the basis for the most recent Web Ontology Language proposal, OWL 2 [CGHM+08]. It is an extension of $\mathcal{SHOIQ}$, the DL that extends $\mathcal{SHIQ}$ with nominals (see Definition 2.1.24). Its sublogics $\mathcal{SRIQ}$, $\mathcal{SROQ}$ and $\mathcal{SROI}$ are analogous to $\mathcal{ZIQ}$, $\mathcal{ZOQ}$ and $\mathcal{ZOI}$.

The most prominent feature of the logics in the $\mathcal{SR}$ family are complex role inclusion axioms of the form $P_1 \circ \cdots \circ P_n \sqsubseteq P$. It is also possible to explicitly state certain properties of roles like transitivity, (ir)reflexivity and disjointness. Some of these additions increase the expressivity of the logic, while others are just 'syntactic sugar' and are intended to be useful for ontology engineering. We recall the definition of $\mathcal{SROIQ}$ from [HKS06], borrowing some notation from [Kaz08].

Concepts and roles in $\mathcal{SROIQ}$

As usual, we start by defining concepts and roles. Concepts are essentially as for $\mathcal{ZOIQ}$, and roles are simpler, as the only role constructors are inverses and role composition $\circ$.

Definition 2.1.31 ($\mathcal{SROIQ}$ concepts and roles) *In $\mathcal{SROIQ}$,* Atomic concepts B, concepts C, atomic roles P, S, *and* roles R, *obey the following grammar, where $a \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, $p \in \mathsf{N_R} \setminus \{\mathsf{T}\}$:*

$$\begin{aligned} B &::= A \mid \{a\} \\ C, C_1, C_2 &::= B \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall P.C \mid \exists P.C \mid \forall P.C \mid \exists \mathsf{T}.C \mid \forall \mathsf{T}.C \mid \\ &\quad \geqslant n\, S.C \mid \leqslant n\, S.C \mid \exists S.\mathsf{Self} \\ P, S &::= p \mid p^- \\ R, R_1, R_2 &::= S \mid R_1 \circ R_2 \end{aligned}$$

Roles of the form $P_1 \circ \cdots \circ P_n$ may be called role chains.

Note that T may only occur in universal and existential restrictions, and that it is not a $\mathcal{SROIQ}$ role in general.

Assertions and axioms in $\mathcal{SROIQ}$

$\mathcal{SROIQ}$ supports some assertions and axioms that were not present in the other logics so far. In particular, the rich RBox axioms are its main distinguishing feature.

Definition 2.1.32 ($\mathcal{SROIQ}$ ABox assertions, TBox axioms, RBox axioms) *In $\mathcal{SROIQ}$,* ABox assertions *are as follows:*

- *If C is a concept and $a \in \mathsf{N_I}$ an individual, then $C(a)$ is a* concept membership assertion.
- *If P is an atomic role and $a, b \in \mathsf{N_I}$ are individuals, then $P(a, b)$ is a* (positive) role membership assertion.
- *If S is an atomic role and $a, b \in \mathsf{N_I}$ are individuals, then $\neg S(a, b)$ is a* (negative) role membership assertion.
- *If $a, b \in \mathsf{N_I}$ are individuals, then $a \not\approx b$ is an* inequality assertion.

TBox axioms *are CIAs, defined as usual.* RBox axioms *are as follows:*

- *If R is a role chain and P is an atomic role, then $R \sqsubseteq P$ is a* complex role inclusion axiom (CRIA). *Note that $R \sqsubseteq P$ is a RIA in case R is atomic.*
- *If P, S, S' are atomic roles, then the following are* role property axioms:[3]

$$\mathsf{Ref}(P), \qquad \mathsf{Irr}(S), \qquad \mathsf{Asy}(S), \qquad \textit{and} \qquad \mathsf{Dis}(S, S'),$$

Knowledge Bases in $\mathcal{SROIQ}$

To define $\mathcal{SROIQ}$ knowledge bases, we need some additional conditions that were designed to ensure decidability. In particular, we need a notion called *regularity* and, similarly to $\mathcal{SHOIQ}$, we must define *simple roles*, and restrict the roles occurring in certain positions to be simple. As for $\mathcal{SHOIQ}$, we define a relation $\sqsubseteq^{\mathcal{R}}$ that contains the pairs R, P of roles such that $R^{\mathcal{I}} \subseteq P^{\mathcal{I}}$ for each model $\mathcal{I}$ of $\mathcal{R}$, but the definition is more involved due to the presence of role chains in the role inclusion axioms.

Definition 2.1.33 (regular RBoxes, simple roles, knowledge bases) *An RBox $\mathcal{R}$ is* regular*, if there exists a strict partial order $\prec$ on the set $\overline{\mathsf{N_R}}$ of all atomic roles such that $\mathsf{Inv}(P) \prec P'$ iff $P \prec P'$ for every $P, P' \in \overline{\mathsf{N_R}}$, and such that every CRIA in $\mathcal{R}$ is of one of the following forms:*

- $P \circ P \sqsubseteq P$,
- $P^- \sqsubseteq P$,
- $R \sqsubseteq P$,
- $R \circ P \sqsubseteq P$, *or*
- $P \circ R \sqsubseteq P$,

where $R = P_1 \circ \cdots \circ P_n$ and $P_i \prec P$ for each $1 \leq i \leq n$.

The relation $\sqsubseteq^{\mathcal{R}}$ is the smallest relation such that

(i) $P \sqsubseteq^{\mathcal{R}} P$ *for every role $P \in \overline{\mathsf{N_R}}$ such that P or $\mathsf{Inv}(P)$ occurs in $\mathcal{R}$, and*

(ii) $P_1 \circ \cdots \circ P_n \sqsubseteq^{\mathcal{R}} P$ *for each $P_1 \circ \cdots \circ P_{i-1} \circ P' \circ P_{j+1} \circ \cdots \circ P_n \sqsubseteq^{\mathcal{R}} P$ such that $P_i \circ \cdots \circ P_j \sqsubseteq P' \in \mathcal{R}$ or $\mathsf{Inv}(P_j) \circ \cdots \circ \mathsf{Inv}(P_i) \sqsubseteq P' \in \mathcal{R}$, for some $P' \in \overline{\mathsf{N_R}}$ and $1 \leq i \leq j \leq n$.*

[3] We use the term *role property axiom* instead of *role assertions* used in [HKS05], since the latter is often used to refer to the ABox role membership assertions.

$$\mathsf{hasAncestor} \circ \mathsf{hasAncestor} \sqsubseteq \mathsf{hasAncestor} \tag{2.19}$$
$$\mathsf{hasParent} \sqsubseteq \mathsf{hasDirectRelative} \tag{2.20}$$
$$\mathsf{hasChild} \sqsubseteq \mathsf{hasDirectRelative} \tag{2.21}$$
$$\mathsf{hasParent} \circ \mathsf{hasParent}^{-} \sqsubseteq \mathsf{hasSibling} \tag{2.22}$$
$$\mathsf{hasParent} \circ \mathsf{hasSibling} \circ \mathsf{hasChild} \sqsubseteq \mathsf{hasCousin} \tag{2.23}$$
$$\mathsf{Dis}(\mathsf{hasParent}, \mathsf{hasChild}) \tag{2.24}$$

Figure 2.6: RBox axioms (2.19)–(2.24) in $\mathcal{SROIQ}$

A role P is simple *w.r.t. $\mathcal{R}$, if there are no roles $P_1, \ldots, P_n$ with $n \geq 2$ such that $P_1 \circ \cdots \circ P_n \sqsubseteq^{\mathcal{R}} P$.*

A $\mathcal{SROIQ}$ knowledge base *is a triple $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $\mathcal{A}$ in an ABox, $\mathcal{T}$ is a TBox, and $\mathcal{R}$ is* regular *RBox in $\mathcal{SROIQ}$, and, additionally, all roles S, S' are simple w.r.t. $\mathcal{R}$ for every*

- *number restriction $\geqslant n\, S.C$, $\leqslant n\, S.C$,*
- *concept of the form $\exists S.\mathsf{Self}$,*
- *negative role assertion $\neg S(a, b)$ in $\mathcal{A}$, and*
- *role property axiom $\mathsf{Irr}(S)$, $\mathsf{Asy}(S)$ or $\mathsf{Dis}(S, S')$ in $\mathcal{R}$.*

Semantics of $\mathcal{SROIQ}$

Note that all concept and role constructors in $\mathcal{SROIQ}$ are also present in $\mathcal{ZOIQ}$, and thus their semantics is already defined. We only need to give semantics to the special ABox assertions and RBox axioms.

Definition 2.1.34 (satisfaction of ABox assertions, TBox axioms and RBox axioms) *Let $\mathcal{I}$ be an interpretation. Then $\mathcal{I}$ satisfies a negated role membership assertion $\neg S(a, b)$, i.e., $\mathcal{I} \models \neg S(a, b)$, if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin S^{\mathcal{I}}$. For other ABox assertions, all TBox axioms, and CRIAs, satisfaction is defined as usual.*

For role property axioms, we have:

- *$\mathcal{I} \models \mathsf{Ref}(P)$ if $R^{\mathcal{I}}$ is* reflexive*, i.e., $(d, d) \in P^{\mathcal{I}}$ for every $d \in \Delta^{\mathcal{I}}$;*
- *$\mathcal{I} \models \mathsf{Irr}(S)$ if $S^{\mathcal{I}}$ is* irreflexive*, i.e., $(d, d) \notin S^{\mathcal{I}}$ for every $d \in \Delta^{\mathcal{I}}$;*
- *$\mathcal{I} \models \mathsf{Asy}(S)$ if $S^{\mathcal{I}}$ is* asymmetric*, i.e., $(d, d') \in S^{\mathcal{I}}$ implies $(d', d) \notin S^{\mathcal{I}}$; and*
- *$\mathcal{I} \models \mathsf{Dis}(S, S')$ if the relations S and S' are* disjoint*, i.e., $S^{\mathcal{I}} \cap S'^{\mathcal{I}} = \emptyset$.*

Example 2.1.35 *Our example $\mathcal{SROIQ}$ knowledge base $\mathcal{K}^{theo} = \langle \mathcal{A}^{theo}, \mathcal{T}_4^{theo}, \mathcal{R}_4^{theo} \rangle$ has the usual ABox $\mathcal{A}^{theo}$. The TBox $\mathcal{T}_4^{theo}$ contains most of the TBox axioms from the previous examples: all the $\mathcal{ALCH}$ axioms in $\mathcal{T}_1^{theo}$ (Figure 2.2), all the $\mathcal{SHOIQ}$ axioms in $\mathcal{T}_2^{theo}$ (Figure 2.4), and additionally the $\mathcal{ZOIQ}$ axioms* (2.15) *and* (2.16) *(Figure 2.5). The $\mathcal{SR}$ family differs from the other DLs we have considered so far mostly in the RBox, and this is reflected in our example: $\mathcal{R}_4^{theo}$ includes the RIAs* (2.6), (2.7), (2.11) *and* (2.12) *from the previous examples, and additionally the axioms* (2.19)–(2.24) *in Figure 2.6. The CRIA* (2.19) *is equivalent to* (2.13)*. The two CRIAs* (2.20) *and* (2.21) *simulate the right to left inclusion of* (2.17)*, that is, parents and children are direct relatives. Unlike $\mathcal{ZOIQ}$, $\mathcal{SROIQ}$ can not enforce the other direction,*

and in the models of $\mathcal{K}^{theo}$ the possible existence of direct relatives of an object that are neither its parents nor its children can not be ruled out in general. The CRIAs (2.22) *and* (2.19) *illustrate how we can express in $\mathcal{SROIQ}$ relatively complex relations, e.g., that certain relatives are siblings or cousins. Note that in $\mathcal{ZOIQ}$ we do not have these inclusions, but we can directly use expressions such as* $\mathsf{hasParent} \circ \mathsf{hasParent}^-$ *in the place of* $\mathsf{hasSibling}$ *in TBox axioms and ABox assertions to achieve a similar result. Finally, the role property axiom* (2.24) *says, like the $\mathcal{ZOIQ}$ BRIA* (2.18)*, that somebody's child can not be also his parent.*

Sublogics of $\mathcal{SROIQ}$

We consider three sublogics of $\mathcal{SROIQ}$ which are analogous to $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$.

Definition 2.1.36 (The DLs $\mathcal{SRIQ}$, $\mathcal{SROQ}$ and $\mathcal{SROI}$) *A knowledge base $\mathcal{K}$ in $\mathcal{SROIQ}$ is*

- *in $\mathcal{SRIQ}$, if no* nominal concepts $\{a\}$ *occur;*
- *in $\mathcal{SROQ}$, if no* inverse roles P^- *occur; and*
- *in $\mathcal{SROI}$, if no* number restrictions $\geqslant n\, S.C$, $\leqslant n\, S.C$ *occur.*

We remark that the definition of $\mathcal{SROIQ}$ presented here is a minor restriction of the original definition in [HKS06], but it is not less expressive. In particular, Horrocks et al. allow role property axioms of two additional forms. First, they allow transitivity axioms $\mathsf{trans}(P)$, as in $\mathcal{SHOIQ}$, which can be equivalently expressed using CRIAs of the form $P \circ P \sqsubseteq P$. Second, $\mathsf{Sym}(P)$ asserts that P is symmetric, which can be expressed as $P^- \sqsubseteq P$; please note that this is not supported in the $\mathcal{SROQ}$ fragment. ABox assertions $\neg R(a, b)$ for non-simple R are allowed in $\mathcal{SROIQ}$ [HKS06], but not in $\mathcal{SRIQ}$ [HKS05]. Our syntax allows us to express them in all logics with nominals, since the assertion $\neg R(a, b)$ can be equivalently rewritten as $\{a\} \sqsubseteq \forall R.\neg\{b\}$.

To allow for a more uniform definition of $\mathcal{SROIQ}$ and its sublogics, we have only allowed for the universal role in universal and existential restrictions. The definition of $\mathcal{SROIQ}$ in [HKS06] allows T to occur as an ordinary role everywhere except in RBoxes, while the definition of $\mathcal{SRIQ}$ in [HKS05] does not allow it. Our definition is not less expressive, as role membership assertions $\mathsf{T}(a, b)$ or $\mathsf{T}^-(a, b)$ are trivially satisfied in every interpretation and can be ignored, while $\neg\mathsf{T}(a, b)$ or $\neg\mathsf{T}^-(a, b)$ are trivially unsatisfiable and can be replaced by $\bot(a)$. Concepts of the form $\exists\mathsf{T}.\mathsf{Self}$ and $\exists\mathsf{T}^-.\mathsf{Self}$ are equivalent to $\top$. Hence, the only interesting consequence of our restrictions on the occurrences of the universal role are that one can not write number restrictions $\leqslant n\, \mathsf{T}.C$, $\geqslant n\, \mathsf{T}.C$ in our syntax. This is not a limitation, since they can be simulated using nominals. To simulate the effect of $\leqslant n\, \mathsf{T}.C$, we replace it by a fresh concept name $A_{(\leqslant n\, \mathsf{T}.C)}$ wherever it occurs in $\mathcal{K}$, and make sure that whenever an object satisfies $A_{(\leqslant n\, \mathsf{T}.C)}$ in a model, then the cardinality of the interpretation of C is bounded by n. The latter can be achieved using n fresh individuals $a_1, \ldots, a_n$, and a CIA

$$\exists\mathsf{T}.A_{(\leqslant n\, \mathsf{T}.C)} \sqcap C \sqsubseteq \{a_1\} \sqcup \cdots \sqcup \{a_n\},$$

which ensures that if the interpretation of $A_{(\leqslant n\, \mathsf{T}.C)}$ is non-empty, then there are at most n instances of C. Dually, $\geqslant n\, \mathsf{T}.C$ means that in every model the cardinality of the interpretation of C has to be at least n. We also replace $\geqslant n\, \mathsf{T}.C$ by a fresh concept name $A_{(\geqslant n\, \mathsf{T}.C)}$. To put n different fresh individuals into the extension of C whenever the interpretation of $A_{(\geqslant n\, \mathsf{T}.C)}$ is not empty, we use a CIA

$$A_{(\geqslant n\, \mathsf{T}.C)} \sqsubseteq \prod_{1 \leq i \leq n} \exists\mathsf{T}.(C \sqcap \{a_i\} \sqcap \prod_{1 \leq j < n} \neg\{a_j\}).$$

$$
\begin{array}{rcl}
\neg(C_1 \sqcap C_2)^{\mathcal{I}} & = & (\neg C_1 \sqcup \neg C_2)^{\mathcal{I}} \\
\neg(C_1 \sqcup C_2)^{\mathcal{I}} & = & (\neg C_1 \sqcap \neg C_2)^{\mathcal{I}} \\
\neg(\forall R.C)^{\mathcal{I}} & = & (\exists R.\neg C)^{\mathcal{I}} \\
\neg(\exists R.C)^{\mathcal{I}} & = & (\forall R.\neg C)^{\mathcal{I}} \\
\neg(\leqslant n\, R.C)^{\mathcal{I}} & = & (\geqslant n+1\, R.\neg C)^{\mathcal{I}} \\
\neg(\geqslant n\, R.C)^{\mathcal{I}} & = & (\leqslant n-1\, R.\neg C)^{\mathcal{I}}
\end{array}
\qquad
\begin{array}{rcl}
(S \setminus (S_1 \cap S_2))^{\mathcal{I}} & = & ((S \setminus S_1) \cup (S \setminus S_2))^{\mathcal{I}} \\
(S \setminus (S_1 \cup S_2))^{\mathcal{I}} & = & ((S \setminus S_1) \cap (S \setminus S_2))^{\mathcal{I}} \\
(S \setminus (S_1 \setminus S_2))^{\mathcal{I}} & = & ((S \cap S_1) \cup (S \setminus S_2))^{\mathcal{I}}
\end{array}
$$

Table 2.2: Translation of concepts and roles into Negation Normal Form (NNF)

Note that these number restrictions are not supported by $\mathcal{SRIQ}$ according to [HKS05], and they can not be added to it without modifying its expressive power. Allowing expressions of the form $\leqslant n\, \mathsf{T}.C$ amounts to imposing a cardinality restriction on the (interpretation of) the concept C. This is enough to simulate nominals [Tob00], making $\mathcal{SRIQ}$ as expressive as $\mathcal{SROIQ}$.

2.1.5 Negation Normal Form

It is sometimes convenient to consider expressions in *negation normal form (NNF)*, where negation has been pushed inside as much as possible. In most DLs we consider *concepts in NNF*, which means that negation can only occur before concepts that have no proper subconcepts. In the case of $\mathcal{ZOIQ}$, as we also have Boolean role operators, it makes sense to consider *expressions in NNF* that can be concepts or roles. In addition to the restrictions on negated subconcepts above, role difference can only occur with an atomic role on the right hand side.

Definition 2.1.37 (Negation Normal Form (NNF)) *Let $\mathcal{L}$ be any DL that is a sublogic of $\mathcal{SHIQ}$ or $\mathcal{SROIQ}$. Then we say that a concept C in $\mathcal{L}$ is in* negation normal form (NNF)*, if every negated subconcept of C is of the form $\neg A$, $\neg\{a\}$, or $\neg\exists S.\mathsf{Self}$, where $A \in \mathsf{N_C}$, $a \in \mathsf{N_I}$ and S is a role in $\mathcal{L}$.*

Analogously, for a $\mathcal{ZOIQ}$ concept or role E, we say that E is in NNF, if (i) every negated subconcept of E has the form $\neg A$, $\neg\{a\}$, or $\neg\exists S.\mathsf{Self}$, where $A \in \mathsf{N_C}$, $a \in \mathsf{N_I}$ and S is a simple role, and (ii) for every subrole of E of the form $S \setminus S'$, S is either p or p^- for some $p \in \mathsf{N_R}$.

It is well known that, in all the DLs we have introduced, concepts and roles can be efficiently (in polynomial time) rewritten into NNF using the equivalences in Table 2.2, which hold for every interpretation $\mathcal{I}$.

Proposition 2.1.38 *Let $\mathcal{L}$ be any of the DLs we have defined. For every concept or role E in $\mathcal{L}$, there is a concept or role $nnf(E)$ in NNF such that $E^{\mathcal{I}} = nnf(E)^{\mathcal{I}}$ for every interpretation $\mathcal{I}$.*

2.1.6 Reasoning in DLs

DLs, as knowledge representation formalisms, are expected to provide reasoning services, that is, to allow for solving some *inference* or *reasoning problems* that may be of interest to the users. There are quite a few such problems that are considered important and that have been studied for most DLs, and are usually called *standard reasoning problems* (as opposed to *non-standard* problems such as reasoning with queries, which will be mentioned later).

The most notable one is *knowledge base satisfiability*:

Definition 2.1.39 (knowledge base satisfiability) *A knowledge base $\mathcal{K}$ is* satisfiable *if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$, and* unsatisfiable *otherwise. The* knowledge base satisfiability problem *is to decide whether a given knowledge base is satisfiable or unsatisfiable.*

We recall a few other standard reasoning problems, which are, in essence, equivalent to (un)satisfiability via linear time reductions:

Concept satisfiability w.r.t. a knowledge base: Given a concept C and a knowledge base $\mathcal{K}$, decide whether C *is satisfiable w.r.t.* $\mathcal{K}$, that is, whether there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$ and $C^{\mathcal{I}} \neq \emptyset$.

This reasoning problem can be reduced to knowledge base satisfiability: C is satisfiable w.r.t. $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ iff $\langle \mathcal{A} \cup \{C(a)\}, \mathcal{T}, \mathcal{R} \rangle$ is satisfiable, where a is a fresh individual.

Concept subsumption w.r.t. a knowledge base: Given two concepts C and D and a knowledge base $\mathcal{K}$, decide whether C *is subsumed by* D *w.r.t.* $\mathcal{K}$, that is, whether $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$.

Concept subsumption reduces easily to concept *unsatisfiability*, and hence to knowledge base unsatisfiability: C is subsumed by D iff $\neg C \sqcup D$ is not satisfiable.

Instance checking w.r.t. a knowledge base: Given a concept C, an individual a, and a knowledge base $\mathcal{K}$, decide whether a *is an instance of* C *w.r.t.* $\mathcal{K}$, that is, whether $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$.

This reasoning problem can be reduced to knowledge base unsatisfiability: a is an instance of C w.r.t. $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ iff $\langle \mathcal{A} \cup \{\neg C(a)\}, \mathcal{T}, \mathcal{R} \rangle$ is unsatisfiable.

In this thesis we only consider knowledge base satisfiability, but by the reductions above all complexity results given for this problem extend for the other standard reasoning tasks.

2.2 Queries over Description Logic knowledge bases

The focus of this thesis is on *query answering* reasoning services, that is, using query languages whose expressive power goes beyond that of DL concept and role expressions, and that allow one to use variables in order to join pieces of information when finding query answers, thus overcoming one of the most significant drawbacks of DLs as languages for data management.

In this section, we introduce some of these query languages and define some problems that are relevant when reasoning about queries. We first define a general notion of a *query* and its semantics, and introduce some general notation that we use throughout the thesis. We introduce some of the problems that arise when reasoning with queries in DLs, and define *query entailment* as the central query reasoning problem considered in this work. We also show how other reasoning problems reduce to this one. Finally, we define some *query languages* as restricted families of queries explicitely addressed in this thesis.

2.2.1 Syntax and Semantics of Queries

First we introduce a general notion of a *query*, which is a positive existential formula built from *query atoms*. A query atom is built from a DL concept or a DL role and is syntactically similar to an ABox assertion, but it allows for variables. Since these variables may be shared by the different query atoms of a query in an arbitrary way, it is possible to combine pieces of information in a flexible way.

Definition 2.2.1 (query atom, (positive) query) *Let* $\mathsf{N_V}$ *denote a countably infinite set of* variables*, and let* $\mathcal{L}$ *be a DL. Then a* query atom in $\mathcal{L}$ *is an expression* $C(v)$ *(a* concept atom*) or* $R(v, v')$ *(a* role atom*), where each of the* arguments v *and* v' *is either a variable from* $\mathsf{N_V}$ *or an individual from* $\mathsf{N_I}$*,* C *is a concept in* $\mathcal{L}$*, and* R *is a role in* $\mathcal{L}$*.*

A (positive) $\mathcal{L}$ query (with answer variables $\vec{x}$) *is an expression* $\exists \vec{v}.\varphi(\vec{x}, \vec{v})$*, where* φ *is built using* $\wedge$ *and* $\vee$ *from query atoms in* $\mathcal{L}$ *and where only variables from* $\vec{v} \cup \vec{x}$ *occur. If* $\vec{x} = x_1, \ldots, x_n$*, then we may call* q *a query with* n *answer variables. If* $\vec{x} = \emptyset$*, that is, if all variables in* φ *are existentially quantified, then we call* q *a* Boolean *query.*

Example 2.2.2 *We consider the following queries. First, a* $\mathcal{ZOIQ}$ *query:*

$$q_1^{theo} = \exists v.\ \mathsf{hasParent}^* \circ \mathsf{hasParent}^{-*}(x_1, x_2) \wedge \mathsf{hasChild}(x_1, v) \wedge \mathsf{hasChild}(x_2, v) \wedge \mathsf{Male}(x_1) \wedge \mathsf{Female}(x_2) \wedge (\neg \mathsf{Deity}(x_1) \vee \neg \mathsf{Deity}(x_2))$$

Its answer variables are x_1 *and* x_2*. Informally, it asks for pairs of individuals who are relatives (i.e., related by the expression* $\mathsf{hasParent}^* \circ \mathsf{hasParent}^{-*}$*), who have a common child* v*, one is male and the other female, and at least one of them is not a deity.*

Another $\mathcal{ZOIQ}$ *query is the following:*

$$q_2^{theo} = \exists v_1, v_2.\ \mathsf{Hero}(x) \wedge \mathsf{hasMother}(x, v_1) \wedge \mathsf{hasParent}^*(v_1, v_2) \wedge \mathsf{Deity}(v_2)$$

It is not Boolean, as it has one answer variable x*. Informally, the answers to* q_1^{theo} *are the heroes (represented by* x*) that have a divine ancestor on the maternal side.*

We also give two examples of Boolean queries:

$$q_3^{theo} = \exists v_1, v_2, v_3, v_4.\ \mathsf{Hero}(v_1) \wedge \mathsf{hasMother}(v_1, v_2) \wedge \mathsf{hasParent}(v_2, v_3) \wedge \mathsf{Deity}(v_3) \wedge \mathsf{hasChild}(v_4, v_3) \wedge \mathsf{Primordial}(v_4)$$

$$q_4^{theo} = \exists v_1, v_2, v_3, v_4.\ \mathsf{hasDirectRelative}(v_1, v_2) \wedge \mathsf{hasDirectRelative}(v_2, v_3) \wedge \mathsf{hasDirectRelative}(v_3, v_4) \wedge \mathsf{hasDirectRelative}(v_4, v_1) \wedge \mathsf{Deity}(v_1) \wedge \mathsf{Hero}(v_2) \wedge \mathsf{Male}(v_4)$$

Informally, q_3^{theo} *asks whether there exist some hero that has a divine maternal grandparent that is a child of a primordial god. The other query* q_4^{theo} *asks whether there is a cycle of four direct relatives* v_1, v_2, v_3, v_4 *such that* v_1 *is male,* v_2 *is a deity, and* v_3 *is female.*

We will use the following notation:

Notation 2.2.3 *For a query* $q = \exists \vec{v}.\varphi(\vec{x}, \vec{v})$*, we let* $\mathsf{Atoms}(q)$ *denote the set of all atoms occurring in* q*, and let* $\mathsf{VI}(q)$ *denote the set individuals and variables that occur as arguments in the atoms of* q*. Similarly as in Notation 2.1.8, the set of concept names, role names, and individual names that occur in* q *are respectively denoted* $\mathsf{N_C}(q)$*,* $\mathsf{N_R}(q)$ *and* $\mathsf{N_I}(q)$*.*

The semantics of queries is given in terms of interpretations in the natural way.

Definition 2.2.4 (binding, query match) *A* binding for a query q in an interpretation $\mathcal{I}$ *is a total function* $\pi : \mathsf{VI}(q) \to \Delta^{\mathcal{I}}$ *such that* $\pi(a) = a^{\mathcal{I}}$ *for each individual* $a \in \mathsf{VI}(q)$*. We write* $\mathcal{I}, \pi \models C(v)$*, if* $\pi(v) \in C^{\mathcal{I}}$*, and* $\mathcal{I}, \pi \models R(v, v')$*, if* $(\pi(v), \pi(v')) \in R^{\mathcal{I}}$*.*

We say that a set $At \subseteq \mathsf{Atoms}(q)$ makes q true*, if the Boolean formula that results from replacing each* $\alpha \in At$ *with* ***true*** *and each* $\alpha \in \mathsf{Atoms}(q) \setminus At$ *with* ***false*** *evaluates to* ***true****. We call a binding* π *a* match for q in $\mathcal{I}$*, if the set* $\{\alpha \in \mathsf{Atoms}(q) \mid \mathcal{I}, \pi \models \alpha\}$ *makes* q *true.*

Let $\vec{x} = x_1, \ldots, x_n$ *be the answer variables of* q*, and let* $\vec{a} = a_1, \ldots, a_n$ *be a tuple of individuals. Then we write* $\mathcal{I}, \pi \models q(\vec{a})$ *if* π *is a match for* q *in* $\mathcal{I}$ *and* $\pi(x_i) = a_i$ *for each* $1 \leq i \leq n$*, and we write* $\mathcal{I} \models q(\vec{a})$ *if there is a match* π *such that* $\mathcal{I}, \pi \models q(\vec{a})$*.*

If q *is Boolean, i.e., if* $n = 0$*, then we write simply* $\mathcal{I}, \pi \models q$ *and* $\mathcal{I} \models q$*.*

2.2.2 Reasoning with Queries

We next consider some reasoning problems associated to queries over DL knowledge bases. First, for queries with answer variables, the basic decision problem is usually called *query answering*, and it consists in deciding whether a given tuple is an answer to the query in *all* the models of the knowledge base, sometimes referred to as a *certain answer*. Boolean queries (which have no answer variables) evaluate to true or to false in an interpretation. For them we consider the query entailment problem, where we want to decide whether the query evaluates to true in *all* the models of a knowledge base. Finally, we consider the *query containment* problem, where we want to decide whether the answers of one query are contained in the answers of another query.

Query Answering and Query Entailment

For non-Boolean queries, the basic reasoning task is query answering:

Definition 2.2.5 (Query Answering) *Given a query q with answer variables $\vec{x} = x_1, \ldots, x_n$ and a knowledge base $\mathcal{K}$, we say that $\vec{a} = a_1, \ldots, a_n$ is* an answer for q in $\mathcal{K}$*, in symbols $\mathcal{K} \models q(\vec{a})$, if $\mathcal{I} \models q(\vec{a})$ for every model $\mathcal{I}$ of $\mathcal{K}$.*

The query answer enumeration problem *for $\mathcal{K}$ and q consists in listing all the answers for q in $\mathcal{K}$. The associated decision problem, which we call simply* query answering*, consists in deciding, given a query q, a knowledge base $\mathcal{K}$, and a tuple $\vec{a}$, whether $\vec{a}$ is an answer for q in $\mathcal{K}$.*

For Boolean queries, whose only possible answer is the empty tuple, one usually talks about *query entailment*.

Definition 2.2.6 (Query Entailment) *Given a Boolean query q and a knowledge base $\mathcal{K}$, we say that $\mathcal{K}$* entails *q, in symbols $\mathcal{K} \models q$, if $\mathcal{I} \models q$ for every model $\mathcal{I}$ of $\mathcal{K}$.*

The query entailment *problem consists in deciding, given a knowledge base $\mathcal{K}$ and a Boolean query q, whether $\mathcal{K} \models q$.*

Example 2.2.7 *Recall the interpretation $\mathcal{I}_1$ from Example 2.1.22 (see Figure 2.3), and the query q_1^{theo} from the previous example. The binding π with $\pi(x_1) = \text{zeus}^{\mathcal{I}} = 1$, $\pi(x_2) = \text{alcmene}^{\mathcal{I}} = 3$ and $\pi(v) = \text{heracles}^{\mathcal{I}} = 2$ is a match for q_1^{theo} in $\mathcal{I}_1$. In fact, we can set $\pi(v_1) = \text{zeus}^{\mathcal{I}}$, $\pi(v_2) = \text{alcmene}^{\mathcal{I}}$ and $\pi(v_3) = \text{heracles}^{\mathcal{I}}$ to obtain a match in* any *model of $\mathcal{K}_2^{theo}$. Hence the pair* $(\text{zeus}, \text{alcmene})$ *is an answer of q_1^{theo}, in symbols $\mathcal{K}_2^{theo} \models q_1^{theo}(\text{zeus}, \text{alcmene})$.*

It is well known that the query answering problem for an arbitrary query reduces linearly to the entailment problem of a Boolean query.

Proposition 2.2.8 *Let $\exists \vec{v}.\varphi(\vec{x}, \vec{v})$ be a query with answer variables $\vec{x} = x_1, \ldots, x_n$, and let $\vec{a} = a_1, \ldots, a_n$ be a tuple of individuals of the same arity. Then, for each knowledge base $\mathcal{K}$, $\mathcal{K} \models q(\vec{a})$ iff $\mathcal{K} \models q^{\vec{a}}$, where $q^{\vec{a}}$ is the Boolean query $\exists \vec{v}.\varphi'(\vec{v})$ obtained by replacing in φ each occurrence of x_i by a_i, for each $x_i \in \vec{x}$.*

By this proposition, it suffices to consider the entailment of Boolean queries in order to characterize the computational complexity of query answering, as we do in most of this thesis.

Without loss of generality, we may even consider the query entailment problem under some simplifying assumptions. For example, it is sometimes convenient to assume that queries contain only variables and no constants as arguments, i.e., $\mathsf{VI}(q) \subseteq \mathsf{N_V}$. For all DLs $\mathcal{L}$ containing nominals, queries in $\mathcal{L}$ can easily simulate constants. If queries are in a DL $\mathcal{L}$ not supporting nominals, then constants can be simulated using distinguished predicate names and adding assertions to the ABox.

Proposition 2.2.9 *For every Boolean query q and knowledge base $\mathcal{K}$, it is possible to obtain a Boolean query q' and a knowledge base $\mathcal{K}'$ such that $\mathcal{K} \models q$ iff $\mathcal{K}' \models q'$ and $\mathsf{VI}(q') \subseteq \mathsf{N_V}$.*

Proof. Let $q = \exists \vec{v}.\varphi(\vec{v})$ with $\vec{v} = v_1, \ldots, v_n$, and let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. For each $a \in \mathsf{VI}(q) \cap \mathsf{N_I}$, let v_a be a fresh variable, and let A_a be a fresh concept name. We define $\mathcal{K}' = \langle \mathcal{A}', \mathcal{T} \rangle$ with

$$\mathcal{A}' = \mathcal{A} \cup \bigcup_{a \in \mathsf{VI}(q) \cap \mathsf{N_I}} \{A_a(a)\}$$

and define

$$q' = \exists \vec{v'}.\varphi'(\vec{v'}) \wedge \bigwedge_{a \in \mathsf{VI}(q) \cap \mathsf{N_I}} A_a(v_a)$$

where φ' is obtained from φ by replacing each occurrence of an individual a as an argument by v_a, and $\vec{v'} = \vec{v} \cup \vec{v_a}$ for some $\vec{v_a}$ containing all the variables v_a. Then $\mathcal{K} \models q$ iff $\mathcal{K}' \models q'$ as desired.

We note that if the query is in a DL supporting nominals, then we do not need the auxiliary concepts A_a. Instead of $C_a(v_a)$, we can use atoms of the form $\{a\}(v_a)$ in the query q'. Then we do not need to modify the knowledge base, and $\mathcal{K} \models q$ iff $\mathcal{K} \models q'$. □

For deciding query entailment $\mathcal{K} \models q$, we may also assume that all vocabulary symbols (concept names, role names and individuals) occurring in the query occur in $\mathcal{K}$ as well.

Proposition 2.2.10 *Let $\mathcal{L}$ be any DL containing $\mathcal{ALC}$. For every Boolean query q and every $\mathcal{L}$ knowledge base $\mathcal{K}$, it is possible to obtain an $\mathcal{L}$ knowledge base $\mathcal{K}'$ such that $\mathsf{N_C}(q) \subseteq \mathsf{N_C}(\mathcal{K})$, $\mathsf{N_R}(q) \subseteq \mathsf{N_R}(\mathcal{K})$, $\mathsf{N_I}(q) \subseteq \mathsf{N_I}(\mathcal{K})$, and $\mathcal{K} \models q$ iff $\mathcal{K}' \models q$.*

Proof. Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. To obtain $\mathcal{K}'$, we simply add to $\mathcal{T}$ a CIA $A \sqsubseteq \top$ for each $A \in \mathsf{N_C}(q) \setminus \mathsf{N_C}(\mathcal{K})$, and a CIA $\exists p.\top \sqsubseteq \top$ for each $p \in \mathsf{N_R}(q) \setminus \mathsf{N_R}(\mathcal{K})$, and we add to $\mathcal{A}$ an assertion $\top(a)$ for each $a \in \mathsf{N_I}(q) \setminus \mathsf{N_I}(\mathcal{K})$. Then $\mathcal{K}$ and $\mathcal{K}'$ have the same models and entail the same queries. □

Reducing knowledge base satisfiability to query entailment is trivial:

Proposition 2.2.11 *Let $\mathcal{L}$ be a DL, and let $\mathcal{K}$ be an $\mathcal{L}$ knowledge base. Then $\mathcal{K}$ is satisfiable iff $\mathcal{K} \not\models \exists v.\bot(v)$.*

Query Containment

Another important problem when reasoning with queries is *query containment*, which we can consider for both Boolean and non-Boolean queries. We receive as an input two queries q_1 and q_2 with tuples of answer variables of the same arity and we want to decide whether always all answers for q_1 are answers for q_2. In particular, if the arity of the answer tuples is 0, that is, the queries are Boolean, then this amounts to deciding whether q_2 evaluates to true in every model of $\mathcal{K}$ where q_1 evaluates to true.

The importance of query containment is widely acknowledged in the database field. It plays a fundamental role in query optimization: two mutually containing queries are equivalent, and replacing queries by 'simpler' but equivalent ones is the main goal of query optimization [AHV95]. As a basic query reasoning problem, it also lies at the core of other database problems such as consistency checking, and data exchange and integration, and has thus received considerable attention in the literature, cf. [MLF00, LR98b, CDGL08, CDGV03].

Definition 2.2.12 (Query containment) *Given a knowledge base $\mathcal{K}$ and two queries q_1 with answer variables $x_1, \ldots, x_n$ and q_2 with answer variables $x'_1, \ldots, x'_n$, the* query containment problem (w.r.t. a knowledge base) *is to decide whether $\mathcal{K} \models q_1(\vec{a})$ implies $\mathcal{K} \models q_2(\vec{a})$ for each tuple of individuals $\vec{a} = a_1, \ldots, a_n$, in symbols $\mathcal{K} \models q_1 \subseteq q_2$. In particular, if $n = 0$ (i.e., q_1 and q_2 are Boolean queries), $\mathcal{K} \models q_1 \subseteq q_2$ if $\mathcal{K} \models q_1$ implies $\mathcal{K} \models q_2$.*

Clearly, query containment is not easier than query entailment:

Proposition 2.2.13 *Let $\mathcal{K}$ be a knowledge base and let q be a Boolean query. Then $\mathcal{K} \models q$ iff $\mathcal{K} \models \exists v.\top(v) \subseteq q$.*

It is well known in the database field that query containment can usually be reduced to query entailment. In particular, $\mathcal{K} \models q_1 \subseteq q_2$ can be decided by stating the query q_1 as part of the ABox (i.e., 'freezing' it as ABox assertion(s)), and checking whether the resulting knowledge base entails q_2.

Proposition 2.2.14 *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ be an $\mathcal{L}$ knowledge base and let q_1 and q_2 be two queries. If $\mathcal{A}_1$ is an ABox in $\mathcal{L}$ such that, for every $\mathcal{I}$,*

$$\mathcal{I} \models \mathcal{A}_1 \text{ iff } \mathcal{I} \models q_1$$

then $\mathcal{K} \models q_1 \subseteq q_2$ iff $\langle \mathcal{A} \cup \mathcal{A}_1, \mathcal{T} \rangle \models q_2$.

In particular, the latter is the case if one of the following holds:

1. *q_1 contains only conjunction, and $\mathcal{A}_1$ is a ABox in $\mathcal{L}$ that contains*
   - *an assertion $C(a_v)$ for each concept atom $C(v) \in \mathsf{Atoms}(q_1)$, and*
   - *an assertion $R(a_v, a_{v'})$ for each role atom $R(v, v') \in \mathsf{Atoms}(q_1)$; or*
2. *$q_1 = \exists \vec{v}.\varphi(\vec{x}, \vec{v})$ and $\mathcal{A}_1 = \{C_{q_1}(a)\}$ for a fresh individual a and an $\mathcal{L}$ concept C_{q_1} obtained from φ by replacing:*
   - *$\wedge$ by $\sqcap$ and $\vee$ by $\sqcup$,*
   - *each concept atom $C(v) \in \mathsf{Atoms}(q_1)$ by $\neg\{a_v\} \sqcup C$, and*
   - *each role atom $R(v, v') \in \mathsf{Atoms}(q_1)$ by $\neg\{a_v\} \sqcup \exists R.\{a_{v'}\}$;*

where a_v is a fresh individual if v is a variable, and $a_v = v$ if v is an individual.

Example 2.2.15 *Recall the query q_2^{theo} from the Example 2.2.2, and consider*

$$q'^{theo}_2 = \exists v_1, v_2.\ \mathsf{Hero}(x) \wedge \mathsf{hasMother}(x, v_1) \wedge \mathsf{hasAncestor}(v_1, v_2) \wedge \mathsf{Deity}(v_2)$$

The two queries are very similar, except that in q'^{theo}_2 v_2 must be matched to an ancestor of the mother of the binding of x, rather than to a maternal ancestor of x. In every interpretation, a match for q'^{theo}_2 is also a match for q_2^{theo}. This shows that $\mathcal{K} \models q_2^{theo} \subseteq q'^{theo}_2$ for each $\mathcal{K}$. The converse does not hold. For example, in a model where the only divine maternal ancestor of any hero is the hero's mother herself, q_2^{theo} may have some answer that is not an answer to q'^{theo}_2.

2.2.3 Query Languages

In this thesis, we usually consider queries that are in some restricted families. We will of course talk about *conjunctive queries* and their unions, which are the most popular queries in the DL literature. However, we also push the current frontiers by considering more expressive queries, and in particular, considering the rich language of *positive two-way regular path queries*.

Conjunctive Queries and Unions of Conjunctive Queries

We start by defining two query languages that have received special attention in the literature: *Conjunctive Queries (CQs)* and *Unions of Conjunctive Queries (UCQs)* [AHV95]. CQs are the formal counterpart of the most widely used fragment of SQL (or relational algebra) queries, namely plain select-project-join queries, and UCQs are simply the unions of such queries. These popular query languages are widely studied in databases [CM77, AHV95], and due to a good trade-off between expressive power and nice computational properties, they have been adopted as core query languages in several contexts, such as query optimization [GU92, ACPS96], data integration [Len02, Ull00, Noy04], and ontology-based data access [PLC+08]. Most of the work concerning query answering in DLs refers in fact to CQs and UCQs.

Definition 2.2.16 (CQ, UCQ) *A CQ is a query not containing $\vee$ and where the role constructors $\circ$, * and $id()$ do not occur. A UCQ is a disjunction of CQs, that is a query $q = \exists \vec{v}.\big(\varphi_1(\vec{x},\vec{v}) \vee \cdots \vee \varphi_n(\vec{x},\vec{v})\big)$ where each $\varphi_i(\vec{x},\vec{v})$ is a conjunction of atoms without $\circ$, * and $id()$.*

Note that (non-simple) regular role expressions and role chains are not allowed in UCQs, even in the logics of the $\mathcal{Z}$ and $\mathcal{SR}$ family. Boolean roles, on the other hand, may be expressed.

When we talk about CQs or UCQs *over* a DL $\mathcal{L}$, we mean CQs or UCQs with atoms in $\mathcal{L}$ over knowledge bases in $\mathcal{L}$.

We sometimes restrict concepts and roles in CQs to *atomic* ones. That is, concepts may only be concept names or nominals, and roles may only be role names and inverses.

Definition 2.2.17 *We define $\mathsf{N_{CI}} = \mathsf{N_C} \cup \{\{a\} \mid a \in \mathsf{N_I}\}$, and recall that $\overline{\mathsf{N_R}} = \mathsf{N_R} \cup \{p^- \mid p \in \mathsf{N_R}\}$. A CQ q is* extensionally reduced*, if each concept atom in $\mathsf{Atoms}(q)$ is of the form $B(v)$ with $B \in \mathsf{N_{CI}}$, and each role atom of the form $P(v,v')$ with $P \in \overline{\mathsf{N_R}}$.*

The restrictions to extensionally reduced CQs is done only for convenience. In the standard setting of $\mathcal{L}$ queries over $\mathcal{L}$ knowledge bases, allowing complex concepts and roles does not make queries more expressive. Indeed, we can simply rewrite q and $\mathcal{K}$ in such a way that we obtain an extensionally reduced CQ and query entailment is preserved.

Proposition 2.2.18 *Let q be a query in $\mathcal{L}$ and let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R}\rangle$ be an $\mathcal{L}'$ knowledge base such that every $\mathcal{L}$ concept in q is also an $\mathcal{L}'$ concept, and every $\mathcal{L}$ role is also an $\mathcal{L}'$ role. Then $\mathcal{K} \models q$ iff $\langle \mathcal{A}, \mathcal{T}', \mathcal{R}'\rangle \models q'$, where q', $\mathcal{T}'$ and $\mathcal{R}'$ are obtained*

- *by replacing each concept atom $C(v) \in \mathsf{Atoms}(q)$ such that $C \notin \mathsf{N_{CI}}$ with $A_C(v)$ for a fresh concept name A_C, and by adding a CIA $C \sqsubseteq A_C$ to $\mathcal{T}$, and*
- *by replacing each role atom $S(v,v') \in \mathsf{Atoms}(q)$ such that $S \notin \overline{\mathsf{N_R}}$ with $p_S(v,v')$ for a fresh concept name p_S, and by adding a BRIA $S \sqsubseteq p_S$ to $\mathcal{R}$.*

Example 2.2.19 *The queries q_3^{theo} and q_4^{theo} (see Example 2.2.2) are CQs in every DL $\mathcal{L}$, and they are extensionally reduced. On the other hand, q_2^{theo} is not a CQ or UCQ, because it contains a regular role expression $\mathsf{hasParent}^*$. However, in the DLs of the $\mathcal{SH}$ and $\mathcal{SR}$ families we can use the following UCQ to obtain the same answers:*

$$q''^{theo}_2 = \exists v_1, v_2.\ \big(\mathsf{Hero}(x) \wedge \mathsf{hasMother}(x,v_1) \wedge \mathsf{hasAncestor}(v_1,v_2) \wedge \mathsf{Deity}(v_2)\big) \vee \big(\mathsf{Hero}(x) \wedge \mathsf{hasMother}(x,v_1) \wedge \mathsf{Deity}(v_1)\big)$$

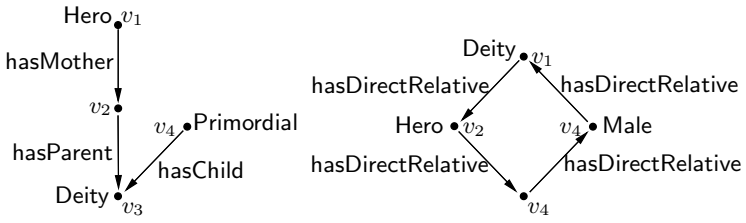

Figure 2.7: The query graphs of q_3^{theo} and q_4^{theo}

Since a CQ $q = \exists\vec{v}.\varphi(\vec{x},\vec{v})$ contains only conjunctions, the query is uniquely determined by its answer variables and the set of its atoms, and we do not need to store the formula φ. Further, if the query is Boolean, we can simply use the set $\mathsf{Atoms}(q)$ to represent it. Similarly, a Boolean UCQ can be represented as a collection of sets of atoms.

Notation 2.2.20 *For a Boolean UCQ $Q = \exists\vec{v}.\varphi_1 \vee \cdots \vee \varphi_n$, we may write*

$$Q = \bigcup_{1\leq i\leq n} \{\mathsf{Atoms}(\exists\vec{v}.\varphi_i)\}$$

and, abusing notation, use $q_i \in Q$ in the place of $\mathsf{Atoms}(q_i) \in Q$ for each $q_i = \exists\vec{v}.\varphi_i$, $1 \leq i \leq n$.

Viewing a Boolean UCQ as a set of atoms suggests a very natural notion of a *subquery*, which is sometimes used throughout the thesis.

Definition 2.2.21 (subquery, query restriction) *A query CQ q' is a* subquery *of a CQ q, if $\mathsf{Atoms}(q') \subseteq \mathsf{Atoms}(q)$. For Boolean CQ q and a set $V \subseteq \mathsf{Vl}(q)$, we denote by $q|_V$ the CQ obtained by restricting to atoms of q to those whose arguments are contained in V, i.e., the query such that $\mathsf{Atoms}(q|_V) = \{\alpha \mid \alpha \in \mathsf{Atoms}(q)$ and $\mathsf{Vl}(\alpha) \subseteq V\}$, where $\mathsf{Vl}(\alpha)$ denotes the arguments of α.*

Query Graph of a CQ, Connected Queries The set $\mathsf{Atoms}(q)$ has a natural graph representation that we call *query graph*.

Definition 2.2.22 (Query Graph) *The* query graph *of a CQ q is the node-arc labeled graph $\mathsf{G}(q) = \langle V, E, \lambda\rangle$ with nodes $V = \mathsf{Vl}(q)$ and edges $E = \{(v,v') \mid R(v,v') \in q$ for some role $R\}$, where each node v is labeled with the set of concepts $\lambda(v) = \{A \mid A(v) \in q\}$ and each arc (v,v') is labeled with the set of roles $\lambda(v,v') = \{R \mid R(v,v') \in q\}$.*

Example 2.2.23 *The query graphs of q_3^{theo} and q_4^{theo} (see Example 2.2.2) are depicted in Figure 2.7.*

Note that finding a match for q in an interpretation $\mathcal{I}$ reduces to finding a homomorphic embedding of $\mathsf{G}(q)$ into $\mathcal{I}$, as the one depicted in Figure 2.8.

Definition 2.2.24 (connected (U)CQ, maximal connected queries) *We say that a CQ q is* connected *if the graph $\mathsf{G}(q)$. We call a Boolean UCQ Q* connected *if each $q \in Q$ is connected.*

Let q be a CQ. A maximal connected subquery *of q is a $q|_V$ for some $V \subseteq \mathsf{Vl}(q)$, such that $q|_V$ is connected and $q|_{V'}$ is not connected for each $V \subset V' \subseteq \mathsf{Vl}(q)$.*

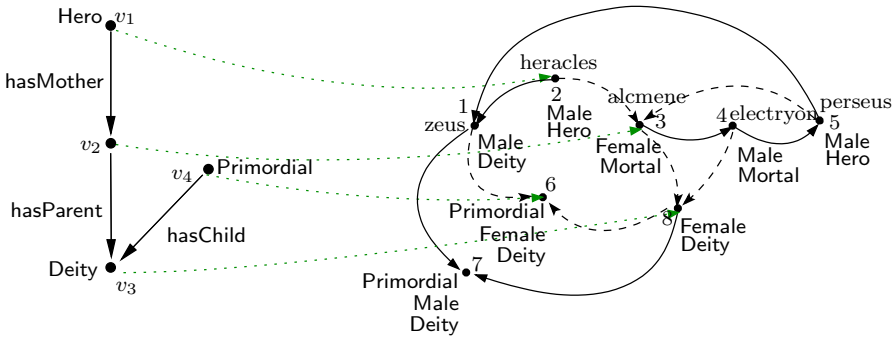

Figure 2.8: An example of a query match for q_3^{theo} in $\mathcal{I}_1$

It is not hard to see that, for disconnected subqueries, we can separately verify the entailment of each maximal connected subquery. Hence, when deciding the entailment of a CQ q, we may assume without loss of generality that q is connected.

Proposition 2.2.25 *Let q be a CQ and $\mathcal{K}$ a knowledge base $\mathcal{K}$ in some DL $\mathcal{L}$. Then $\mathcal{K} \models q$ iff $\mathcal{K} \models q'$ for each maximal connected subquery q' of q.*

Positive two-way Regular Path Queries

Although CQs in $\mathcal{L}$ over knowledge bases in $\mathcal{L}$ are the standard setting, it is sometimes interesting to allow in the query constructors that are no present in the knowledge base. For example, *regular path queries* (RPQs) [Bun97, ABS00, CDGLV99] allow one to ask for pairs of objects that are connected by a path conforming to a regular expression. Due to their ability to express complex navigations in graphs, RPQs are the fundamental mechanism for querying semi-structured data. RPQs are particularly useful when *inverse roles* are allowed to occur in the regular expression, since they can express complex conditions that require to navigate the data, without being constrained by the direction initially chosen by the designer to represent relations between data items.

We finally introduce Positive two-way Regular Path Queries (P2RPQs), the most expressive query language studied in this thesis. They allow to use regular expressions formed over role names and their inverses to query a knowledge base in a given DL $\mathcal{L}$, even if $\mathcal{L}$ itself does not allow for regular expressions and inverses, and they allow for testing the objects encountered during navigation for membership in $\mathcal{L}$ concepts. P2RPQs subsume simultaneously regular path queries and unions of conjunctive queries, and are also a natural generalization of several extensions of RPQs that have been studied by different authors, e.g. [CDGLV03, GT03, CDGLV02, DT01, CDGLV00, AV99, FLS98]. They are, to our knowledge, the most expressive query language considered so far to query DL knowledge bases [CEO09b, CEO07].

Definition 2.2.26 (P2RPQs) *A* positive two-way regular path query *(P2RPQ) over a DL $\mathcal{L}$ is a query $\exists \vec{v}.\varphi(\vec{x}, \vec{v})$ such that each atom $\alpha \in \mathsf{Atoms}(q)$ is either a concept atom $C(v)$ for an $\mathcal{L}$ concept C, or a role atom $R(v, v')$ for a regular role expression R that obeys the following grammar, where C is a concept in $\mathcal{L}$ and S is a role in $\mathcal{L}$:*

$$R, R_1, R_2 ::= S \mid R_1 \cup R_2 \mid R_1 \circ R_2 \mid R^* \mid id(C)$$

Example 2.2.27 *Since the query q_1^{theo} contains only concept names and regular role expressions, we can see it as a P2RPQ in any DL $\mathcal{L}$.*

Finally, we note that one can assume without loss of generality that only role atoms $R(v, v')$ occur in a P2RPQ, as each concept atom $C(v)$ can be equivalently expressed as $id(C)(v, v)$.

2.3 Measuring the Complexity of Reasoning

One of the goals of this thesis is characterize the *computational complexity* of answering different kinds of queries over different DLs, that is, to classify query answering problems into *complexity classes* according to their computational difficulty.

2.3.1 Complexity Classes

Complexity classes classify problems in terms of resources (like *time* or *space*) needed to solve them using different sorts of computation devices, like *Turing machines*. We refer the reader to [Pap94] for the basic notions of Complexity Theory.

Each decision problem can viewed as a language $L \subseteq \Sigma^*$ together with the task of deciding whether a word $w \in \Sigma^*$ (encoding an instance of the problem) belong to L or not. One of the most important complexity classes is PTime, which is the set of all problems (i.e. languages) that can be decided by a *deterministic* Turing machine (DTM) in time polynomial in the length of the input word. More precisely, a language L is in PTime iff there exists DTM M and a polynomial $p(n)$ such that for any word w, the machine M decides $w \in L$ within $p(|w|)$ steps. Another notable class is NP comprising the problems that can be decided in polynomial time by a *nondeterministic* Turing machine (NTM). Dually, coNP consists of languages L whose complement $\overline{L}$ is in NP. For instance, satisfiability of Boolean formulas is in NP, while unsatisfiability of such formulas is in coNP. It is widely believed that NP $\neq$ PTime and NP $\neq$ coNP, but these statements have not been proved yet. The class ExpTime (resp., NExpTime) consists of problems that can be solved by a DTM (resp., NTM) in single exponential time in the size of the input, i.e., the computation length can be bounded by $2^{p(n)}$, where $p(n)$ is a polynomial; co-NExpTime consists of the languages L whose complement $\overline{L}$ is in NExpTime. By allowing *double* or *triple* exponential time computations in DTMs, we obtain classes 2ExpTime and 3ExpTime. This can be continued to arbitrary towers of exponentials, obtaining a hierarchy of classes nExpTime, $n > 0$, and this can also be carried over to the case of NTMs.

The classes PSpace and NPSpace (resp., ExpSpace and NExpSpace) are defined similarly to PTime and NP (resp., ExpTime and NExpTime), but instead of putting a bound on time, we apply the bound on the space the computations use. In this setting, however, PSpace = NPSpace and ExpSpace = NExpSpace, which is due to the Savitch's Theorem [Sav70]. We recall some of the inclusions between the above mentioned complexity classes:

$$\text{PTime} \subseteq \text{NP} \subseteq \text{PSpace} = \text{NPSpace} \subseteq \text{ExpTime},$$

$$\text{ExpTime} \subseteq \text{NExpTime} \subseteq \text{ExpSpace} \subseteq \text{2ExpTime}.$$

A problem (i.e. some language L) in a complexity class $\mathcal{C}$ is *complete* for $\mathcal{C}$, if any problem in $\mathcal{C}$ can be *reduced* to it. Intuitively, this means that L is among the hardest problems in $\mathcal{C}$. More formally, a reduction from a language L to a language L' is a mapping f from words to words such that, for any word w, $w \in L$ iff $f(w) \in L'$. For our purposes, we consider *polynomial time reductions*, which means that f must be computable in polynomial time in the size of the input word. We say a language L is $\mathcal{C}$*-hard*, if any language $L' \in \mathcal{C}$ can be reduced to L. If $L \in \mathcal{C}$ and L is $\mathcal{C}$-hard, then L is $\mathcal{C}$*-complete*.

2.3.2 Combined and Data Complexity

The computational complexity of a problem is always measured in terms of the *size* of the problem instance. However, when reasoning with queries or in the presence of extensional data, the definition of the input that is relevant for evaluating the complexity may differ from one scenario to another. In the database setting, when queries in some query language are to be answered over a family of databases, Vardi identified two main settings where the complexity may be measured in different ways [Var82]:

- If one is interested in the complexity of evaluating any given query in the query language over any arbitrary database, then both components are part of the input, and complexity should be measured in the terms of the combined size of the query and the database. This is known as the *combined complexity* of query answering.
- If one is interested in the complexity of evaluating some fixed query over any input database, then only the size of the data determines the size of the input, and complexity should be measured in the terms of the size of the database only. This is known as the *data complexity* of query answering.

In the setting of querying DL knowledge bases, these are also accepted as significant measures of complexity. Combined complexity is considered the classical way to measure complexity, and it takes as an input a full knowledge base and a query. Data complexity considers only the extensional data in the ABox as an input, and both the intensional axioms and the query are assumed to be fixed. Data complexity becomes relevant when we see a knowledge base as a data repository, because then the extensional data can become very large, and it is usually much larger than the intensional axioms expressing constraints on the data. Therefore, the contribution of the extensional assertions to the complexity of inference should be singled out, and one must pay attention to optimizing inference techniques with respect to data size, as opposed to the overall size of the knowledge base.

We use the following notation throughout the thesis:

Notation 2.3.1 *We use $|X|$ to denote the cardinality of a set X, and $||X||$ to denote the length of some string encoding X.*

The measures of complexity that we consider are the following:

Definition 2.3.2 (Combined complexity of query entailment) *The* (combined) complexity *of query entailment is the complexity (in terms of complexity classes) of deciding, given a knowledge base $\mathcal{K}$ and a query q, whether $K \models q$.*

For *data complexity*, we assume that $\mathcal{T}$, $\mathcal{R}$ and q are fixed, and measure the complexity of deciding $\langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle \models q$ in terms of $||\mathcal{A}||$. In order for this characterization to be meaningful, we must ensure that all the intensional information is indeed given in $\mathcal{T}$ and $\mathcal{R}$, and that $\mathcal{A}$ only contains simple assertions over the vocabulary employed in the intensional component.

Definition 2.3.3 (Data complexity of query entailment) *The* data complexity *(in terms of complexity classes) of query entailment is the complexity of deciding $\langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle \models q$, where $\mathcal{T}$, $\mathcal{R}$ and q are assumed to be fixed, and the input $\mathcal{A}$ is an* extensionally reduced ABox *where all assertions are of the form $A(a)$ or $p(a, b)$ for A a concept name and p a role name, and where $\mathsf{N_C}(\mathcal{A}) \cup \mathsf{N_R}(\mathcal{A}) \subseteq \mathsf{N_C}(\mathcal{T}) \cup \mathsf{N_R}(\mathcal{T}) \cup \mathsf{N_R}(\mathcal{R})$, i.e., and all concepts and roles that occur in $\mathcal{A}$ occur also in $\mathcal{T}$ or $\mathcal{R}$.*

2.4 Trees and Forests

Many of techniques and results in this thesis are concerned with *trees* and *forests* of different kinds. Hence, before we start with the core material of the thesis, we give some general definitions and notation that will be used throughout all chapters.

Definition 2.4.1 (Forests, Trees) *We use $\mathbb{N}^*$ to denote the set of all words over the natural numbers $\mathbb{N}$. For a word w, $|w|$ denotes the length of w, i.e., the number of its symbols, and $w{\cdot}w'$ denotes the concatenation of w and w'. The empty word is denoted ε; note that $|\varepsilon| = 0$. By convention, $w{\cdot}\varepsilon = w$, and $(w{\cdot}c){\cdot}{-}1 = w$.*

An (infinite) tree (with root $c \in \mathbb{N}^*$) *is a set $T = \{c{\cdot}w \mid w \in N\}$ where $N \subseteq \mathbb{N}^*$ is prefix-closed, that is, if $w{\cdot}i \in N$ then $w \in N$. We call a tree* proper *if its root is the empty word ε. An* (infinite) forest *is a set $F \subseteq \mathbb{N}^+$ such that $w{\cdot}i \in F$ and $w \in \mathbb{N}^+$ imply $w \in F$. That is, a forest is a union of trees with roots in $\mathbb{N}$. The roots of a forest are $\mathsf{roots}(F) = F \cap \mathbb{N}$. Note that each forest with $\mathsf{roots}(F) = \{c\}$ for some $c \in \mathbb{N}$ is also a tree with root c.*

The elements of a tree or forest F are called nodes. *For each $w \in F$, the nodes in $\mathsf{succ}(w) = \{w{\cdot}c \in F \mid c \in \mathbb{N}\}$ are called* successors *of w, and w is their* predecessor. *The* ancestor *relation is the transitive closure of predecessor. The* branching degree *$d(w)$ of a node w is the number of its successors, and F has* branching degree bounded by *b if $d(w) \leq b$ for each node w of F. For $k \geq 0$, we call F a k-ary if $\mathsf{succ}(w) = \{w \cdot 1, \ldots, w \cdot k\}$ for each node $w \in F$.*

For a $w \in \mathbb{N}$, the subtree of F rooted at w *is the set T_w of nodes of the form $w \cdot w' \in F$. An* infinite path *π of F is a prefix-closed set $\pi \subseteq T$ where for every $i \geq 0$ there exists a unique node $w \in P$ with $|w| = i$.*

Given a set of symbols Σ (called alphabet*) a* Σ-labeled forest (or tree) *is a pair $\mathbf{F} = \langle F, L\rangle$, where F is a forest (or tree) and $V : F \to \Sigma$ is a* labeling function *that assigns a* label *$L(w)$ to each node $w \in F$. If the alphabet Σ is clear from the context, we may omit it and simply talk about labeled forests and trees.*

Chapter 3

Reasoning with Automata for the $\mathcal{ZOIQ}$ Family

In this chapter, we aim at pushing the ExpTime decidability frontier for reasoning in expressive description logics. The DL $\mathcal{ZOIQ}$, introduced in Section 2.1.3, extends $\mathcal{ALC}$ with nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), qualified number restrictions ($\mathcal{Q}$), regular expressions over roles (reg), safe Boolean role expressions (b) and inclusion axioms, and concepts of the form $\exists S.\mathsf{Self}$ (Self). In this chapter we study its sublogics $\mathcal{ZIQ}$, $\mathcal{ZOQ}$ and $\mathcal{ZOI}$, which are obtained by respectively disallowing nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), and qualified number restrictions ($\mathcal{Q}$). These are highly expressive DLs that, to our knowledge, are not subsumed by any other DLs for which reasoning algorithms and tight complexity upper bounds are known. They are closely related to other program and description logics, but they support additional features that allow us to easily simulate popular DLs, like the ones in the $\mathcal{SR}$ and $\mathcal{SH}$ families underlying the OWL standards, as we will show later.

Reasoning in $\mathcal{ZOIQ}$ is at least NExpTime-hard [Tob00]. By showing that the KB satisfiability problem of $\mathcal{ZIQ}$, $\mathcal{ZOQ}$ and $\mathcal{ZOI}$ can be solved in ExpTime, we identify three maximal and mutually incomparable DLs that are decidable in ExpTime. Further, the algorithm for these logics that we develop in this chapter is the basis for the query answering algorithm in Chapter 4, and for the results for the $\mathcal{SR}$ family that we give in Chapter 5.

We obtain our results by exploiting techniques based on *automata on infinite trees* [Tho90], and in particular on *alternating automata*. They are an elegant tool for reasoning in temporal and program logics [EJ91], and have been widely exploited for solving the satisfiability problem of many variants of PDL, the μ-calculus, and similar logics, proving useful for deriving tight complexity upper bounds [Var85, VW86, Var98, KSV02, BLMV08].

Specifically, we exploit the recently introduced *fully enriched automata (FEA)* [BLMV08], a powerful extension of other well known automata models well suited for the rich set of concept and role constructors supported in the $\mathcal{Z}$ family. We present a reduction of the existence of certain *canonical models* of a given KB to the emptiness test of a FEA. Relying on the upper bounds for the latter problem given in [BLMV08], we obtain a decision procedure for KB satisfiability in all sublogics of $\mathcal{ZOIQ}$ that enjoy the *canonical model property*, and in particular for $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$. Our automata procedure is the first one, to our knowledge, that simultaneously handles $\mathcal{Q}$, $\mathcal{I}$, and $\mathcal{O}$; it additionally considers Boolean role expressions and Self concepts. Notably, the use of FEAs allow us to obtain optimal complexity bounds even when

the numbers are encoded in binary.

The chapter is organized as follows. Section 3.1 shows that reasoning over $\mathcal{ZOIQ}$ KBs can be effectively reduced to reasoning over (restricted) $\mathcal{ZOIQ}$ concepts. The crucial *canonical model property* of these concepts is shown in Section 3.2, and the automata-based technique for deciding the existence of canonical models is presented in 3.3. Complexity results are given in 3.4, and related work is discussed in Section 3.5.

3.1 From Knowledge Bases to Concepts

In this section we reduce reasoning over KBs to reasoning over concepts. Although the aim of this chapter is to present an algorithm for knowledge base satisfiability, the reduction in this section and the canonical model property in the next one are also important for the the query entailment algorithm in the following chapter, thus we formulate the claims for the more general setting of query entailment. Specifically, in this section we rewrite a given knowledge base $\mathcal{K}$ into a concept $C_\mathcal{K}$ in a restricted form, in such a way that the entailment of a query q by $\mathcal{K}$ reduces to (a suitable notion of) entailment of q by $C_\mathcal{K}$.

Recall from Definition 2.1.37 that a concept is in *negation normal form* (NNF), if $\neg$ is applied only to atomic concepts, and $\setminus$ only to atomic roles. If additionally it contains no special symbols, we call it *normal*.

Definition 3.1.1 (normal concepts) *A $\mathcal{ZOIQ}$ concept C is* normal *if it is in NNF and does not contain $\top$, $\bot$, T, or B.*

The rewriting of a knowledge base into a normal concept is as follows.

Definition 3.1.2 ($R_\mathcal{K}$, KB transformations, $C_\mathcal{K}$) *Given a knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, the role $R_\mathcal{K}$ is defined as*

- $R_\mathcal{K} = \bigcup_{p \in \mathsf{N_R}(\mathcal{K})} p$ *if no inverse roles p^- occur in $\mathcal{K}$, and*
- $R_\mathcal{K} = \bigcup_{p \in \mathsf{N_R}(\mathcal{K})} p \cup p^-$ *otherwise.*

We define the following transformations on KBs. When applied to $\mathcal{K}_\mathsf{i} = \langle \mathcal{A}_\mathsf{i}, \mathcal{T}_\mathsf{i}, \mathcal{R}_\mathsf{i} \rangle$, a transformation results in the KB $\mathcal{K}_\mathsf{o} = \langle \mathcal{A}_\mathsf{o}, \mathcal{T}_\mathsf{o}, \mathcal{R}_\mathsf{o} \rangle$.

1. ***ABox reduction.*** *$\mathcal{A}_\mathsf{i}$ is transformed into an* extensionally reduced ABox *$\mathcal{A}_\mathsf{o}$ in which only concept and role names are used, possibly adding new CIAs.*

   The resulting $\mathcal{A}_\mathsf{o}$, $\mathcal{T}_\mathsf{o}$ and $\mathcal{R}_\mathsf{o}$ are the smallest sets such that $\mathcal{T}_\mathsf{i} \subseteq \mathcal{T}_\mathsf{o}$, $\mathcal{R}_\mathsf{i} \subseteq \mathcal{R}_\mathsf{o}$ and:

   - *if $S(a,b) \in \mathcal{A}_\mathsf{i}$ then $p_S(a,b) \in \mathcal{A}_\mathsf{o}$ and $p_S \sqsubseteq S \in \mathcal{R}_\mathsf{o}$ for a fresh $p_S \in \mathsf{N_R}$, and*
   - *if $C(a) \in \mathcal{A}_\mathsf{i}$ then $A_C(a) \in \mathcal{A}_\mathsf{o}$ and $A_C \sqsubseteq C \in \mathcal{T}_\mathsf{o}$ for a fresh $A_C \in \mathsf{N_C}$.*

2. ***BRIA elimination.*** *BRIAs $S_1 \sqsubseteq S_2$ are replaced by CIAs $\exists (S_1 \setminus S_2).\top \sqsubseteq \bot$ (cf. [RKH08a]).*

   In the resulting $\mathcal{K}_\mathsf{o}$, the ABox $\mathcal{A}_\mathsf{o} = \mathcal{A}_\mathsf{i}$ remains unchanged, while $\mathcal{R}_\mathsf{o} = \emptyset$ and

   $$\mathcal{T}_\mathsf{o} = \mathcal{T}_\mathsf{i} \cup \{\exists (S_1 \setminus S_2).\top \sqsubseteq \bot \mid S_1 \sqsubseteq S_2 \in \mathcal{R}_\mathsf{i}\}$$

3. ***Elimination of $\top$, $\bot$ and B.*** *The symbols $\top$, $\bot$ and B are simulated using other concept and role names. Let $A_\top$ and $A_\bot$ be fresh concept names, and let p_B be a fresh role name.*

For an ABox assertion, a TBox axiom or an RBox axiom γ, let γ' denote the result of replacing each occurrence of $\top$ with $A_\top$, of $\bot$ with $A_\bot$, and of B with p_B. Then

$$\begin{array}{rl} \mathcal{A}_\mathsf{o} = & \{\gamma' \mid \gamma \in \mathcal{A}_\mathsf{i}\}, \\ \mathcal{T}_\mathsf{o} = & \{\gamma' \mid \gamma \in \mathcal{T}_\mathsf{i}\} \cup \{\top \sqsubseteq \forall p_\mathsf{B}.\bot, A \sqcup \neg A \sqsubseteq A_\top, A_\top \sqsubseteq \neg A_\bot\}, \text{ and} \\ \mathcal{R}_\mathsf{o} = & \{\gamma' \mid \gamma \in \mathcal{R}_\mathsf{i}\}, \end{array}$$

where A is an arbitrary concept name.

4. ***Elimination of*** T. *The symbol T is simulated using a complex role expression, possibly adding some assertions and axioms to the KB.*

   Let p_T be a fresh role name, let $a_1, \ldots, a_m$ be an arbitrary enumeration of the individuals in $\mathsf{N_I}(\mathcal{K})$, and let

   $$\mathcal{A}_\mathsf{T} = \{p_\mathsf{T}(a_1, a_2), p_\mathsf{T}(a_2, a_3), \ldots, p_\mathsf{T}(a_{m-1}, a_m), p_\mathsf{T}(a_m, a_1)\}.$$

   Let $R_\mathsf{T} = (p_\mathsf{T} \cup R_\mathcal{K})^*$, and for an ABox assertion, a TBox axiom or an RBox axiom γ, let γ' denote the result of replacing each occurrence of T with R_T. Then

   $$\begin{array}{rl} \mathcal{A}_\mathsf{o} = & \{\gamma' \mid \gamma \in \mathcal{A}_\mathsf{i}\} \cup \mathcal{A}_\mathsf{T}, \\ \mathcal{T}_\mathsf{o} = & \{\gamma' \mid \gamma \in \mathcal{T}_\mathsf{i}\} \cup \{\top \sqsubseteq \exists p_\mathsf{T}.\{a_1\} \in \mathcal{T}_\mathsf{o}\}, \text{ if there are no inverse roles } p^- \text{ in } R_\mathcal{K}, \\ \mathcal{T}_\mathsf{o} = & \{\gamma' \mid \gamma \in \mathcal{T}_\mathsf{i}\}, \text{ otherwise, and} \\ \mathcal{R}_\mathsf{o} = & \{\gamma' \mid \gamma \in \mathcal{R}_\mathsf{i}\}, \end{array}$$

5. ***ABox internalization.*** *Using nominals, the ABox is* internalized *into the TBox in the usual way.*

   That is, $\mathcal{A}_\mathsf{o} = \emptyset$, $\mathcal{R}_\mathsf{o} = \mathcal{R}_\mathsf{i}$, and $\mathcal{T}_\mathsf{o} = \mathcal{T}_\mathsf{i} \cup \mathcal{T}_{\mathcal{A}_\mathsf{i}}$, where

   $$\mathcal{T}_{\mathcal{A}_\mathsf{i}} = \{\{a\} \sqsubseteq A \mid A(a) \in \mathcal{A}_\mathsf{i}\} \cup \{\{a\} \sqsubseteq \exists p.\{b\} \mid p(a,b) \in \mathcal{A}_\mathsf{i}\} \cup \{\{a\} \sqsubseteq \neg\{b\} \mid a \not\approx b \in \mathcal{A}_\mathsf{i}\}.$$

For a given $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R}\rangle$, let $\hat{\mathcal{K}} = \langle \emptyset, \hat{\mathcal{T}}, \emptyset\rangle$ be obtained by applying successively the transformations 1 to 5, in the given order. The next transformation relies rewrites $\mathcal{T}$ into a concept $C_\mathcal{K}$.

TBox internalization. *Using the role $R_{\hat{\mathcal{K}}}$, the set of CIAs $\hat{\mathcal{T}}$ is* internalized *into the following concept (cf. [Baa91, Sch91]):*

$$C_\mathcal{K} = \forall (R_{\hat{\mathcal{K}}})^*. \bigsqcap_{C_1 \sqsubseteq C_2 \in \hat{\mathcal{T}}} nnf(\neg C_1 \sqcup C_2)$$

The rewriting of $\mathcal{K}$ into a concept $C_\mathcal{K}$ creates new nominals for the individuals in the ABox, hence if we apply it to a $\mathcal{ZIQ}$ KB, we obtain a $\mathcal{ZOIQ}$ concept. We will later exploit the following restriction on the nominals in this concept.

Definition 3.1.3 (nominal restricted concepts) *A $\mathcal{ZOIQ}$ concept is* nominal restricted *if it can be written as $C \sqcap \forall R^*.(C_1 \sqcap \ldots \sqcap C_n)$, where C and R are nominal free, and each C_i is of the form $\neg\{a\} \sqcup A$, $\neg\{a\} \sqcup \exists p.\{b\}$, or $\neg\{a\} \sqcup \neg\{b\}$.*

We show below that an interpretation is a model of $\mathcal{K}$ if and only if all its elements satisfy $C_\mathcal{K}$, and that the latter holds if and only if the interpretation of each individual node is in the interpretation of $C_\mathcal{K}$. Hence the following special notion of model of $C_\mathcal{K}$ captures the satisfiability of $\mathcal{K}$ in a convenient way:

Definition 3.1.4 (ind-model) *Given a $\mathcal{ZOIQ}$ concept C and an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, we say that $\mathcal{I}$ is a* model of C w.r.t. the individuals*, or for short, that $\mathcal{I}$ is an* ind-model of C*, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every $a \in \mathsf{N_I}(C)$; in symbols, $\mathcal{I} \models_{\mathsf{ind}} C$. Given a P2RPQ q, we write $C \models_{\mathsf{ind}} q$ if $\mathcal{I} \models q$ for each $\mathcal{I}$ with $\mathcal{I} \models_{\mathsf{ind}} C$.*

Reasoning—in particular, deciding satisfiability and query entailment—over the models of $\mathcal{K}$ is effectively reduced to reasoning over the ind-models of $C_{\mathcal{K}}$.

Proposition 3.1.5 *Given a $\mathcal{ZOIQ}$ KB $\mathcal{K}$, one can construct in linear time a normal concept $C_{\mathcal{K}}$ such that:*

1. *for every P2RPQ q, $\mathcal{K} \models q$ iff $C_{\mathcal{K}} \models_{\mathsf{ind}} q$;*
2. *if $\mathcal{K}$ is in $\mathcal{L}$, then $C_{\mathcal{K}}$ is in $\mathcal{L}$, for any $\mathcal{L}$ of $\mathcal{ZOIQ}$, $\mathcal{ZOI}$, or $\mathcal{ZOQ}$; and*
3. *if $\mathcal{K}$ is in $\mathcal{ZIQ}$, then $C_{\mathcal{K}}$ is a nominal restricted $\mathcal{ZOIQ}$ concept.*

Proof. It is not hard to see that each of the KB transformations 1 to 5 is linear in $||\mathcal{K}||$, so $\hat{\mathcal{K}}$ can be obtained in linear time. The rewriting of $\hat{\mathcal{T}}$ into $C_{\mathcal{K}}$ is also linear in $||\mathcal{K}||$. Since each of the transformations 1 to 5 preserves all the properties enforced by the preceding transformations, if applied to $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ in the described order, it results in a KB $\hat{\mathcal{K}} = \langle \emptyset, \hat{\mathcal{T}}, \emptyset \rangle$ such that in the TBox $\hat{\mathcal{T}}$ special symbols $\top$, $\bot$, B and T do not occur. Hence there are no special symbols in $C_{\mathcal{K}}$, and since it is in NNF, it is a normal concept.

To show item 1, we first consider the KB transformations 1 to 5. It is well known that the ABox reduction introduces new symbols, but does not change the semantics of the existing ones. That is, the resulting KB is a *conservative extension* of the original one and their models are identical modulo the new symbols in the signature. The BRIA elimination does not change the models, i.e., for every interpretation $\mathcal{I}$ and every KB $\langle \mathcal{A}_{\mathsf{i}}, \mathcal{T}_{\mathsf{i}}, \mathcal{R}_{\mathsf{i}} \rangle$, we have $\mathcal{I} \models \langle \mathcal{A}_{\mathsf{i}}, \mathcal{T}_{\mathsf{i}}, \mathcal{R}_{\mathsf{i}} \rangle$ iff $\mathcal{I} \models \langle \mathcal{A}_{\mathsf{o}}, \mathcal{T}_{\mathsf{o}}, \mathcal{R}_{\mathsf{o}} \rangle$ [RKH08a]. The elimination of $\top$, $\bot$ and B also results in a conservative extension of the original KB. The elimination of T is neither equivalence preserving nor a conservative extension in general (in fact, T can not be expressed in $\mathcal{ZOIQ}$ without using a special symbol), but under certain conditions it does preserve models in a way that is enough for our purposes.

We say that an interpretation $\mathcal{I}$ is $R_{\mathcal{K}}$*-connected* if for each $d \in \Delta^{\mathcal{I}}$, there is some $a \in \mathsf{N_I}(\mathcal{K})$ and some sequence $d_0, \ldots, d_n$ such that $a^{\mathcal{I}} = d_0$, $d = d_n$ and for each $0 \leq i < n$ we have $(d_i, d_{i+1}) \in {R_{\mathcal{K}}}^{\mathcal{I}}$. We now show that R_{T} correctly simulates the universal role over $R_{\mathcal{K}}$-connected models of $\mathcal{K}$. First, observe that if $\mathcal{I}$ is $R_{\mathcal{K}}$-connected then for every d there is some individual a_i such that $({a_i}^{\mathcal{I}}, d) \in {R_{\mathsf{T}}}^{\mathcal{I}}$, as $R_{\mathcal{K}}$ is contained in R_{T}. Further, if there are inverse roles in $R_{\mathcal{K}}$, then $R_{\mathcal{K}}$ is symmetric and this also implies $(d, {a_i}^{\mathcal{I}}) \in {R_{\mathsf{T}}}^{\mathcal{I}}$. The assertions in $\mathcal{A}_{\mathsf{T}} = \{p_{\mathsf{T}}(a_1, a_2), p_{\mathsf{T}}(a_2, a_3), \ldots, p_{\mathsf{T}}(a_{m-1}, a_m), p_{\mathsf{T}}(a_m, a_1)\}$ ensure the existence of a p_{T} path between (the interpretations of) any two individuals, i.e., $\{({a_i}^{\mathcal{I}}, {a_j}^{\mathcal{I}}) \mid a_i, a_j \in \mathsf{N_I}(\mathcal{K})\} \subseteq ({p_{\mathsf{T}}}^*)^{\mathcal{I}}$ for every $\mathcal{I}$ that is a model of $\mathcal{A}_{\mathsf{T}}$. As ${p_{\mathsf{T}}}^*$ is also contained in R_{T}, and R_{T} is transitively closed, we get that in the presence of inverse roles $(d^{\mathcal{I}}, {d'}^{\mathcal{I}}) \in R_{\mathsf{T}}$ for every $d, d' \in \Delta^{\mathcal{I}}$ as desired. If there are no inverse roles we know that for every d there is some a_i such that $({a_i}^{\mathcal{I}}, d) \in {R_{\mathsf{T}}}^{\mathcal{I}}$, but $(d, {a_i}^{\mathcal{I}}) \in {R_{\mathsf{T}}}^{\mathcal{I}}$ need not hold. However, by adding the CIA $\top \sqsubseteq \exists p_{\mathsf{T}}.\{a_1\}$ to the TBox, we ensure that $(d, {a_1}^{\mathcal{I}}) \in {R_{\mathsf{T}}}^{\mathcal{I}}$ holds for every d, and then $(d^{\mathcal{I}}, {d'}^{\mathcal{I}}) \in R_{\mathsf{T}}$ for every pair d, d' easily follows. Thus the transformation correctly simulates the T role in $R_{\mathcal{K}}$-connected models and the resulting KB is a conservative extension of the original one. In the following claim we show that it preserves query entailment.

Claim: Let $\mathcal{K}$ be a $\mathcal{ZOIQ}$ KB. Then, for every P2RPQ q, $\mathcal{K} \not\models q$ implies that there is a $R_{\mathcal{K}}$-connected $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$ and $\mathcal{I} \not\models q$.

To show the claim, we consider any model $\mathcal{I}$ of $\mathcal{K}$ with $\mathcal{I} \not\models q$ and restrict it to the elements $d \in \Delta^{\mathcal{I}}$ for which an individual a and a sequence $d_0, \ldots, d_n$ as above exist; the resulting interpretation

$\mathcal{I}'$ is clearly $R_\mathcal{K}$-connected. It is easy to verify that $\mathcal{I}' \models \mathcal{K}$. Roughly, for each $d \in \Delta^\mathcal{I}$, removing elements not reachable from d does not alter the satisfaction of any concept at d, nor the participation of d in the extension $r^\mathcal{I}$ of any role r occurring in $\mathcal{K}$. Hence no CIA or BRIA is violated in $\mathcal{I}'$. The ABox also remains satisfied, since in $\mathcal{I}'$ all domain elements interpreting some ABox individual remain unchanged, and they participate in the same concept and roles as in $\mathcal{I}$. Finally, since any query match in $\mathcal{I}'$ would also be a query match in $\mathcal{I}$, $\mathcal{I} \not\models q$ implies $\mathcal{I}' \not\models q$ and the claim holds.

Clearly, ABox internalization results in a KB with exactly the same models. Hence after transformations 1 to 5, and since each symbol in q appears in $\mathcal{K}$, we have that $\mathcal{K} \models q$ iff $\hat{\mathcal{K}} \models q$.

By definition $\mathcal{I} \models \hat{\mathcal{K}} = \langle \emptyset, \mathcal{T}, \emptyset \rangle$ iff $d \in \neg C \sqcup D$ for every $d \in \Delta^\mathcal{I}$ and every $C \sqsubseteq D \in \mathcal{T}$. Furthermore, if $d \in (C_\mathcal{K})^\mathcal{I}$ for some d, then $d' \in (C_\mathcal{K})^\mathcal{I}$ for all d that are $R_\mathcal{K}$-connected to d [Baa91, Sch91]. In particular, in a $R_\mathcal{K}$-connected interpretation $\mathcal{I}$, we clearly have $\mathcal{I} \models \hat{\mathcal{K}}$ iff $a^\mathcal{I} \in C^\mathcal{I}_\mathcal{K}$ for each $a \in \mathsf{N_I}(\mathcal{K})$. By this $\mathcal{K} \models q$ implies $C_\mathcal{K} \models_{\mathsf{ind}} q$, and together with the claim above, $\mathcal{K} \not\models q$ implies $C_\mathcal{K} \not\models_{\mathsf{ind}} q$. This concludes the proof of the first item.

Item 2 holds because none of the transformations introduces number restrictions, and the only complex role they introduce is $R_\mathcal{K}$, which does not contain inverses if $\mathcal{K}$ is a $\mathcal{ZOQ}$ KB. Item 3 holds because there are no nominals in $\mathcal{T}$, and the only nominals in $C_\mathcal{K}$ are those in $\mathcal{T}_\mathcal{A}$ that were introduced in the ABox internalization step. That is, $C_\mathcal{K}$ can be written as follows (observe that $\forall R^*.(C \sqcap C')$ is equivalent to $\forall R^*.C \sqcap \forall R^*.C'$):

$$\big(\forall (R_{\hat{\mathcal{K}}})^*. \bigsqcap_{C_1 \sqsubseteq C_2 \in \hat{\mathcal{T}}} nnf(\neg C_1 \sqcup C_2)\big) \sqcap \big(\forall (R_{\hat{\mathcal{K}}})^*. \bigsqcap_{C_1 \sqsubseteq C_2 \in \mathcal{T}_\mathcal{A}} \neg C_1 \sqcup C_2\big)$$

As there are no nominals in $\hat{\mathcal{T}}$ or in $R_{\hat{\mathcal{K}}}$, the left conjunct is nominal free. For each $C_1 \sqsubseteq C_2 \in \mathcal{T}_\mathcal{A}$, the corresponding $\neg C_1 \sqcup C_2$ is of one of the allowed forms $\neg\{a\} \sqcup A$, $\neg\{a\} \sqcup \exists p.\{b\}$, or $\neg\{a\} \sqcup \neg\{b\}$. Hence $C_\mathcal{K}$ is nominal restricted. □

3.2 Canonical Models

We have shown that satisfiability of $\mathcal{ZOIQ}$ knowledge bases can be reduced to the existence of ind-models of normal concepts. In the rest of this chapter we show that using automata on forests we can decide the existence of a special kind of ind-models, which we call *canonical*, that can be suitably represented as a labeled forest.

In what follows, we assume a fixed normal $\mathcal{ZOIQ}$ concept D. We will show that if D has an ind-model and either is in $\mathcal{ZOQ}$, is in $\mathcal{ZOI}$, or is nominal restricted, then it has a canonical model. By this, the algorithm to decide the existence of a canonical model for D that we describe below will be enough to decide KB satisfiability in the logics $\mathcal{ZOQ}$, $\mathcal{ZOI}$ and $\mathcal{ZIQ}$.

3.2.1 Syntactic Closure

Before showing the canonical model property, we introduce the *(syntactic) closure* of D, which contains all the concepts and simple roles that are relevant for deciding its satisfiability. More specifically it contains D, it is closed under subconcepts and simple subroles, as well as under negations in NNF, and it is also *Fischer-Ladner closed* in the style of the well known closure for PDL [FL79].

For defining the closure we modify the syntax of simple roles, and instead of role difference $S \setminus S'$, we use negation $\neg S$ as a simple role constructor. Semantically, $\neg S^\mathcal{I} = (\Delta^\mathcal{I} \times \Delta^\mathcal{I}) \setminus S^\mathcal{I}$,

if	$C \in Cl_{\mathsf{C}}(D)$	then	$\sim C \in Cl_{\mathsf{C}}(D)$
if	$C_1 \boxdot C_2 \in Cl_{\mathsf{C}}(D)$	then	$C_1, C_2 \in Cl_{\mathsf{C}}(D)$
if	$\exists S.C \in Cl_{\mathsf{C}}(D)$	then	$\geqslant 1\, S.C \in Cl_{\mathsf{C}}(D)$
if	$\forall S.C \in Cl_{\mathsf{C}}(D)$	then	$\leqslant 0\, S.{\sim}C \in Cl_{\mathsf{C}}(D)$
if	$\geqslant n\, S.C \in Cl_{\mathsf{C}}(D)$	then	$C \in Cl_{\mathsf{C}}(D)$
if	$\exists(R_1 \cup R_2).C \in Cl_{\mathsf{C}}(D)$	then	$\exists R_1.C \sqcup \exists R_2.C \in Cl_{\mathsf{C}}(D)$
if	$\exists(R_1 \circ R_2).C \in Cl_{\mathsf{C}}(D)$	then	$\exists R_1.\exists R_2.C \in Cl_{\mathsf{C}}(D)$
if	$\exists R^*.C \in Cl_{\mathsf{C}}(D)$	then	$C \sqcup \exists R.\exists R^*.C \in Cl_{\mathsf{C}}(D)$
if	$\exists id(C_1).C_2 \in Cl_{\mathsf{C}}(D)$	then	$C_1 \sqcap C_2 \in Cl_{\mathsf{C}}(D)$
if	$\forall(R_1 \cup R_2).C \in Cl_{\mathsf{C}}(D)$	then	$\forall R_1.C \sqcap \forall R_2.C \in Cl_{\mathsf{C}}(D)$
if	$\forall(R_1 \circ R_2).C \in Cl_{\mathsf{C}}(D)$	then	$\forall R_1.\forall R_2.C \in Cl_{\mathsf{C}}(D)$
if	$\forall R^*.C \in Cl_{\mathsf{C}}(D)$	then	$C \sqcap \forall R.\forall R^*.C \in Cl_{\mathsf{C}}(D)$
if	$\forall id(C_1).C_2 \in Cl_{\mathsf{C}}(D)$	then	$\sim C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(D)$

if	$\exists S.\mathsf{Self} \in Cl_{\mathsf{C}}(D)$	then	$S \in Cl_{\mathsf{R}}(D)$
if	$\geqslant n\, S.C \in Cl_{\mathsf{C}}(D)$	then	$S \in Cl_{\mathsf{R}}(D)$
if	$S \in Cl_{\mathsf{R}}(D)$	then	$\sim S \in Cl_{\mathsf{R}}(D)$
if	some inverse p^- occurs in D and $S \in Cl_{\mathsf{R}}(D)$	then	$\mathsf{Inv}(S) \in Cl_{\mathsf{R}}(D)$
if	$S_1 \bigcirc S_2 \in Cl_{\mathsf{R}}(D)$	then	$S_1, S_2 \in Cl_{\mathsf{R}}(D)$

Table 3.1: Syntactic closure $Cl(D) = Cl_{\mathsf{C}}(D) \cup Cl_{\mathsf{R}}(D)$ (where $\boxdot \in \{\sqcup, \sqcap\}$, $\bigcirc \in \{\cap, \cup\}$)

hence $S \setminus S'$ can be expressed as $S \cap \neg S'$. A simple role in this extended syntax is called *safe* if it is equivalent to a standard simple role in $\mathcal{ZOIQ}$.[1]

In what follows, $\sim E$ denotes the NNF of $\neg E$, for a concept or simple role E. The symbol $\gtrless$ is generic for $\geqslant$ or $\leqslant$, $\boxdot$ for $\sqcap$ and $\sqcup$, and $\bigcirc$ for $\cap$ and $\cup$. Recall that for a role name $p \in \mathsf{N}_{\mathsf{R}}$, the *inverse of* p is p^- and the inverse of p^- is p; the inverse of an atomic role P is denoted $\mathsf{Inv}(P)$. For any simple role S, $\mathsf{Inv}(S)$ denotes the role obtained by replacing each atomic role P occurring in S by its inverse $\mathsf{Inv}(P)$. As usual, possibly subindexed C, S and R respectively stand for concepts, simple roles, and arbitrary roles.

Definition 3.2.1 *The* concept closure $Cl_{\mathsf{C}}(D)$ *of* D *is the smallest set of* $\mathcal{ZOIQ}$ *concepts that contains* D *and is closed under the rules in the upper part of Table 3.1. The* role closure $Cl_{\mathsf{R}}(D)$ *of* D *is defined in terms of the concept closure, and it is the smallest set of possibly negated simple roles closed under the rules in the lower part of Table 3.1. We call their union* $Cl(D) = Cl_{\mathsf{C}}(D) \cup Cl_{\mathsf{R}}(D)$ *the* (syntactic) closure *of* D.

Clearly, $|Cl(D)|$ is linear in the length of D. Please note that although $Cl(D)$ contains negated roles (which are not safe in general), S is safe in every concept of the form $\gtrless n\, S.C \in Cl(D)$.

3.2.2 Canonical Model Property

We now define *canonical models* for a $\mathcal{ZOIQ}$ concept D. The domain of such a model is a b-forest, for some b that depends only on D, whose roots are *exactly* the elements interpreting the ABox individuals. It is connected and each node may only be directly connected to itself, its parent, its children, and to the roots.

[1] A role is safe iff every clause in its conjunctive normal form has a positive atomic role as a disjunct, cf. [RKH08a].

Recall that $\mathsf{N_C}(D)$, $\mathsf{N_R}(D)$, and $\mathsf{N_I}(D)$ respectively denote the sets of concept names, role names, and individuals occurring in a concept D.

Definition 3.2.2 (canonical interpretation, canonical model) *Let D be a $\mathcal{ZOIQ}$ concept. An interpretation $\mathcal{I}$ is* canonical (for D) *if:*

1. $\Delta^{\mathcal{I}}$ *is a b_D-forest, where* $b_D = |Cl_{\mathsf{C}}(D)| \cdot \max(\{n \mid \geqslant n\, S.C \in Cl_{\mathsf{C}}(D)\} \cup \{0\})$;
2. $\mathsf{roots}(\Delta^{\mathcal{I}}) = \{a^{\mathcal{I}} \mid a \in \mathsf{N_I}(D)\}$
3. $\mathcal{I}$ *is* connected, *that is, for each pair* $\{w, w'\} \subseteq \Delta^{\mathcal{I}}$ *with* $w' \in \mathsf{succ}(w)$ *there is some* $P \in \overline{\mathsf{N_R}}(\mathcal{K})$ *such that* $(w, w') \in P^{\mathcal{I}}$, *and*
4. *for every* $w, w' \in \Delta^{\mathcal{I}}$ *such that* $(w, w') \in p^{\mathcal{I}}$ *for some role name* p, *either*
   a) $w' \in \mathsf{succ}(w)$,
   b) $w \in \mathsf{succ}(w')$,
   c) $\{w, w'\} \cap \mathsf{roots}(\Delta^{\mathcal{I}}) \neq \emptyset$, *or*
   d) $w = w'$.

We call $\mathcal{I}$ a canonical model of D *if $\mathcal{I} \models_{\mathsf{ind}} D$. Note that, by definition, $\mathcal{I} \models_{\mathsf{ind}} D$ iff $\mathsf{roots}(\Delta^{\mathcal{I}}) \subseteq D^{\mathcal{I}}$.*

Note that, since $\Delta^{\mathcal{I}}$ is a b_D-forest with $\mathsf{roots}(\Delta^{\mathcal{I}}) = \{a^{\mathcal{I}} \mid a \in \mathsf{N_I}(D)\}$, the notion of connectedness in item 3 implies the one used in the proof of Proposition 3.1.5.

The above definition generalizes those of related logics (e.g., in [BLMV08, CEO07, SV01]), and accommodates all constructs of $\mathcal{ZOIQ}$. As usual in logics with inverses, $(w, w') \in p^{\mathcal{I}}$ may hold if (4a) w' is a child of w or (4b) w' is the parent of w. Additionally, $(w, w') \in p^{\mathcal{I}}$ may hold if (4c) w or w' is a root of $\Delta^{\mathcal{I}}$, to accommodate nominals, and if (4d) w' is w itself, to accommodate $\exists S.\mathsf{Self}$ concepts.

Example 3.2.3 *The $\mathcal{ZOIQ}$ KB $\mathcal{K}_g$ is shown in Figure 3.1 (it is a simplified version of the knoweldge base $\mathcal{K}^{theo}$ from Example 2.1.28), and Figure 3.2 partially depicts a canonical model $\mathcal{I}_g$ of $\mathcal{K}_g$. Its roots are* $1 = \text{gaia}^{\mathcal{I}_g}$, $2 = \text{zeus}^{\mathcal{I}_g}$, $3 = \text{heracles}^{\mathcal{I}_g}$, $4 = \text{alcmene}^{\mathcal{I}_g}$, $5 = \text{electryon}^{\mathcal{I}_g}$, *and* $6 = \text{perseus}^{\mathcal{I}_g}$. *They are depicted as large dots, and each of them is labeled with the name of the individual it interprets as well as with the concept names from $\mathsf{N_I}(C_{\mathcal{K}})$ to whose interpretation it belongs. Other domain elements are represented by smaller dots, and are also labeled with the concept names to whose interpretation they belong. For readability, we use the following label names:* $L_1 = \{\mathsf{Male}, \mathsf{Deity}\}$, $L_2 = \{\mathsf{Female}, \mathsf{Deity}\}$, $L_3 = \{\mathsf{Female}, \mathsf{Mortal}\}$, $L_4 = \{\mathsf{Male}\}$, *and* $L_5 = \{\mathsf{Female}\}$. *The interpretation represented here is infinite, but only some of its domain elements are depicted. Every non-root element has two successors, which are the fulfillers of the* hasMother *and* hasFather *relations, respectively. The* hasFather *relation is represented by solid arrows, while* hasMother *is represented by dashed arrows.*

We want to show the following *canonical model property*:

Proposition 3.2.4 *Let D be a normal $\mathcal{ZOQ}$ or $\mathcal{ZOI}$ concept, or a $\mathcal{ZOIQ}$ concept that is nominal restricted. For every P2RPQ q, if $D \not\models_{\mathsf{ind}} q$, then there is a canonical model $\mathcal{I}$ of D with $\mathcal{I} \not\models q$.*

$$
\begin{array}{rr}
\mathsf{Male}\,(\mathrm{zeus}) & \mathsf{hasFather}\,(\mathrm{heracles}, \mathrm{zeus}) \\
\mathsf{Deity}\,(\mathrm{zeus}) & \mathsf{hasMother}\,(\mathrm{heracles}, \mathrm{alcmene}) \\
\mathsf{Female}\,(\mathrm{alcmene}) & \mathsf{hasFather}\,(\mathrm{alcmene}, \mathrm{electryon}) \\
\mathsf{Mortal}\,(\mathrm{alcmene}) & \mathsf{hasFather}\,(\mathrm{electryon}, \mathrm{perseus}) \\
\mathsf{Hero}\,(\mathrm{heracles}) & \mathsf{hasFather}\,(\mathrm{perseus}, \mathrm{zeus})
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{Male} & \equiv & \neg\mathsf{Female} \\
\mathsf{Mortal} & \sqsubseteq & \neg\mathsf{Deity} \\
\mathsf{Primordial} & \sqsubseteq & \mathsf{Deity} \\
\neg\mathsf{Primordial} & \sqsubseteq & \exists\mathsf{hasFather}.\mathsf{Male} \sqcap \exists\mathsf{hasMother}.\mathsf{Female} \\
\mathsf{Mortal} & \sqsubseteq & \leqslant 2\,(\mathsf{hasMother} \cup \mathsf{hasFather}).\top \\
\mathsf{Deity} & \sqsubseteq & \forall(\mathsf{hasMother} \cup \mathsf{hasFather})^*.\mathsf{Deity} \\
\mathsf{Deity} & \sqsubseteq & \exists(\mathsf{hasMother} \cup \mathsf{hasFather})^*.\mathsf{Primordial} \\
\mathsf{Primordial} & \equiv & \{\mathrm{gaia}\} \\
\\
\exists(\mathsf{hasMother} \cup \mathsf{hasFather}).\mathsf{Self} & \sqsubseteq & \bot \\
(\mathsf{hasMother} \cup \mathsf{hasFather}) \cap \mathsf{hasChild} & \sqsubseteq & \mathsf{B}
\end{array}
$$

Figure 3.1: A knowledge base $\mathcal{K}_g$ in $\mathcal{ZOIQ}$

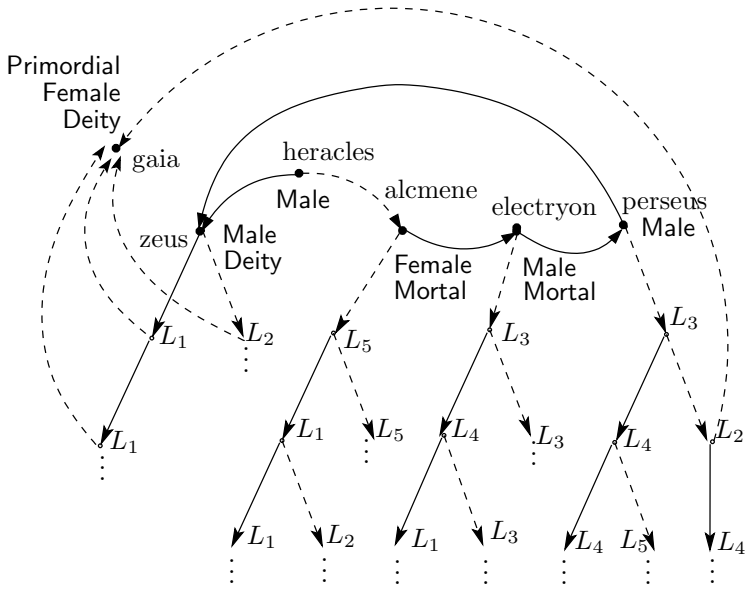

Figure 3.2: A canonical $\mathcal{I}_g$ model for $\mathcal{K}_g$

We prove Proposition 3.2.4 following the lines of similar proofs for variations of the μ-calculus in [BLMV08, Var98] (which in turn, are adaptations of the original proof in [SE89]), and show that we can 'unravel' every model $\mathcal{I}$ of D into a canonical $\mathcal{I}'$, and that this unraveling preserves query non-entailment. The presence of additional constructs makes some arguments more involved, but does not affect the core of the proof. On the other hand, some aspects are simpler in our setting because we do not need to deal with the alternating fixed points present in the μ-calculus.

The proof requires some technical intermediate steps. We show that if D has a model, then it has a *well-founded adorned pre-model*. Roughly, the latter is an interpretation enhanced with

For B atomic, $\mathcal{I}, d \models_{\mathsf{pre}} B$	iff	$d \in B^{\mathcal{I}}$, and
$\mathcal{I}, d \models_{\mathsf{pre}} \neg B$	iff	$d \notin B^{\mathcal{I}}$
$\mathcal{I}, d \models_{\mathsf{pre}} C_1 \sqcap C_2$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} C_1$ and $\mathcal{I}, d \models_{\mathsf{pre}} C_2$
$\mathcal{I}, d \models_{\mathsf{pre}} C_1 \sqcup C_2$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} C_1$ or $\mathcal{I}, d \models_{\mathsf{pre}} C_2$
$\mathcal{I}, d \models_{\mathsf{pre}} \exists(R_1 \cup R_2).C_1$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} \exists R_1.C_1 \sqcup \exists R_2.C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \exists(R_1 \circ R_2).C_1$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} \exists R_1.\exists R_2.C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \exists R^*.C_1$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} C_1 \sqcup \exists R.\exists R^*.C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \exists id(C_1).C_2$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} C_1 \sqcap C_2$
$\mathcal{I}, d \models_{\mathsf{pre}} \exists S.C_1$ with S simple	iff	$\mathcal{I}, d \models_{\mathsf{pre}} \geqslant 1\, S.C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \forall(R_1 \cup R_2).C_1$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} \forall R_1.C_1 \sqcap \forall R_2.C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \forall(R_1 \circ R_2).C_1$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} \forall R_1.\forall R_2.C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \forall R^*.C_1$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} C_1 \sqcap \forall R.\forall R^*.C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \forall id(C_1).C_2$	iff	$\mathcal{I}, d \models_{\mathsf{pre}} \sim C_1 \sqcup C_2$
$\mathcal{I}, d \models_{\mathsf{pre}} \forall S.C_1$ with S simple	iff	$\mathcal{I}, d \models_{\mathsf{pre}} \leqslant 0\, S.\sim C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \geqslant n\, S.C_1$	iff	there exists a set $V \subseteq \Delta^{\mathcal{I}}$ such that $\lvert V\rvert \geq n$ and for each $d' \in V$, $(d, d') \in S^{\mathcal{I}}$ and $\mathcal{I}, d' \models_{\mathsf{pre}} C_1$
$\mathcal{I}, d \models_{\mathsf{pre}} \leqslant n\, S.C_1$	iff	there exists a set $V \subseteq \Delta^{\mathcal{I}}$ such that $\lvert V\rvert \leq n$ and $\mathcal{I}, d' \models_{\mathsf{pre}} \sim C_1$ for each $d' \in \Delta^{\mathcal{I}} \setminus V$ such that $(d, d') \in S^{\mathcal{I}}$
$\mathcal{I}, d \models_{\mathsf{pre}} \exists S.\mathsf{Self}$	iff	$(d, d) \in S^{\mathcal{I}}$

Table 3.2: Pre-satisfaction of a normal concept at an element $d \in \Delta^{\mathcal{I}}$.

additional information that allows us to 'trace' the satisfaction of concepts of the form $\exists R^*.C$. Then we show that a well-founded adorned pre-model can be unraveled into a well-founded adorned pre-model that contains a canonical model of D.

In this and other proofs below, it will sometimes be convenient to talk about a special, weaker type of concept satisfaction that we call *pre-satisfaction*. It is defined relative to one specific domain element d. Intuitively, $\mathcal{I}, d \models_{\mathsf{pre}} C$ is almost equivalent to $d \in C^{\mathcal{I}}$, but for concepts of the form $\exists R^*.C$ pre-satisfaction is less strict and it does not require that C is eventually satisfied. Instead, $\mathcal{I}, d \models_{\mathsf{pre}} \exists R^*.C$ holds also if from d there is an infinite sequence of (not necessarily distinct) elements related by R, and does not require any of them to be an instance of C.

Definition 3.2.5 (pre-satisfaction, eventual realization) *For an interpretation $\mathcal{I}$, an element $d \in \Delta^{\mathcal{I}}$ and a normal concept C, the* pre-satisfaction *relation $\mathcal{I}, d \models_{\mathsf{pre}} C$ is inductively defined in Table 3.2.*

For a concept $\exists R^*.C_1$ and a $d \in \Delta^{\mathcal{I}}$ such that $\mathcal{I}, d \models_{\mathsf{pre}} \exists R^*.C_1$, we say that $\exists R^*.C_1$ is eventually realized for *d if there is a finite sequence $d_0, \ldots, d_n$, $n \geq 0$, such that $d_0 = d$, $(d_i, d_{i+1}) \in R^{\mathcal{I}}$ for every $0 \leq i < n$, and $\mathcal{I}, d_n \models_{\mathsf{pre}} C_1$.*

The following lemma is trivial:

Lemma 3.2.6 *Let $\mathcal{I}$ be an interpretation. Then for every $d \in \Delta^{\mathcal{I}}$ and every concept C, $d \in C^{\mathcal{I}}$ implies $\mathcal{I}, d \models_{\mathsf{pre}} C$. Moreover, for a each concept $\exists R^*.C_1$, if $d \in \exists R^*.{C_1}^{\mathcal{I}}$ then $\exists R^*.C_1$ is eventually realized for d.*

Towards the proof of Proposition 3.2.4, we now introduce *concept types* and *role types*, which are consistent sets of concepts and sets of roles from the syntactic closure of D, satisfying certain closure conditions. Then we define *pre-models* as interpretations in which each object is mapped

if	$C \in Cl_{\mathsf{C}}(D)$	then	$C \in \mathbf{t}$	iff	$\sim C \notin \mathbf{t}$
if	$C_1 \sqcap C_2 \in Cl_{\mathsf{C}}(D)$,	then	$C_1 \sqcap C_2 \in \mathbf{t}$	iff	$\{C_1, C_2\} \subseteq \mathbf{t}$
if	$C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(D)$,	then	$C_1 \sqcup C_2 \in \mathbf{t}$	iff	$\{C_1, C_2\} \cap \mathbf{t} \neq \emptyset$
if	$\exists(R_1 \cup R_2).C \in Cl_{\mathsf{C}}(D)$,	then	$\exists(R_1 \cup R_2).C \in \mathbf{t}$	iff	$\exists R_1.C \sqcup \exists R_2.C \in \mathbf{t}$
if	$\exists(R_1 \circ R_2).C \in Cl_{\mathsf{C}}(D)$,	then	$\exists(R_1 \circ R_2).C \in \mathbf{t}$	iff	$\exists R_1.\exists R_2.C \in \mathbf{t}$
if	$\exists R^*.C \in Cl_{\mathsf{C}}(D)$,	then	$\exists R^*.C \in \mathbf{t}$	iff	$C \sqcup \exists R.\exists R^*.C \in \mathbf{t}$
if	$\exists id(C_1).C_2 \in Cl_{\mathsf{C}}(D)$,	then	$\exists id(C_1).C_2 \in \mathbf{t}$	iff	$C_1 \sqcap C_2 \in \mathbf{t}$
if	$\exists S.C \in Cl_{\mathsf{C}}(D)$ with S simple,	then	$\exists S.C \in \mathbf{t}$	iff	$\geqslant 1\, S.C \in \mathbf{t}$
if	$\forall(R_1 \cup R_2).C \in Cl_{\mathsf{C}}(D)$,	then	$\forall(R_1 \cup R_2).C \in \mathbf{t}$	iff	$\forall R_1.C \sqcap \forall R_2.C \in \mathbf{t}$
if	$\forall(R_1 \circ R_2).C \in Cl_{\mathsf{C}}(D)$,	then	$\forall(R_1 \circ R_2).C \in \mathbf{t}$	iff	$\forall R_1.\forall R_2.C \in \mathbf{t}$
if	$\forall R^*.C \in Cl_{\mathsf{C}}(D)$,	then	$\forall R^*.C \in \mathbf{t}$	iff	$C \sqcap \forall R.\forall R^*.C \in \mathbf{t}$
if	$\forall id(C_1).C_2 \in Cl_{\mathsf{C}}(D)$,	then	$\forall id(C_1).C_2 \in \mathbf{t}$	iff	$\sim C_1 \sqcup C_2 \in \mathbf{t}$
if	$\forall S.C \in Cl_{\mathsf{C}}(D)$ with S simple,	then	$\forall S.C \in \mathbf{t}$	iff	$\leqslant 0\, S.\sim C \in \mathbf{t}$

Table 3.3: Concept type $\mathbf{t} \subseteq Cl_{\mathsf{C}}(D)$

if	P is an atomic role in $Cl_{\mathsf{R}}(D)$	then	$P \in \mathbf{r}$	iff	$\neg P \notin \mathbf{r}$
if	$S_1 \cap S_2 \in Cl_{\mathsf{R}}(D)$,	then	$S_1 \cap S_2 \in \mathbf{r}$	iff	$\{S_1, S_2\} \subseteq \mathbf{r}$
if	$S_1 \cup S_2 \in Cl_{\mathsf{R}}(D)$,	then	$S_1 \cup S_2 \in \mathbf{r}$	iff	$\{S_1, S_2\} \cap \mathbf{r} \neq \emptyset$

Table 3.4: Role type $\mathbf{r} \subseteq Cl_{\mathsf{R}}(D)$

to a concept type and each pair of objects to a role type, in such a way that the pre-satisfaction of D at each node the interpretation of all the individual nodes is ensured.

Definition 3.2.7 ((concept/role) type, pre-model) *A* concept type *of a $\mathcal{ZOIQ}$ concept D is a set $\mathbf{t} \subseteq Cl_{\mathsf{C}}(D)$ of concepts closed under the rules of Table 3.3, while a* role type *of D is a set $\mathbf{r} \subseteq Cl_{\mathsf{R}}(D)$ of simple roles closed under the rules of Table 3.4. The set of all concept types of D is denoted by $\mathsf{types}\mathsf{C}(D)$, and the set of all role types by $\mathsf{types}\mathsf{R}(D)$. For a role type $\mathbf{r}$, we define $\mathsf{Inv}(\mathbf{r}) = \{\mathsf{Inv}(S) \mid S \in \mathbf{r}\}$, that is, $\mathsf{Inv}(\mathbf{r})$ contains the inverse of each role in $\mathbf{r}$.*

A pre-model *of D is a pair $\langle \mathcal{I}, \theta \rangle$ where $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an interpretation and θ is a function that maps each $d \in \Delta^{\mathcal{I}}$ to a concept type $\theta(d) \in \mathsf{types}\mathsf{C}(D)$ and each pair $(d, d') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to a role type $\theta(d, d') \in \mathsf{types}\mathsf{R}(D)$ such that:*

1. *$D \in \theta(a^{\mathcal{I}})$ for each individual $a \in \mathsf{N_I}(D)$,*
2. *if some inverse role p^- occurs in D, then $\theta(d, d') = \mathsf{Inv}(\theta(d', d))$ for each $d, d' \in \Delta^{\mathcal{I}}$;*
3. *for each $d \in \Delta^{\mathcal{I}}$ and each individual $a \in \mathsf{N_I}(D)$, $\{a\} \in \theta(d)$ iff $d = a^{\mathcal{I}}$;*
4. *for each $d \in \Delta^{\mathcal{I}}$ and each concept name $A \in \mathsf{N_C}(D)$, $A \in \theta(d)$ iff $d \in A^{\mathcal{I}}$;*
5. *for each $d, d' \in \Delta^{\mathcal{I}}$ and each role name $p \in \mathsf{N_R}(D)$, $p \in \theta(d, d')$ iff $(d, d') \in p^{\mathcal{I}}$;*
6. *for each $d \in \Delta^{\mathcal{I}}$ and each concept of the form $\exists p.\mathsf{Self} \in Cl_{\mathsf{C}}(D)$, $\exists p.\mathsf{Self} \in \theta(d)$ iff $p \in \theta(d, d)$, and*
7. *for each $d \in \Delta^{\mathcal{I}}$*
   a) *if $\geqslant n\, S.C \in \theta(d)$, then there is some $V \subseteq \mathsf{neigh}_{\mathcal{I},\theta}(d, S)$ such that $|V| \geq n$ and $C \in \theta(d')$ for every $d' \in V$, and*

b) if $\leqslant n\, S.C \in \theta(d)$*, then there is some* $V \subseteq \mathsf{neigh}_{\mathcal{I},\theta}(d,S)$ *such that* $|V| \leq n$ *and* $\sim C \in \theta(d')$ *for every* $d' \in \mathsf{neigh}_{\mathcal{I},\theta}(d,S) \setminus V$*,*

where $\mathsf{neigh}_{\mathcal{I},\theta}(d,S) = \{d' \in \Delta^{\mathcal{I}} \mid S \in \theta(d,d')\}$.

Conditions 2 to 7 above and the closure conditions of the types imply the following:

Lemma 3.2.8 *If* $\langle \mathcal{I}, \theta \rangle$ *is a pre-model of* D*, then the following hold for each pair* $d, d' \in \Delta^{\mathcal{I}}$*:*

1. *for every* $C \in Cl_{\mathsf{R}}(D)$*,* $\mathcal{I}, d \models_{\mathsf{pre}} C$ *iff* $C \in \theta(d)$*, and*
2. *for every* $S \in Cl(D)$*,* $(d,d') \in S^{\mathcal{I}}$ *iff* $S \in \theta(d,d')$*.*

To trace the eventual realization of concepts of the form $\exists R^*.C$, we introduce *adorned pre-models* that extend pre-models with a *choice function*, and define the notion of *well-foundedness*.

Definition 3.2.9 (choice function, (well-founded) adorned pre-model) *A* choice function *for a pre-model* $\langle \mathcal{I}, \theta \rangle$ *is a partial function* ch *such that:*

- *for each pair* $(d, C_1 \sqcup C_2)$ *with* $d \in \Delta^{\mathcal{I}}$ *and* $C_1 \sqcup C_2 \in \theta(d)$*,* $\mathsf{ch}(d, C_1 \sqcup C_2)$ *is a concept in* $\{C_1, C_2\} \cap \theta(d)$*;*
- *for each pair* $(d, \geqslant n\, S.C)$ *with* $d \in \Delta^{\mathcal{I}}$ *and* $\geqslant n\, S.C \in \theta(d)$*,* $\mathsf{ch}(d, \geqslant n\, S.C)$ *is a subset* V *of* $\mathsf{neigh}_{\mathcal{I},\theta}(d,S)$ *such that* $|V| \geq n$ *and* $C \in \theta(d')$ *for every* $d' \in V$*; and*
- *for each pair* $(d, \leqslant n\, S.C)$ *with* $d \in \Delta^{\mathcal{I}}$ *and* $\leqslant n\, S.C \in \theta(d)$*,* $\mathsf{ch}(d, \leqslant n\, S.C)$ *is a subset* V *of* $\mathsf{neigh}_{\mathcal{I},\theta}(d,S)$ *such that* $|V| \leq n$ *and* $\sim C \in \theta(d')$ *for every* $d' \in \mathsf{neigh}_{\mathcal{I},\theta}(d,S) \setminus V$*.*

An adorned pre-model *is a tuple* $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$*, where* $\langle \mathcal{I}, \theta \rangle$ *is a pre-model and* ch *is a choice function for it. For an adorned pre-model* $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ *of* D*, the* derivation relation $\rightsquigarrow \subseteq (\Delta^{\mathcal{I}} \times Cl(D)) \times (\Delta^{\mathcal{I}} \times Cl(D))$ *is the smallest relation such that for every* $d \in \Delta^{\mathcal{I}}$*:*

for each $C_1 \sqcup C_2 \in \theta(d)$*,*	$(d, C_1 \sqcup C_2) \rightsquigarrow (d, \mathsf{ch}(d, C_1 \sqcup C_2))$*,*
for each $C_1 \sqcap C_2 \in \theta(d)$*,*	$(d, C_1 \sqcap C_2) \rightsquigarrow (d, C_1)$ *and* $(d, C_1 \sqcap C_2) \rightsquigarrow (d, C_2)$*,*
for each $\exists(R_1 \cup R_2).C \in \theta(d)$*,*	$(d, \exists(R_1 \cup R_2).C) \rightsquigarrow (d, \exists R_1.C \sqcup \exists R_2.C)$*,*
for each $\exists(R_1 \circ R_2).C \in \theta(d)$*,*	$(d, \exists(R_1 \circ R_2).C) \rightsquigarrow (d, \exists R_1.\exists R_2.C)$*,*
for each $\exists R^*.C \in \theta(d)$*,*	$(d, \exists R^*.C) \rightsquigarrow (d, C \sqcup \exists R.\exists R^*.C)$*,*
for each $\exists id(C_1).C_2 \in \theta(d)$*,*	$(d, \exists id(C_1).C_2) \rightsquigarrow (d, C_1 \sqcap C_2)$*,*
for each $\exists S.C \in \theta(d)$ *with* S *simple,*	$(d, \exists S.C) \rightsquigarrow (d, \geqslant 1\, S.C)$*,*
for each $\forall(R_1 \cup R_2).C \in \theta(d)$*,*	$(d, \forall(R_1 \cup R_2).C) \rightsquigarrow (d, \forall R_1.C \sqcap \forall R_2.C)$*,*
for each $\forall(R_1 \circ R_2).C \in \theta(d)$*,*	$(d, \forall(R_1 \circ R_2).C) \rightsquigarrow (d, \forall R_1.\forall R_2.C)$*,*
for each $\forall R^*.C \in \theta(d)$*,*	$(d, \forall R^*.C) \rightsquigarrow (d, C \sqcap \forall R.\forall R^*.C)$*,*
for each $\forall id(C_1).C_2 \in \theta(d)$*,*	$(d, \forall id(C_1).C_2) \rightsquigarrow (d, \sim C_1 \sqcup C_2)$*,*
for each $\forall S.C \in \theta(d)$ *with* S *simple,*	$(d, \forall S.C) \rightsquigarrow (d, \leqslant 0\, S.\sim C)$*,*
for each $\geqslant n\, S.C \in \theta(d)$*,*	$(d, \geqslant n\, S.C) \rightsquigarrow (d', C)$ *for every* $d' \in \mathsf{ch}(d, \geqslant n\, S.C)$*, and*
for each $\leqslant n\, S.C \in \theta(d)$*,*	$(d, \leqslant n\, S.C) \rightsquigarrow (d', \sim C)$ *for every* $d' \in \mathsf{neigh}_{\mathcal{I},\theta}(d,S) \setminus \mathsf{ch}(d, \leqslant n\, S.C)$*.*

A concept $\exists R^*.C$ *is* regenerated *from* d *to* d' *in* $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$*, if there is a sequence* $(d_1, C_1), \ldots, (d_n, C_n)$ *with* $n > 1$ *such that* $d_1 = d$*,* $d_n = d'$*,* $C_1 = C_n = \exists R^*.C$*,* $\exists R^*.C$ *is a subconcept of*

every C_i and $(d_i, C_i) \rightsquigarrow (d_{i+1}, C_{i+1})$ for each $1 \leq i < n$. We say that $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ is well-founded*, if there is no $\exists R^*.C \in Cl(D)$ and infinite sequence $d_1, d_2, \ldots$ such that $\exists R^*.C$ is* regenerated *from d_i to d_{i+1} for every $i \geq 1$.*

We can not show the following:

Lemma 3.2.10 *The following hold for D:*

1. *For each $\mathcal{I}$ such that $\mathcal{I} \models_{\mathsf{ind}} D$, there is a well-founded adorned pre-model $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ of D.*
2. *If $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ is a well-founded adorned pre-model of D, then $\mathcal{I} \models_{\mathsf{ind}} D$.*

Proof. [Sketch] For the first part of the claim, consider an arbitrary $\mathcal{I}$ with $\mathcal{I} \models_{\mathsf{ind}} D$. By Lemma 3.2.8, we can set $\theta(d) = \{C \in Cl_{\mathsf{C}}(D) \mid \mathcal{I}, d \models_{\mathsf{pre}} C\}$ and $\theta(d, d') = \{S \in Cl_{\mathsf{R}}(C_T) \mid (d, d') \in S^{\mathcal{I}}\}$ for every $d, d' \in \Delta^{\mathcal{I}}$ to obtain a pre-model $\langle \mathcal{I}, \theta \rangle$.

As $\langle \mathcal{I}, \theta \rangle$ is a pre-model, it can be trivially adorned with a choice function ch. Relying on the fact that in $\langle \mathcal{I}, \theta \rangle$ each $\exists R^*.C \in Cl_{\mathsf{C}}(D)$ is eventually realized for every $d \in \Delta^{\mathcal{I}}$ with $\exists R^*.C \in \theta(d)$, and show that there exists a choice function ch in which no such concept is regenerated infinitely often. Intuitively, ch can be guided by the sequence $d_0, \ldots, d_n$ with $(d_i, d_{i+1}) \in R^{\mathcal{I}}$ and $C \in \theta(d_n)$ that exists for each concept $\exists R^*.C$. By selecting always neighbors that are closer to the next d_i until d_n is reached, the infinite regeneration of $\exists R^*.C$ is avoided.

Formally, we start by assigning a natural number $\ell(d, \exists R^*.C)$ to each d and each $\exists R^*.C$ such that $\exists R^*.C$ is a subconcept of some $C' \in \theta(d)$ as follows.

- If $C \in \theta(d)$, then $\ell(d, \exists R^*.C) = 0$.
- If $\ell(d, \exists R^*.C) \neq n'$ for every $n' \leq n$, $\geqslant 1\, S.C' \in \theta(d)$, $\exists R^*.C$ is a subconcept of C', and there is some $d' \in \mathsf{neigh}_{\mathcal{I},\theta}(d, S)$ with $C' \in \theta(d')$ and $\ell(d', \exists R^*.C) = n$, then $\ell(d, \exists R^*.C) = n + 1$.

Since a sequence $d_0, \ldots, d_n$ leading to a d_n with $C \in \theta(d_n)$ exists for each $\exists R^*.C$, the function ℓ is well defined and assigns a finite number to each pair $(d, \exists R^*.C)$. Hence we can define a choice function ch such that, for each pair $(d, \geqslant 1\, S.C)$ where C has a subconcept of the form $\exists R^*.C'$, $\mathsf{ch}(d, \geqslant 1\, S.C) = \{d'\}$ for some d' with $\ell(d', \exists R^*.C') < \ell(d, \exists R^*.C')$. Clearly, each sequence of the form $(d_i, C_i) \rightsquigarrow \cdots \rightsquigarrow (d_j, C_j)$ with $d_i \neq d_j$ and $\exists R^*.C$ a subconcept of both C_i and C_j, contains $(d, \exists S.C') \rightsquigarrow (d, \geqslant 1\, S.C') \rightsquigarrow (d', C')$ for $\mathsf{ch}(d, \geqslant 1\, S.C') = \{d'\}$ and some C' containing $\exists R^*.C$ as a subconcept. Hence, $\ell(d_i, \exists R^*.C) > \ell(d_{i+1}, \exists R^*.C)$ whenever $\exists R^*.C$ is regenerated from d_i to d_{i+1}, and this can only occur finitely often. This shows that $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ is well-founded.

The second item is easier. It suffices to prove by induction on the structure of C that $C \in \theta(d)$ implies $d \models C$. U sing the well-foundedness of ch, the induction is strightforward. The claim follows from this and item 1 in Definition 3.2.7. □

Now we are ready to prove the canonical model property stated in Proposition 3.2.4, i.e., that for a given normal concept D and P2RPQ q, provided that D is in $\mathcal{ZOQ}$, in $\mathcal{ZOI}$, or nominal restricted, $D \not\models_{\mathsf{ind}} q$ implies that there is a canonical model of D which admits no match for q. We will show this separately for the three cases above, using a similar technique that unravels an arbitrary well-founded adorned pre-model of D into one that is forest-shaped.

In what follows, we denote by $t_\emptyset$ the *empty role type* that has $\neg P \in t_\emptyset$ for every $P \in \overline{\mathsf{N_R}}(D)$. We note that, in every pre-model $\langle \mathcal{I}, \theta \rangle$, for each pair $d, d' \in \Delta^{\mathcal{I}}$ and each $\geqslant n\, S.C$ in $\theta(d)$, $d' \in \mathsf{neigh}_{\mathcal{I},\theta}(d, S)$ implies that S is a safe role and hence $\theta(d, d') \neq t_\emptyset$.

Canonical model property of normal $\mathcal{ZOQ}$ concepts

Lemma 3.2.11 *Let D be a normal $\mathcal{ZOQ}$ concept. For every P2RPQ q, if $D \not\models_{\mathsf{ind}} q$, then there is a canonical model $\mathcal{I}'$ of D with $\mathcal{I}' \not\models q$.*

Proof. Assume $D \not\models_{\mathsf{ind}} q$. By item 1 of Lemma 3.2.10, there is some well-founded adorned pre-model $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ of D such that $\mathcal{I} \not\models q$. We show that we can unravel $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ into an adorned pre-model $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$ that is also well founded, such that $\mathcal{I}'$ is a canonical interpretation, and such that $\mathcal{I}' \not\models q$. By item 2 of Lemma 3.2.10, this implies that $\mathcal{I}'$ is a canonical model of D with $\mathcal{I}' \not\models q$.

To unravel $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ into $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$, we inductively build the domain $\Delta^{\mathcal{I}'}$ of $\mathcal{I}'$ as a forest and define a mapping $\xi : \Delta^{\mathcal{I}'} \to \Delta^{\mathcal{I}}$ that keeps track of the correspondence between nodes of the forest and elements of $\mathcal{I}$, while simultaneously defining the functions θ' and ch'. The interpretation of concepts and roles in $\mathcal{I}'$ will be defined after the inductive construction, using the mapping ξ.

When defining θ', we give explicitly the value of $\theta(d, d')$ only if (d, d') is a pair of the form (w, w), $(w, w \cdot i)$ or (w, c) with c a root. For all other pairs of elements (d, d') of $\Delta^{\mathcal{I}'}$, $\theta(d, d') = \theta(d', d) = t_\emptyset$ is the empty role type.

We let $\mathsf{R}(\mathcal{I}) = \{a^{\mathcal{I}} \mid a \in \mathsf{N_I}(D)\}$ and define $\mathsf{roots}(\mathcal{I}') = \{1, \dots, |\mathsf{R}(\mathcal{I})|\}$. Intuitively, to build the forest $\Delta^{\mathcal{I}'}$, we start with the set $\mathsf{roots}(\mathcal{I}')$ that will be the roots of $\mathcal{I}'$. Then we continue building trees from these roots. At each node w, the choice function indicates which successors w requires to satisfy the concepts of the form $\geqslant n\, S.C$. If some successor is a root c, then an S relation between w and c is created. Otherwise, a new node is added as a successor of w.

We are ready to start the inductive construction of $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$. For the base case, we let $\Delta^{\mathcal{I}'} := \mathsf{roots}(\mathcal{I}')$, and let $\xi : \mathsf{roots}(\mathcal{I}') \to \mathsf{R}(\mathcal{I})$ be an arbitrary bijection. Then we set, for each $c, c' \in \Delta^{\mathcal{I}'}$:

- $\theta'(c) = \theta(\xi(c))$,
- $\theta'(c, c') = \theta(\xi(c), \xi(c'))$, and
- for each $C_1 \sqcup C_2 \in \theta'(c)$, $\mathsf{ch}'(c, C_1 \sqcup C_2) = \mathsf{ch}(\xi(c), C_1 \sqcup C_2)$.

Choices for $\geqslant n\, S.C$ and $\leqslant n\, S.C$ concepts are defined in the induction step.

For the induction step, consider an $w \in \Delta^{\mathcal{I}'}$ of maximal length, and let $(\geqslant n_1\, S_1.C_1, e_1), \dots, (\geqslant n_m\, S_m.C_m, e_m)$ be all pairs of a formula $\geqslant n_i\, S_i.C_i \in \theta'(w)$ and an $e_i \in \mathsf{ch}(\xi(w), \geqslant n_i\, S_i.C_i)$. For each $1 \leq i \leq m$, we define:

$$\phi(e_i) \quad = \quad \begin{cases} u, & \text{if } e_i = \xi(u) \text{ with either } u = w \text{ or } u \in \mathsf{roots}(\mathcal{I}'), \\ w{\cdot}i & \text{otherwise.} \end{cases}$$

Then we set $\Delta^{\mathcal{I}'} := \Delta^{\mathcal{I}'} \cup \{\phi(e_1), \dots, \phi(e_m)\}$ and $\xi(w{\cdot}i) = e_i$ for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$. To extend θ', set for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$

- $\theta'(w{\cdot}i) = \theta(\xi(w{\cdot}i))$,
- $\theta'(w{\cdot}i, w{\cdot}i) = \theta(\xi(w{\cdot}i), \xi(w{\cdot}i))$,
- $\theta'(w, w{\cdot}i) = \theta(\xi(w), \xi(w{\cdot}i))$, and
- $\theta'(w{\cdot}i, u) = \theta(\xi(w{\cdot}i), \xi(u))$.

Finally, we extend the choice to concepts of the form $C \sqcup C'$ in $\theta'(w{\cdot}i)$ for the new domain elements $w{\cdot}i$, and to concepts $\geqslant n\, S.C$ and $\leqslant n\, S.C$ in $\theta'(w)$ for the previously existing w:

- for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$ and each $C \sqcup C' \in \theta'(w{\cdot}i)$, $\mathsf{ch}'(w{\cdot}i, C \sqcup C') = \mathsf{ch}(\xi(w{\cdot}i), C \sqcup C')$;
- for each $\geqslant n\, S.C \in \theta'(w)$, $\mathsf{ch}'(w, \geqslant n\, S.C) = \{\phi(e) \mid e \in \mathsf{ch}(\xi(w), \geqslant n\, S.C)\}$; and
- for each $\leqslant n\, S.C \in \theta'(w)$, $\mathsf{ch}'(w, \leqslant n\, S.C) = \{\phi(e) \mid e \in \mathsf{ch}(\xi(w), \leqslant n\, S.C) \cap \{e_1, \ldots, e_m\}\}$.

This concludes the inductive construction of $\Delta^{\mathcal{I}'}$, ξ, θ' and ch'. Now we define the interpretation function for $\mathcal{I}' = (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ using the mapping ξ:

- for each $a \in \mathsf{N_I}(D)$, $a^{\mathcal{I}'} := c$ for the $c \in \mathsf{roots}(\mathcal{I}')$ such that $\xi(c) = a^{\mathcal{I}}$,
- for each $A \in \mathsf{N_C}(D)$, $A^{\mathcal{I}'} := \{w \in \Delta^{\mathcal{I}'} \mid \xi(w) \in A^{\mathcal{I}}\}$, and
- for each $p \in \mathsf{N_R}(D)$, $p^{\mathcal{I}'} := \{(w, y) \in \Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'} \mid (\xi(w), \xi(y)) \in p^{\mathcal{I}}\}$.

Now we show that $\mathcal{I}'$ is a canonical interpretation and that $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$ is a well-founded adorned pre-model of D. For the former, first we observe that for every sequence $\phi(e_1), \ldots, \phi(e_m)$ above, m is bounded by $b_D = |Cl(D)| \cdot \max(\{n \mid \geqslant n\, S.C \in Cl(D)\} \cup \{0\})$. Hence each $w' \in \Delta^{\mathcal{I}'}$ is of the form $c{\cdot}w$ with $c \in \mathsf{roots}(\mathcal{I}')$ and $w \in \{1, \ldots, b_D\}^*$, the domain $\Delta^{\mathcal{I}'}$ of $\mathcal{I}'$ is a b_D-forest, and (1) of Definition 3.2.2 holds. Clearly, (2) $\mathsf{roots}(\Delta^{\mathcal{I}}) = \mathsf{roots}(\mathcal{I}') = \{a^{\mathcal{I}'} \mid a \in \mathsf{N_I}(D)\}$ holds. Furthermore, each $c{\cdot}w \in \Delta^{\mathcal{I}'}$ is reachable from a root c and $\mathcal{I}'$ is connected as required by (3). This is because each $w \cdot i$ in $\Delta^{\mathcal{I}'}$ satisfies $\theta(w, w \cdot i) \neq t_\emptyset$ and hence $(w, w \cdot i) \in P^{\mathcal{I}'}$ for some $P \in \overline{\mathsf{N_R}}(D)$. Next, for each pair $w, w' \in \Delta^{\mathcal{I}'}$ with $\theta(w, w') \neq t_\emptyset$ we have that either $w = w'$, $w = w \cdot i$, or $w' \in \mathsf{roots}(\Delta^{\mathcal{I}'})$, so (4) holds.

A simple inspection shows that $\langle \mathcal{I}', \theta' \rangle$ satisfies items 2 to 6 in Definition 3.2.7. For 7a, consider an arbitrary $w \in \Delta^{\mathcal{I}'}$, and a concept $\geqslant n\, S.C \in \theta(w)$. By the construction of $\mathcal{I}'$ and the definition of ch, there are at least n elements e_i such that $(\geqslant n\, S.C, e_i)$ is in the sequence above and $e_i \in \mathsf{neigh}_{\mathcal{I},\theta}(\xi(w), S)$. For each such e_i there is a domain element $\phi(e_i) \in \Delta^{\mathcal{I}'}$ with $C \in \theta(\phi(e_i)) = \theta(e_i)$. As $S \in \theta(\xi(w), \xi(\phi(e_i)))$ and $\theta(w, \phi(e_i)) = \theta(\xi(w), \xi(\phi(e_i)))$, we have $\phi(e_i) \in \mathsf{neigh}_{\mathcal{I}',\theta'}(w, S)$ for each e_i and 7a holds. For 7b, we consider an arbitrary $w \in \Delta^{\mathcal{I}'}$ and a concept $\leqslant n\, S.C \in \theta(w)$. We observe that if $\theta(w, w') \neq t_\emptyset$, then $w' = \phi(e_i)$ for some e_i in a sequence above (note that $\theta(w, w') = t_\emptyset$ for every other pair, including the pairs with w' a root). Hence, if $w' \in \mathsf{neigh}_{\mathcal{I}',\theta'}(w, S)$ for some S, then $w' = \phi(e_i)$ for some e_i. By the construction of $\mathcal{I}'$ and the definition of ch there are at most n such elements with $C \in \theta(w')$, and 7b follows. Hence $\langle \mathcal{I}', \theta' \rangle$ is a pre-model of D.

It is easy to verify that $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$ is an adorned pre-model of D. To see that it is well-founded, if a concept $\exists R^*.C_1$ is regenerated from w to w' in $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$, then $\exists R^*.C_1$ is also regenerated from $\xi(w)$ to $\xi(w')$ in $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$. As a consequence, well-foundedness of $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ implies well-foundedness of $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$. Finally, it is easy to observe that if $\mathcal{I}', \pi \models q$ for some π, then the composition π^* of π and ξ would be a match for q in $\mathcal{I}$, contradicting the assumption that $\mathcal{I} \not\models q$. □

Canonical model property of normal $\mathcal{ZOI}$ concepts

Lemma 3.2.12 *Let D be a normal $\mathcal{ZOI}$ concept. For every P2RPQ q, if $D \not\models_{\mathsf{ind}} q$, then there is a canonical model $\mathcal{I}'$ of D with $\mathcal{I}' \not\models q$.*

Proof. We proceed similarly, showing how a well-founded adorned pre-model $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ of D such that $\mathcal{I} \not\models q$ can be unraveled into a well founded adorned pre-model $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$ such that $\mathcal{I}'$ is a canonical interpretation and $\mathcal{I}' \not\models q$. We inductively build the domain $\Delta^{\mathcal{I}'}$ of $\mathcal{I}'$ as a forest and define a mapping $\xi : \Delta^{\mathcal{I}'} \to \Delta^{\mathcal{I}}$ that keeps track of the correspondence between nodes of the

forest and elements of $\mathcal{I}$, while simultaneously defining the functions θ' and ch'. When defining θ', we give explicitly the value of $\theta(d, d')$ only if (d, d') is a pair of the form (w, w), $(w, w \cdot i)$ or (w, c) with c a root. Now we implicitly assume that $\theta(d', d) = \mathsf{Inv}(\theta(d, d'))$ for the corresponding pairs d, d', and for all other pairs of elements (d, d') of $\Delta^{\mathcal{I}'}$, we have $\theta(d, d') = \theta(d', d) = t_\emptyset$.

The construction of $\Delta^{\mathcal{I}'}$ starts from the roots $\mathsf{roots}(\mathcal{I}') = \{1, \ldots, |\mathsf{R}(\mathcal{I})|\}$, where $\mathsf{R}(\mathcal{I}) = \{a^{\mathcal{I}} \mid a \in \mathsf{N_I}(D)\}$. Then we continue adding, to each node w, the successors that it requires to satisfy the concepts of the form $\geqslant n\, S.C$ as indicated by the choice function. If required, we may connect w to a root of the forest, or add a new node as a successor of w. Please note that since D is a $\mathcal{ZOI}$ concept, number restrictions in $Cl(D)$ are only of the forms $\geqslant 1\, S.C$ and $\leqslant 0\, S.{\sim}C$.

The base case is as in the proof of Lemma 3.2.11. That is, $\Delta^{\mathcal{I}'} := \mathsf{roots}(\mathcal{I}')$, and $\xi : \mathsf{roots}(\mathcal{I}') \to \mathsf{R}(\mathcal{I})$ is an arbitrary bijection. For each $c, c' \in \Delta^{\mathcal{I}'}$, we set:

- $\theta'(c) = \theta(\xi(c))$,
- $\theta'(c, c') = \theta(\xi(c), \xi(c'))$, and
- for each $C_1 \sqcup C_2 \in \theta'(c)$, $\mathsf{ch}'(c, C_1 \sqcup C_2) = \mathsf{ch}(\xi(c), C_1 \sqcup C_2)$.

For the induction step, consider an $w \in \Delta^{\mathcal{I}'}$ of maximal length, and let $(\geqslant n_1\, S_1.C_1, e_1)\, , \ldots ,$ $(\geqslant n_m\, S_m.C_m, e_m)$ be all pairs of a formula $\geqslant n_i\, S_i.C_i \in \theta'(w)$ and an $e_i \in \mathsf{ch}(\xi(w), \geqslant n_i\, S_i.C_i)$. For each $1 \leq i \leq m$, we define:

$$\phi(e_i) \quad = \quad \begin{cases} u, & \text{if } e_i = \xi(u) \text{ with either } u = w, u = w \cdot -1, \text{ or } u \in \mathsf{roots}(\mathcal{I}'), \\ w{\cdot}i & \text{otherwise.} \end{cases}$$

Then we set $\Delta^{\mathcal{I}'} := \Delta^{\mathcal{I}'} \cup \{\phi(e_1), \ldots, \phi(e_m)\}$ and $\xi(w{\cdot}i) = e_i$ for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$. To extend θ', set for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$

- $\theta'(w{\cdot}i) = \theta(\xi(w{\cdot}i))$,
- $\theta'(w{\cdot}i, w{\cdot}i) = \theta(\xi(w{\cdot}i), \xi(w{\cdot}i))$,
- $\theta'(w, w{\cdot}i) = \theta(\xi(w), \xi(w{\cdot}i))$, and
- $\theta'(w{\cdot}i, u) = \theta(\xi(w{\cdot}i), \xi(u))$ if $u \in \mathsf{roots}(\mathcal{I}')$.

Finally, we extend the choice to concepts of the form $C \sqcup C'$ in $\theta'(w{\cdot}i)$ for the new domain elements $w{\cdot}i$, and to concepts $\geqslant n\, S.C$ and $\leqslant n\, S.C$ in $\theta'(w)$ for the previously existing w:

- for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$ and each $C \sqcup C' \in \theta'(w{\cdot}i)$, $\mathsf{ch}'(w{\cdot}i, C \sqcup C') = \mathsf{ch}(\xi(w{\cdot}i), C \sqcup C')$;
- for each $\geqslant n\, S.C \in \theta'(w)$, $\mathsf{ch}'(w, \geqslant n\, S.C) = \{\phi(e) \mid e \in \mathsf{ch}(\xi(w), \geqslant n\, S.C)\}$; and
- for each $\leqslant n\, S.C \in \theta'(w)$, $\mathsf{ch}'(w, \leqslant n\, S.C) = \{\phi(e) \mid e \in \mathsf{ch}(\xi(w), \leqslant n\, S.C) \cap \{e_1, \ldots, e_m\}\}$.

This concludes the inductive construction of $\Delta^{\mathcal{I}'}$, ξ, θ' and ch'. To define the interpretation function for $\mathcal{I}' = (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ we use the mapping ξ:

- for each $a \in \mathsf{N_I}(D)$, $a^{\mathcal{I}'} := c$ for the (unique) $c \in \mathsf{roots}(\mathcal{I}')$ such that $\xi(c) = a^{\mathcal{I}}$,
- for each $A \in \mathsf{N_C}(D)$, $A^{\mathcal{I}'} := \{w \in \Delta^{\mathcal{I}'} \mid \xi(w) \in A^{\mathcal{I}}\}$, and
- for each $p \in \mathsf{N_R}(D)$, $p^{\mathcal{I}'} := \{(w, y) \in \Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'} \mid (\xi(w), \xi(y)) \in p^{\mathcal{I}}\}$.

To show that $\mathcal{I}'$ is a canonical interpretation and that $\langle \mathcal{I}', \theta' \rangle$ satisfies items 2 to 6 and 7a of Definition 3.2.7, one can proceed exactly as in the proof of Lemma 3.2.11. It is also easy to verify that item 7b holds at all nodes w that are not roots, since $\theta'(w, w') \neq t_\emptyset$ implies that $w' = \phi(e_i)$ for some e_i in a sequence selected by the choice function, and by definition there are at most n elements. If w is a root, then $\theta'(w, w') \neq t_\emptyset$ may also hold for other nodes (in particular, for nodes w' such that $\xi(w) \in \mathsf{ch}(\xi(w'), \geqslant n\, S'.C')$ for some $\geqslant n\, S'.C'$) and we need to show that 7b still holds. To prove this, assume towards a contradiction that there is some $u \in \mathsf{roots}(\mathcal{I}')$ such that $\leqslant n\, S.C \in \theta'(u)$ but $|\{w \in \mathsf{neigh}_{\mathcal{I}',\theta'}(u, S) \mid C \in \theta'(w)\}| > n$. Note that $\leqslant n\, S.C \in \theta'(u)$ implies $\leqslant n\, S.C \in \theta(\xi(u))$. Since D is a $\mathcal{ZOI}$ concept, we know that $n = 0$. Consider some $w \in \mathsf{neigh}_{\mathcal{I}',\theta'}(u, S)$ with $C \in \theta'(w)$. By construction of $\mathcal{I}$, $\xi(w) \in \mathsf{neigh}_{\mathcal{I},\theta}(\xi(u), S)$ and $C \in \theta(\xi(w))$. But this implies that $|\{d \in \mathsf{neigh}_{\mathcal{I},\theta}(\xi(u), S) \mid C \in \theta(d)\}| > 0$, which contradicts the fact that $\langle \mathcal{I}, \theta \rangle$ is a pre-model that satisfies 7b. Hence $\langle \mathcal{I}', \theta' \rangle$ satisfies 1 to 7b and it is a pre-model of D. The rest of the proof is as for Lemma 3.2.11: $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$ is a well-founded adorned pre-model of D, and $\mathcal{I} \not\models q$. □

Canonical model property of normal, nominal restricted $\mathcal{ZOIQ}$ concepts

Lemma 3.2.13 *Let D be a normal nominal restricted $\mathcal{ZOIQ}$ concept. For every P2RPQ q, if $D \not\models_{\mathsf{ind}} q$, then there is a canonical model $\mathcal{I}'$ of D with $\mathcal{I}' \not\models q$.*

Proof. We proceed similarly, showing how a well-founded adorned pre-model $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ of D with $\mathcal{I} \not\models q$ can be unraveled into a well founded adorned pre-model $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$ such that $\mathcal{I}'$ is a canonical interpretation and $\mathcal{I}' \not\models q$. We inductively build the domain $\Delta^{\mathcal{I}'}$ of $\mathcal{I}'$ as a forest and define a mapping $\xi : \Delta^{\mathcal{I}'} \to \Delta^{\mathcal{I}}$ that keeps track of the correspondence between nodes of the forest and elements of $\mathcal{I}$, while simultaneously defining the functions θ' and ch'. When defining θ', we give explicitly the value of $\theta(d, d')$ only if both d and d' are roots, or if (d, d') is a pair of the form (w, w) or $(w, w \cdot i)$. We implicitly assume that $\theta(d', d) = \mathsf{Inv}(\theta(d, d'))$ for the corresponding pairs d, d', and for all other pairs of elements (d, d') of $\Delta^{\mathcal{I}'}$, we have $\theta(d, d') = \theta(d', d) = t_\emptyset$.

As above, we start the construction of $\Delta^{\mathcal{I}'}$ with the roots. Let $\mathsf{roots}(\mathcal{I}') = \{1, \ldots, |\mathsf{R}(\mathcal{I})|\}$, where $\mathsf{R}(\mathcal{I}) = \{a^{\mathcal{I}} \mid a \in \mathsf{N_I}(D)\}$, and let $\xi : \mathsf{roots}(\mathcal{I}') \to \mathsf{R}(\mathcal{I})$ be an arbitrary bijection. Then we let $\Delta^{\mathcal{I}'} := \mathsf{roots}(\mathcal{I}')$ and set, for each $c, c' \in \Delta^{\mathcal{I}'}$:

- $\theta'(c) = \theta(\xi(c))$,
- $\theta'(c, c') = \theta(\xi(c), \xi(c'))$, and
- for each $C_1 \sqcup C_2 \in \theta'(c)$, $\mathsf{ch}'(c, C_1 \sqcup C_2) = \mathsf{ch}(\xi(c), C_1 \sqcup C_2)$.

In the induction step we add to each node w of maximal length the successors w' that it requires to satisfy the concepts of the form $\geqslant n\, S.C$ as indicated by the choice function. The difference in this case is that, if $\xi(w')$ is a root c in the forest, we generate a new w' that is a copy of $\xi(w')$ but does not satisfy any nominal concept.

In what follows, for a concept type $T \in \mathsf{typesC}(D)$, we let $\mathsf{nr}(T)$ denote the set of concepts obtained from T by making false concepts containing nominals as follows:

$$\mathsf{nr}(T) = \big(T \setminus \{\{a\}, \exists p.\{a\}, \geqslant 1\, p.\{a\} \in Cl_{\mathsf{C}}(D)\}\big) \cup \{\neg\{a\} \mid \{a\} \in Cl_{\mathsf{C}}(D)\}$$

By setting $\theta(w')$ to $\mathsf{nr}(T)$ rather than T for each newly added w', we avoid duplicating in $\mathcal{I}'$ the relations between $\xi(w)$ and $\xi(w')$ in case the latter is a node in $\mathsf{R}(\mathcal{I})$, which could cause a violation of a number restriction. We show below that, since D is nominal restricted, $\mathsf{nr}(T)$ is also a concept type in $\mathsf{typesC}(D)$, and the unraveled $\langle \mathcal{I}', \theta' \rangle$ is a pre-model of D.

Now we proceed with the inductive step. Consider an $w \in \Delta^{\mathcal{I}'}$ of maximal length, and let $(\geqslant n_1\, S_1.C_1, e_1)\ , \ldots ,\ (\geqslant n_m\, S_m.C_m, e_m)$ be all pairs of a formula $\geqslant n_i\, S_i.C_i \in \theta'(w)$ and an $e_i \in \mathsf{ch}(\xi(w), \geqslant n_i\, S_i.C_i)$. For each $1 \leq i \leq m$, we define:

$$\phi(e_i) \quad = \quad \begin{cases} u, & \text{if } e_i = \xi(u) \text{ with either } u = w \text{ or } u = w \cdot -1, \\ w{\cdot}i & \text{otherwise.} \end{cases}$$

Then we set $\Delta^{\mathcal{I}'} := \Delta^{\mathcal{I}'} \cup \{\phi(e_1), \ldots, \phi(e_m)\}$ and $\xi(w{\cdot}i) = e_i$ for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$. To extend θ', set for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$

- $\theta'(w{\cdot}i) = \mathsf{nr}(\theta(\xi(w{\cdot}i)))$,
- $\theta'(w{\cdot}i, w{\cdot}i) = \theta(\xi(w{\cdot}i), \xi(w{\cdot}i))$,
- $\theta'(w, w{\cdot}i) = \theta(\xi(w), \xi(w{\cdot}i))$, and
- $\theta'(w{\cdot}i, u) = \theta(\xi(w{\cdot}i), \xi(u))$ if $u \in \mathsf{roots}(\mathcal{I}')$.

Finally, we extend the choice to concepts of the form $C \sqcup C'$ in $\theta'(w{\cdot}i)$ for the new domain elements $w{\cdot}i$, and to concepts $\geqslant n\, S.C$ and $\leqslant n\, S.C$ in $\theta'(w)$ for the previously existing w:

- for each $w{\cdot}i \in \Delta^{\mathcal{I}'}$ and each $C \sqcup C' \in \theta'(w{\cdot}i)$, $\mathsf{ch}'(w{\cdot}i, C \sqcup C') = \mathsf{ch}(\xi(w{\cdot}i), C \sqcup C')$;
- for each $\geqslant n\, S.C \in \theta'(w)$, $\mathsf{ch}'(w, \geqslant n\, S.C) = \{\phi(e) \mid e \in \mathsf{ch}(\xi(w), \geqslant n\, S.C)\}$; and
- for each $\leqslant n\, S.C \in \theta'(w)$, $\mathsf{ch}'(w, \leqslant n\, S.C) = \{\phi(e) \mid e \in \mathsf{ch}(\xi(w), \leqslant n\, S.C) \cap \{e_1, \ldots, e_m\}\}$.

This concludes the inductive construction of $\Delta^{\mathcal{I}'}$, ξ, θ' and ch'. As before, we use the mapping ξ to define the interpretation function for $\mathcal{I}' = (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$:

- for each $a \in \mathsf{N_I}(D)$, $a^{\mathcal{I}'} := c$ for the $c \in \mathsf{roots}(\mathcal{I}')$ such that $\xi(c) = a^{\mathcal{I}}$,
- for each $A \in \mathsf{N_C}(D)$, $A^{\mathcal{I}'} := \{w \in \Delta^{\mathcal{I}'} \mid \xi(w) \in A^{\mathcal{I}}\}$, and
- for each $p \in \mathsf{N_R}(D)$, $p^{\mathcal{I}'} := \{(w, y) \in \Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'} \mid (\xi(w), \xi(y)) \in p^{\mathcal{I}}\}$.

To show that $\mathcal{I}'$ is a canonical interpretation we can proceed exactly as in the proofs of Lemmas 3.2.11 and 3.2.12.

To show that $\langle \mathcal{I}', \theta' \rangle$ is a pre-model, we first show that since D is nominal restricted, $\mathsf{nr}(T)$ is a concept type in $\mathsf{typesC}(D)$. Recall that D is of the form $C' \sqcap \forall R^*.(C_1 \sqcap \ldots \sqcap C_n)$, where C' and R are nominal free, and each C_i is of the form $\neg\{a\} \sqcup A$, $\neg\{a\} \sqcup \exists p.\{b\}$, or $\neg\{a\} \sqcup \neg\{b\}$ (cf. Definition 3.1.3). Let $C_\sqcap$ denote a conjunction of concepts C_i of the listed forms, and let R_1, R_2 and S denote nominal-free roles. Nominals can only occur in T in concepts of the following forms, which may be part of conjunctions and disjunctions:

$$\forall R_1.C_\sqcap \qquad \forall R_1.\forall R_2.C_\sqcap \qquad \leqslant 0\, S.{\sim}C_\sqcap \qquad C_\sqcap \qquad \neg\{a\} \qquad \exists p.\{b\} \qquad \geqslant 1\, p.\{b\}$$

It is not hard to see that if we drop all concepts of the last two forms, and add $\neg\{a\}$ for every $a \in \mathsf{N_I}(a)$, the resulting $\mathsf{nr}(T)$ is still closed under all rules of Table 3.3. In particular, each concept C_i as above has a disjunct of the form $\neg\{a\}$ that is in $\mathsf{nr}(T)$, hence $\mathsf{nr}(T)$ contains C_i and at least one of its disjuncts. Also, each conjunction $C_\sqcap$ is in $\mathsf{nr}(T)$, and so are all its conjuncts. Hence, the closure properties of T are preserved in $\mathsf{nr}(T)$, and $\mathsf{nr}(T)$ is a concept type in $\mathsf{typesC}(D)$. Observe also that, for every $T \in \mathsf{typesC}(D)$ and for every $C \in \mathit{Cl}_\mathsf{C}(D)$ that is nominal free, $C \in \mathsf{nr}(T)$ iff $C \in T$.

Now it is not hard to prove that $\langle \mathcal{I}', \theta' \rangle$ satisfies items 1 to 7b of Definition 3.2.7. Items 1 and 2 are trivial. To see 3, we observe that each node w that is not a root has $\neg\{a\} \in \theta(w)$ for every $a \in \mathsf{N_I}(D)$, hence the root c with $a^{\mathcal{I}} = c$ is the only node in $\Delta^{\mathcal{I}'}$ with $\{a\} \in \theta(c)$. Items 4 and 5 hold by definition of $\mathcal{I}$, and 6 holds in $\langle \mathcal{I}', \theta' \rangle$ because it holds in $\langle \mathcal{I}, \theta \rangle$. Items 7a and 7b hold for every $\geqslant n\, S.C$ where C is nominal free, because the construction of $\Delta^{\mathcal{I}'}$ ensures that every node w has enough neighbors of the required type to satisfy each $\geqslant n\, S.C$ restriction, and no more neighbors of any type than the corresponding $\xi(w)$ has in $\mathcal{I}$, which satisfies the $\leqslant n\, S.C$ restrictions. As C is nominal free, it is not affected by the restriction of any type T to $\mathsf{nr}(T)$. For the number restrictions $\geqslant n\, S.C$ where C contains nominals, we distinguish the case of the roots and of all other nodes. The former case is similar to the nominal free one, and 7a and 7b hold by the construction of $\Delta^{\mathcal{I}'}$. Note that the 'neighborhood' of a root node c in $\mathcal{I}'$ comprises only root nodes and the successors of c, and it is identical to the neighborhood of $\xi(c)$ in $\mathcal{I}$. If w is not a root node and there is some $\geqslant n\, S.C \in \theta(w)$ that is not nominal free, then we see that $\geqslant n\, S.C$ must be of the form $\leqslant 0\, S.{\sim}C_{\sqcap}$. Note that the only concepts of the form $\geqslant n\, S.C$ in $Cl_{\mathsf{C}}(D)$ that contain some nominal are $\geqslant 1\, p.\{b\}$, which do not occur in any type associated to a non-root node. For each $\leqslant 0\, S.{\sim}C_{\sqcap} \in \theta(w)$, we know that the types of all non-root neighbors of w contain $C_{\sqcap}$, hence 7b is satisfied. The only non-root nodes that have root neighbors are the level one nodes, and for them 7b holds as well. With this we finish proving that $\langle \mathcal{I}', \theta' \rangle$ is a pre-model of D.

The rest of the proof is as for Lemmas 3.2.11 and 3.2.12: $\langle \mathcal{I}', \theta', \mathsf{ch}' \rangle$ is a well-founded adorned pre-model of D, and $\mathcal{I} \not\models q$. □

Lemmas 3.2.11, 3.2.12, and 3.2.13 conclude the proof of Proposition 3.2.4.

3.3 Satisfiability via Automata

Since the domain of a canonical model is a forest, we can represent it as a labeled forest in a straightforward way. This allows us to decide the existence of a canonical model using automata on infinite forests.

3.3.1 Representing Canonical Models as Forests

Recall that $\mathsf{N_{CI}}(D)$ denotes the set of atomic concepts B (i.e., B is a concept name $A \in \mathsf{N_C}$ or a nominal $\{a\}$ with $a \in \mathsf{N_I}$) that occur in D, while $\overline{\mathsf{N_R}}(D)$ is the set of atomic roles P (i.e., P is a role name $p \in \mathsf{N_R}$ or an inverse p^-) such that P or $\mathsf{Inv}(P)$ occurs in D.

To represent a canonical model $\mathcal{I}$ of D as a labeled forest, we label each individual w with the set of atomic concepts B such that $w \in B^{\mathcal{I}}$. We also add an atomic role P to the label of w whenever $(w, w') \in P^{\mathcal{I}}$ and w' is not a root. As in [BLMV08], we use special symbols to witness the existence of relations between a non-root node and individual root nodes. More specifically, a special symbol $\uparrow_a^P$ is in the label of w whenever $(w, a^{\mathcal{I}}) \in P^{\mathcal{I}}$ for some atomic role P. To represent loops $(w, w) \in p^{\mathcal{I}}$ for a role name p, we use special labels p_{Self}.

Definition 3.3.1 (Forest encoding) *For a concept D, the alphabet Σ_D is defined as follows:*

$$\begin{aligned} \Sigma_D &= 2^{\Theta(D)}, \text{ where} \\ \Theta(D) &= \mathsf{N_{CI}}(D) \cup \overline{\mathsf{N_R}}(D) \cup \{p_{\mathsf{Self}} \mid p \in \mathsf{N_R}(D)\} \cup \{\uparrow_a^P \mid P \in \overline{\mathsf{N_R}}(D) \text{ and } a \in \mathsf{N_I}(D)\}. \end{aligned}$$

The forest encoding *of a canonical model $\mathcal{I}$ of D is the Σ_D-labeled forest $\mathbf{F}_{\mathcal{I}} = \langle \Delta^{\mathcal{I}}, L^{\mathcal{I}} \rangle$ such*

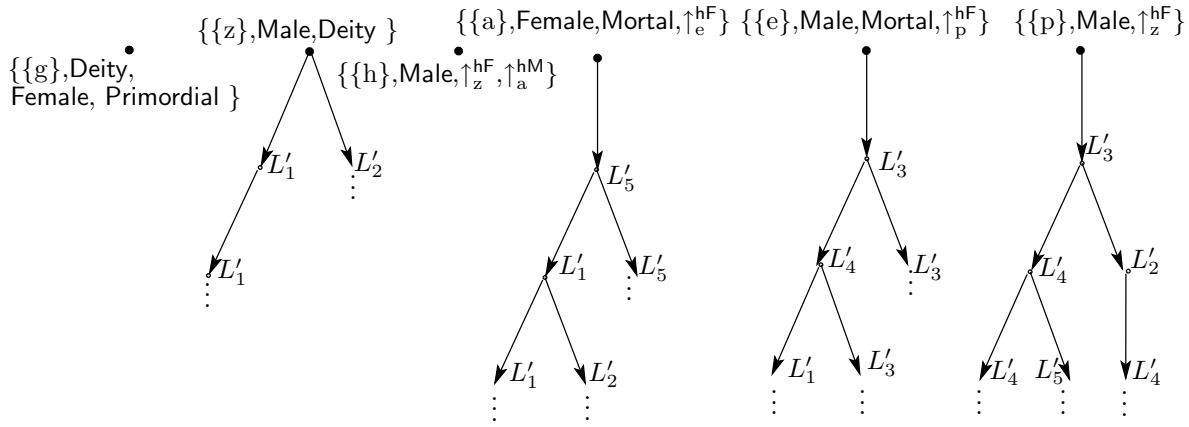

Figure 3.3: The forest encoding of the canonical model in Figure 3.2

that for each $w \in \Delta^{\mathcal{I}}$*:*

$$\begin{aligned} L^{\mathcal{I}}(w) = & \{B \in \Theta(D) \mid w \in B^{\mathcal{I}}\} \cup \\ & \{p_{\mathsf{Self}} \in \Theta(D) \mid p \in \mathsf{N_R}(D) \text{ and } (w,w) \in p^{\mathcal{I}}\} \cup \\ & \{P \in \Theta(D) \mid \ (w',w) \in P^{\mathcal{I}} \text{ and } w \in \mathsf{succ}(w')\} \cup \\ & \{\uparrow^P_a \in \Theta(D) \mid P \in \overline{\mathsf{N_R}}(D), (w, a^{\mathcal{I}}) \in P^{\mathcal{I}}, \text{ and } a \in \mathsf{N_I}(D)\}. \end{aligned}$$

Example 3.3.2 *Figure 3.3 depicts the forest encoding of the model* $\mathcal{I}_g$ *of* $\mathcal{K}_g$ *described in Example 3.2.3. For readability, in the labels we use only the initial letter of each individual name, and we use* hF *and* hM *in place of* hasFather *and* hasMother*, respectively. The labels of the level one nodes are given explicitly in the figure, while for the other nodes we use the labels* $L'_1 = L_1 \cup \{\mathsf{hF}, \uparrow^{\mathsf{hM}}_{\mathrm{g}}\}$*,* $L'_2 = L_2 \cup \{\mathsf{hM}, \uparrow^{\mathsf{hM}}_{\mathrm{g}}\}$*,* $L'_3 = L_3 \cup \{\mathsf{hM}\}$*,* $L'_4 = L_4 \cup \{\mathsf{hF}\}$*, and* $L'_5 = L_5 \cup \{\mathsf{hM}\}$*.*

We can see any Σ_D-labeled forest as a representation of a canonical model, provided that the individual names occur in the label of only one root. Informally, the domain of this interpretation are the roots labeled with some individual, and the nodes that are reachable from any such root through a sequence of roles in D. The extensions of individuals, concepts and roles are determined by the node labels.

Definition 3.3.3 *Given a* Σ_D*-labeled forest* $\mathbf{F} = \langle F, L\rangle$*, we call a root node* $c \in \mathsf{roots}(F)$ *an* individual node *if* $a \in L(c)$ *for some* $a \in \mathsf{N_I}(D)$*, and we call* c *an* a-node *if we want to make the precise a explicit. We say that* $\mathbf{F}$ *is* individual unique *if for each* $a \in \mathsf{N_I}(D)$ *there is exactly one* a*-node.*

An individual unique Σ_D *labeled forest* $\mathbf{F} = \langle F, L\rangle$*, called also a* D*-forest,* represents *the interpretation* $\mathcal{I}_{\mathbf{F}}$ *defined as follows. For each role name* $p \in \mathsf{N_R}(D)$*, let:*

$$\begin{aligned} \mathcal{R}_p \ = \ & \{(w, w{\cdot}i) \mid p \in L(w{\cdot}i)\} \cup \{(w{\cdot}i, w) \mid p^- \in L(w{\cdot}i)\} \cup \\ & \{(w,w) \mid p_{\mathsf{Self}} \in L(w)\} \cup \\ & \{(w,c) \mid \uparrow^p_a \in L(w) \text{ and } \{a\} \in L(c)\} \cup \\ & \{(c,w) \mid \uparrow^{p^-}_a \in L(w) \text{ and } \{a\} \in L(c)\} \end{aligned}$$

Let $\mathsf{i\text{-}roots}(F)$ *be the set of individual nodes in* F*. Then for each* $c \in \mathsf{i\text{-}roots}(F)$*, we let*

$$\mathbf{D}_c = \{\ w \mid (c,w) \in \bigcup_{p \in \mathsf{N_R}(D)} \big(\mathcal{R}_p \cup (\mathcal{R}_p)^-\big)^*\ \},$$

where $(\mathcal{R}_p)^-$ *denotes the inverse of relation* $\mathcal{R}_p$*. The interpretation* $\mathcal{I}_\mathbf{F} = (\Delta^{\mathcal{I}_\mathbf{F}}, \cdot^{\mathcal{I}_\mathbf{F}})$ *is defined as:*

$$\begin{array}{rcll} \Delta^{\mathcal{I}_\mathbf{F}} & = & \bigcup_{c \in \mathsf{i\text{-}roots}(F)} \mathbf{D}_c, & \\ a^{\mathcal{I}_\mathbf{F}} & = & c \in \mathsf{i\text{-}roots}(F) \text{ such that } \{a\} \in L(c), & \text{for each } a \in \mathsf{N_I}, \\ A^{\mathcal{I}_\mathbf{F}} & = & \Delta^{\mathcal{I}_\mathbf{F}} \cap \{\ w \mid A \in L(w)\ \}, & \text{for each concept name } A \in \mathsf{N_C}(D), \text{ and} \\ p^{\mathcal{I}_\mathbf{F}} & = & (\Delta^{\mathcal{I}_\mathbf{F}} \times \Delta^{\mathcal{I}_\mathbf{F}}) \cap \mathcal{R}_p, & \text{for each role name } p \in \mathsf{N_R}(D). \end{array}$$

Note that the forest encoding $\mathbf{F}_\mathcal{I}$ of each canonical $\mathcal{I}$ represents $\mathcal{I}$, and that the interpretation $\mathcal{I}_\mathbf{F}$ is always canonical.

Now we introduce Fully Enriched Automata, which can be used to decide whether there exists a labeled forest that represents a canonical model of D.

3.3.2 Fully Enriched Automata on Infinite Forests

Alternating automata are a powerful tool for reasoning in modal and program logics [MS87, EJ91]. *Two-way* alternating automata were introduced in [Var98] to better handle logics that have 'backward' operators, like inverse roles. They may move up on the input tree or stay at the current position, in contrast to one-way automata that navigate trees in a strictly top-down manner, moving always to the successors of the current node. *Graded transitions* allow the automaton to move up to n or to all but n successors of the current node on the input tree, for any n; this facilitates counting, and makes them convenient for logics that support number restrictions [KSV02]. Finally, with *root transitions*, FEAs can jump to some or all roots of a forest; this facilitates handling connections that may exist between any pair of a node and a root of the tree due to nominals [BLMV08].

We recall the definition of *fully enriched automata (FEAs)* from [BLMV08].

Definition 3.3.4 (FEA) *For a set* W*, let* $\mathcal{B}(W)$ *be the set of Boolean formulas constructible with atoms* $W \cup \{\boldsymbol{true}, \boldsymbol{false}\}$ *and* $\wedge$, $\vee$*. We say that* $V \subseteq W$ makes $\varphi \in \mathcal{B}(W)$ true*, if the formula obtained by assigning* $\boldsymbol{true}$ *to all* $v \in V$ *and* $\boldsymbol{false}$ *to all* $w \in W \setminus V$ *evaluates to true. For* $b > 0$*, let* $\mathbf{D}_b = \{-1, \varepsilon\} \cup \{\langle 0\rangle, \ldots, \langle b\rangle\} \cup \{[0], \ldots, [b]\} \cup \{\langle \mathsf{root}\rangle, [\mathsf{root}]\}$.

A fully enriched automaton (FEA) with parity index n *is a tuple* $\mathbf{A} = \langle \Sigma, b, Q, \delta, q_0, G\rangle$*, where*

- Σ *is a finite input alphabet,*
- $b > 0$ *is a counting bound,*
- Q *is a finite set of states,*
- $\delta : Q \times \Sigma \to \mathcal{B}(\mathbf{D}_b \times Q)$ *is a transition function,*
- $q_0 \in Q$ *is an initial state, and*
- $G = (G_1, \ldots, G_n)$ *with* $G_1 \subseteq G_2 \subseteq \cdots \subseteq G_n = Q$ *is a* parity acceptance condition*, and* n *is the* parity index *of* $\mathbf{A}$.

The transition function δ maps a state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a positive Boolean formula φ over the atoms in $\mathbf{D}_b \times Q$. Intuitively, if $\delta(q, \sigma) = \varphi$, then each atom (d, q') in φ corresponds to a new copy of the automaton that moves in the direction described by d and to state q'.

The direction $d \in \mathbf{D}_b$ may be any of the following:

-1 indicates moving up to the predecessor of the current node,
ε indicates staying at the current node,
$\langle i\rangle$, indicates moving to $i{+}1$ successors of the current node, for $0 \leq i \leq b$,
$[i]$ indicates moving to all but i successors, for $0 \leq i \leq b$,

$\langle \mathsf{root} \rangle$ indicates moving to some root of the forest, and
$[\mathsf{root}]$ indicates moving to all roots.

For example, let $b = 2$ and $\delta(q_1, \sigma) = (\varepsilon, q_2) \wedge ([2], q_3) \vee (\langle \mathsf{root} \rangle, q_1) \wedge (-1, q_3)$. If $\mathbf{A}$ is in the state q_1 and reads the node w labeled with σ, it proceeds by sending off either

(i) one copy in the state q_2 that stays at the node w (i.e., $w{\cdot}\varepsilon$), and a copy in the state q_3 to all but 2 of the successors of w; or

(ii) one copy in the state q_1 to a root of F, and one copy in the state q_3 to the predecessor $w{\cdot} - 1$ of w.

Run of a FEA

The acceptance of a forest $\mathbf{F} = \langle F, L \rangle$ by a FEA $\mathbf{A} = \langle \Sigma, b, Q, \delta, q_0, G \rangle$ can be formalized through the notion of *run*, which is a tree labeled by elements of $F{\times}Q$. Intuitively, the situation in which (a copy of) $\mathbf{A}$ is in a state $q \in Q$ and reading a node w of F is described by a node t in the run that is labeled (w, q). For a move of $\mathbf{A}$ to nodes $w_1, \ldots, w_n$ of F (which can include w, some of its successors, its predecessor, and some roots of F) and states $q_1, \ldots, q_n$, there will be one child t_i of t labeled (w_i, q_i) for each i. In the example above, where $\mathbf{A}$ is in state q_1 and reading the node w, let us suppose that $\mathbf{A}$ proceeds as described in item (ii). That is, it simultaneously moves to a root c of F in the state q_1, and to the predecessor of w in the the state q_3. In a run, this is described by a node t labeled (w, q_1) with two children t'_1 and t'_2, respectively labeled (c, q_1) and $(w{\cdot} - 1, q_3)$.

Now we formally define runs as $F{\times}Q$-labeled trees. A run always starts at a root of F and in the initial state q_0, and each node satisfies some local conditions that ensure the satisfaction of the transition function.

Definition 3.3.5 (run, acceptance, non-emptiness) *A* run *of a FEA* $\mathbf{A} = \langle \Sigma, b, Q, \delta, q_0, G \rangle$ *over a labeled forest* $\langle F, L \rangle$ *is a* $F{\times}Q$*-labeled tree* $\langle T_r, r \rangle$ *such that*

1. $r(\mathsf{root}(T_r)) = (c, q_0)$ *for some* $c \in \mathsf{roots}(F)$*, and*

2. *for every* $t \in T_r$ *with* $r(t) = (w, q)$ *there is some* $W \subseteq \mathbf{D}_b {\times} Q$ *such that* W *makes* $\delta(q, L(w))$ *true and, for all* $(d, q') \in W$*:*

   - *if* $d \in \{-1, \varepsilon\}$*, then* $w{\cdot}d$ *is a node in* F *and there is some* $j \in \mathbb{N}$ *such that* $t{\cdot}j \in T_r$ *and* $r(t{\cdot}j) = (w{\cdot}d, q')$*;*
   - *if* $d = \langle n \rangle$*, then there is some* $M \subseteq \mathsf{succ}(w)$ *with* $|M| > n$ *such that, for each* $z \in M$*, there is some* $j \in \mathbb{N}$ *such that* $t{\cdot}j \in T_r$ *and* $r(t{\cdot}j) = (z, q')$*;*
   - *if* $d = [n]$*, then there is some* $M \subseteq \mathsf{succ}(w)$ *with* $|M| \leq n$ *such that, for each* $z \in \mathsf{succ}(w) \setminus M$*, there is some* $j \in \mathbb{N}$ *such that* $t{\cdot}j \in T_r$ *and* $r(t{\cdot}j) = (z, q')$*;*
   - *if* $d = \langle \mathsf{root} \rangle$*, then there is some* $c \in \mathsf{roots}(F)$ *and* $j \in \mathbb{N}$ *such that* $t{\cdot}j \in T_r$ *and* $r(t{\cdot}j) = (c, q')$*;*
   - *if* $d = [\mathsf{root}]$*, then for each* $c \in \mathsf{roots}(F)$ *there is some* $j \in \mathbb{N}$ *such that* $t{\cdot}j \in T_r$ *and* $r(t{\cdot}j) = (c, q')$*.*

The run $\langle T_r, r \rangle$ *is* accepting*, if for each infinite path* P *of* T_r *there is an* even i *such that* $\mathsf{Inf}(\langle P, r \rangle) \cap G_i \neq \emptyset$ *and* $\mathsf{Inf}(\langle P, r \rangle) \cap G_{i-1} = \emptyset$*, where* $\mathsf{Inf}(\langle P, r \rangle)$ *is the set of all states* $q \in Q$ *such that* $\{t \in P \mid \exists w. r(t) = (w, q)\}$ *is infinite. That is,* $\langle T_r, r \rangle$ *is accepting if for each infinite path* P *the least* i *such that a state in* G_i *occurs infinitely often in* P *is even.*

A FEA $\mathbf{A}$ accepts *a labeled forest* $\langle F, L \rangle$ *if there exists an accepting run of* $\mathbf{A}$ *over* $\langle F, L \rangle$*. The set of all forests accepted by* $\mathbf{A}$ *is called the* language of $\mathbf{A}$ *and is denoted* $\mathscr{L}(\mathbf{A})$*. The* non-emptiness problem *is to decide whether* $\mathscr{L}(\mathbf{A}) \neq \emptyset$ *for a given FEA* $\mathbf{A}$*.*

$$
\begin{array}{rcl}
Q_{\mathsf{N_I}} & = & \{(\!|a \Rightarrow s|\!), (\!|a \Rightarrow \neg s|\!) \mid a \in \mathsf{N_I}(D), s \in \Theta(D)\} \cup \\
 & & \{(\!|a \Rightarrow C|\!), (\!|a \Rightarrow {\sim}C|\!) \mid a \in \mathsf{N_I}(D), C \in \mathit{Cl}_{\mathsf{C}}(D)\} \\
Q_{\mathsf{Self}} & = & \{S_{\mathsf{Self}} \mid S \in \mathit{Cl}_{\mathsf{R}}(D)\} \\
Q_{\uparrow} & = & \{\uparrow_a^S \mid S \in \mathit{Cl}_{\mathsf{R}}(D), a \in \mathsf{N_I}(D)\} \\
Q_{\mathsf{roots}} & = & \{(\!|\langle \mathsf{root} \rangle^k, S', C'|\!), (\!|[\mathsf{root}]^k, S', C'|\!), \mid 0 \leq k \leq |\mathsf{N_I}(D)|, S \in \mathbf{S}, C \in \mathbf{C}\} \\
Q_{\mathsf{Nom}} & = & \{(\!|\vee, \neg a, \neg b|\!) \mid a, b \in \mathsf{N_I}(D), a \neq b\} \\
Q_{\mathsf{bin}} & = & \{(\!|\circ, S, C|\!) \mid \circ \in \{\wedge, \vee\}, S \in \mathbf{S}, C \in \mathbf{C}\} \cup \\
 & & \{(\!|\circ, \alpha, C|\!) \mid \circ \in \{\wedge, \vee\}, \alpha = \{a\} \text{ or } \alpha = \neg\{a\}, a \in \mathsf{N_I}(D), C \in \mathbf{C}\}
\end{array}
$$

Table 3.5: State set $Q_D = \{q_D^0\} \cup \mathit{Cl}_{\mathsf{C}}(D) \cup \mathit{Cl}_{\mathsf{R}}(D) \cup \{s, \neg s \mid s \in \Theta(D)\} \cup Q_{\mathsf{N_I}} \cup Q_{\mathsf{Self}} \cup Q_{\uparrow} \cup Q_{\mathsf{Nom}} \cup Q_{\mathsf{roots}} \cup Q_{\mathsf{bin}}$. Here, $\mathbf{S}$ denotes the set of all simple roles S, ${\sim}S$ and $\mathbf{C}$ the set of all concepts C, ${\sim}C$ such that $\geqslant n\, S.C \in \mathit{Cl}_{\mathsf{C}}(D)$.

The following bound for deciding non-emptiness for a given FEA is given in [BLMV08]:

Theorem 3.3.6 ([BLMV08]) *The non-emptiness problem for a FEA $\mathbf{A} = \langle \Sigma, b, Q, \delta, q_0, G \rangle$ where F has parity index k can be solved in time $(b+2)^{\mathcal{O}(|Q|^3 \cdot k^2 \cdot \log k \cdot \log b^2)}$.*

3.3.3 Reducing Concept Satisfiability to FEA emptiness

We are ready to define, given a concept D, a FEA $\mathbf{A}_D$ whose language is empty iff D has a canonical model. $\mathbf{A}_D$ accepts each forest that represents a canonical model of $\mathbf{A}_D$. Since FEAs can not ensure individual uniqueness, it also accepts forests that are not individual unique, and hence they do not represent a canonical model. Observe that, for example, if a FEA accepts some forest $\mathbf{F}$, then it also accepts any other forest obtained by adding to $\mathbf{F}$ redundant copies of any of its trees. To guarantee the correctness of our reduction, the construction of $\mathbf{A}_D$ ensures that for every individual a and every pair c, c' of a-nodes, the trees rooted at c and c' are indistinguishable by its transition function. This is enough for our purposes, because it allows us to show that if $\mathbf{A}_D$ accepts some forest, then it also accepts a individual unique one which represents a model of D.

Given D, the FEA $\mathbf{A}_D$ is defined next. Since the construction is rather involved, we give many informal explanations along the way. In them, we say that a node w satisfies a concept C if w pre-satisfies C, when seen as a domain element in the represented interpretation (if it is defined). Similarly, we say that w is an R-successor of w' or that R holds between w and w', if the pair (w, w') is in the extension of R in the represented interpretation.

Definition 3.3.7 *For a given $\mathcal{ZOIQ}$ concept D, the automaton $\mathbf{A}_D = \langle \Sigma_D, b_D, Q_D, \delta_D, q_D^0, G_D \rangle$ for D is defined as follows:*

- $\Sigma_D = 2^{\Theta(D)}$ *as in Definition 3.3.1.*
- *The counting bound is* $b_D = \max(\{n \mid \geqslant n\, S.C \in \mathit{Cl}_{\mathsf{C}}(D)\} \cup \{0\})$.
- *The set of states Q_D is shown in Table 3.5.*

  Intuitively, the 'basic' states of $\mathbf{A}_D$ correspond to the concepts in the concept closure of D. The automaton moves to a state to check whether w represents an instance of C. Then C is recursively decomposed, possibly navigating to the neighbors of w. When a simple role

*S and a node w are reached during the decomposition, the automation may need to verify whether S holds between (*i*) the predecessor of w and w, (*ii*) w and an individual node, or (*iii*) w and itself. This will be achieved by transitions that respectively use the states in $Cl_{\mathsf{R}}(D)$, $Q_{\uparrow}$, and Q_{Self}. After fully decomposing concepts and roles, the basic symbols are checked locally at the node labels, using corresponding states from $\{s, \neg s \mid s \in \Theta(D)\}$. The other sets contain auxiliary states whose role will detailed in the description of the transitions.*

- *The transition function $\delta_D : Q_D \times \Sigma_D \to \mathcal{B}(\mathbf{D}_{b_D} \times Q_D)$ is described next. For readability, we organize the transitions into the following groups: 1. root checks, 2. concept checks, 3. role checks, 4. number restriction checks, and 5. atomic checks.*

  There are transitions for each $\sigma \in \Sigma_D$ as follows:

  1. *First, transitions from the initial state:*

$$\begin{aligned}\delta_D(q_D^0, \sigma) = \; & ([\mathsf{root}], D) \wedge \textstyle\bigwedge_{a \in \mathsf{N_I}(D)} (\langle \mathsf{root} \rangle, \{a\}) \;\; \wedge \\ & \textstyle\bigwedge_{a \in \mathsf{N_I}(D), s \in \Theta(D)} \big(([\mathsf{root}], (\!| a \Rightarrow s |\!)) \vee ([\mathsf{root}], (\!| a \Rightarrow \neg s |\!)) \big) \;\; \wedge \\ & \textstyle\bigwedge_{a \in \mathsf{N_I}(D), C \in Cl_{\mathsf{C}}(D)} \big(([\mathsf{root}], (\!| a \Rightarrow C |\!)) \vee ([\mathsf{root}], (\!| a \Rightarrow \sim C |\!)) \big)\end{aligned}$$

     This initial transition checks that the input forest represents a canonical model of D. Its four conjuncts respectively check that (i) all roots satisfy D, (ii) each individual name occurs in the label of some root, (iii) all pairs of roots whose labels share some individual name have identical labels, and (iv) all pairs of roots whose labels share some individual name satisfy the same concepts in the closure. Conditions (1iii) and (1iv) are be important later, when we show that if $\mathbf{A}_D$ accepts some forest, then it accepts an individual unique.

     For testing (1iii) and (1iv), $\mathbf{A}_D$ moves to the states in $Q_{\mathsf{N_I}}$. For each state $(\!| a \Rightarrow \alpha |\!) \in Q_{\mathsf{N_I}}$ there is a transition

$$\delta_D((\!| a \Rightarrow \alpha |\!), \sigma) = (\varepsilon, \neg\{a\}) \vee (\varepsilon, \alpha).$$

     which intuitively ensures that if the current node is an a-node, then it satisfies α. In this way, the automaton can ensure for each a and each α, that either α holds at each pair of a-nodes, or the negation of α holds at each pair of a-nodes.

  2. *The automaton moves to a node w and a state corresponding to a concept C in $Cl_{\mathsf{C}}(D)$ if it wants to verify whether w satisfies C. This is achieved with transitions that recursively decompose C and its subconcepts, as well as the non-simple roles inside the existential and universal restrictions, shown on the left side of Table 3.6.*

  3. *The automaton moves to a node w and a state for a simple role S in $Cl_{\mathsf{R}}(D)$ in order to verify S between the predecessor of w and w. Then it recursively decomposes S with transitions to the corresponding states in $Cl_{\mathsf{R}}(D)$. If it needs to verify whether S holds between between w and itself, the automaton moves to the state S_{Self} in Q_{Self} and then S is decomposed. Finally, to verify whether S holds between w and an a-node, it proceeds similarly moving to $\uparrow_a^S$ and decomposing S with transitions to other states in $Q_{\uparrow}$. These transitions are given on the right side of Table 3.6.*

  4. *We next give the transitions that ensure satisfaction of the number restrictions. For each $\geqslant n\, S.C$ in $Cl_{\mathsf{C}}(D)$, we define:*

$$\delta_D(\geqslant n\, S.C, \sigma) = \bigvee_{0 \leq k \leq |\mathsf{N_I}(D)|} ((\varepsilon, (\!| \langle \mathsf{root} \rangle^k, S, C |\!)) \wedge \mathsf{NR}(n-k, S, C))$$

     where $\mathsf{NR}(m, S, C)$ is as follows:

$$
\begin{array}{rcl}
\delta_{\mathcal{T}}(C_1 \sqcap C_2, \sigma) & = & (\varepsilon, C_1) \ \wedge \ (\varepsilon, C_2) \\
\delta_{\mathcal{T}}(C_1 \sqcup C_2, \sigma) & = & (\varepsilon, C_1) \ \vee \ (\varepsilon, C_2) \\
\delta_{\mathcal{T}}(\exists S.\mathsf{Self}, \sigma) & = & (\varepsilon, S_{\mathsf{Self}}) \\
\delta_{\mathcal{T}}(\neg\exists S.\mathsf{Self}, \sigma) & = & (\varepsilon, (\sim S)_{\mathsf{Self}}) \\
\delta_{\mathcal{T}}(\forall(R_1 \cup R_2).C, \sigma) & = & (\varepsilon, \forall R_1.C \sqcap \forall R_2.C) \\
\delta_{\mathcal{T}}(\forall(R_1 \circ R_2).C, \sigma) & = & (\varepsilon, \forall R_1.\forall R_2.C) \\
\delta_{\mathcal{T}}(\forall R^*.C, \sigma) & = & (\varepsilon, C \sqcap \forall R.\forall R^*.C) \\
\delta_{\mathcal{T}}(\forall id(C_1).C_2, \sigma) & = & (\varepsilon, \sim C_1 \sqcup C_2) \\
\delta_{\mathcal{T}}(\exists(R_1 \cup R_2).C, \sigma) & = & (\varepsilon, \exists R_1.C \sqcup \exists R_2.C) \\
\delta_{\mathcal{T}}(\exists(R_1 \circ R_2).C, \sigma) & = & (\varepsilon, \exists R_1.\exists R_2.C) \\
\delta_{\mathcal{T}}(\exists R^*.C, \sigma) & = & (\varepsilon, C \sqcup \exists R.\exists R^*.C) \\
\delta_{\mathcal{T}}(\exists id(C_1).C_2, \sigma) & = & (\varepsilon, C_1 \sqcap C_2) \\
\delta_{\mathcal{T}}(\forall S.C, \sigma) & = & (\varepsilon, \leqslant 0\, S.\sim C) \\
\delta_{\mathcal{T}}(\exists S.C, \sigma) & = & (\varepsilon, \geqslant 1\, S.C)
\end{array}
$$

$$
\begin{array}{rcl}
\delta_{\mathcal{T}}(S_1 \cap S_2, \sigma) & = & (\varepsilon, S_1) \ \wedge \ (\varepsilon, S_2) \\
\delta_{\mathcal{T}}(S_1 \cup S_2, \sigma) & = & (\varepsilon, S_1) \ \vee \ (\varepsilon, S_2) \\
\delta_{\mathcal{T}}(P_1 \setminus P_2, \sigma) & = & (\varepsilon, P_1) \ \wedge \ (\varepsilon, \neg P_2) \\
\delta_{\mathcal{T}}((S_1 \cap S_2)_{\mathsf{Self}}, \sigma) & = & (\varepsilon, S_{1\mathsf{Self}}) \wedge (\varepsilon, S_{2\mathsf{Self}}) \\
\delta_{\mathcal{T}}((S_1 \cup S_2)_{\mathsf{Self}}, \sigma) & = & (\varepsilon, S_{1\mathsf{Self}}) \vee (\varepsilon, S_{2\mathsf{Self}}) \\
\delta_{\mathcal{T}}((P_1 \setminus P_2)_{\mathsf{Self}}, \sigma) & = & (\varepsilon, P_{1\mathsf{Self}}) \wedge (\varepsilon, \neg P_{2\mathsf{Self}}) \\
\delta_{\mathcal{T}}(p^-{}_{\mathsf{Self}}, \sigma) & = & (\varepsilon, p_{\mathsf{Self}}) \\
\delta_{\mathcal{T}}((\neg p^-)_{\mathsf{Self}}, \sigma) & = & (\varepsilon, (\neg p)_{\mathsf{Self}}) \\
\delta_D(\uparrow_a^{S_1 \cap S_2}, \sigma) & = & (\varepsilon, \uparrow_a^{S_1}) \wedge (\varepsilon, \uparrow_a^{S_2}), \\
\delta_D(\uparrow_a^{S_1 \cup S_2}, \sigma) & = & (\varepsilon, \uparrow_a^{S_1}) \vee (\varepsilon, \uparrow_a^{S_2}), \\
\delta_D(\uparrow_a^{P \setminus P_2}, \sigma) & = & (\varepsilon, \uparrow_a^{P}) \wedge (\varepsilon, \neg\uparrow_a^{P_2})
\end{array}
$$

Table 3.6: Transitions in $\mathbf{A}_D$ from groups 2 (left) and 3 (right), where $\sigma \in \Sigma_D$, $C \in Cl_{\mathsf{C}}(D)$, a is an individual name, and S, P and p in $Cl_{\mathsf{R}}(D)$ are a simple role, an atomic role, and a role name, respectively.

– *for* $2 \leq m \leq n$, $\mathsf{NR}(m, S, C) = \varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4$ *and*

$$
\begin{array}{l}
\varphi_1 = (\langle m\rangle, (\!|\wedge, S, C|\!)), \\
\varphi_2 = (\varepsilon, S_{\mathsf{Self}}) \wedge (\varepsilon, C) \wedge \big(\bigwedge_{a \in \mathsf{N_I}(D)} (\varepsilon, \neg\{a\})\big) \wedge (\langle m{-}1\rangle, (\!|\wedge, S, C|\!)), \\
\varphi_3 = (\varepsilon, \mathsf{Inv}(S)) \wedge (-1, C) \wedge \big(\bigwedge_{a \in \mathsf{N_I}(D)} (-1, \neg\{a\})\big) \wedge (\langle m{-}1\rangle, (\!|\wedge, S, C|\!)), \\
\varphi_4 = (\varepsilon, S_{\mathsf{Self}}) \wedge (\varepsilon, C) \wedge (\varepsilon, \mathsf{Inv}(S)) \wedge (-1, C) \ \wedge \\
\qquad \big(\bigwedge_{a \in \mathsf{N_I}(D)} (-1, \neg\{a\})\big) \wedge (\langle m{-}2\rangle, (\!|\wedge, S, C|\!)).
\end{array}
$$

– $\mathsf{NR}(1, S, C) = \varphi_1 \vee \varphi_2 \vee \varphi_3$ *with* φ_1, φ_2 *and* φ_3 *as above, and*

– $\mathsf{NR}(m, S, C) = \boldsymbol{true}$ *for* $m \leq 0$.

To understand these transitions, suppose satisfaction of $\geqslant n\, S.C$ *is verified at node* w. *Then there must exist distinct nodes* $w'_1, \ldots, w'_n$ *for which the following holds: (*$*$*) w is related to w'_i via S and w'_i satisfies D. Each of these w'_i may be (cf. Def. 3.2.2): (4a) a node in* $\mathsf{succ}(w)$, *(4b) the predecessor of w in F, (4c) an individual node, or (4d) w itself. The big disjunction searches for some k such that k individual nodes (4c) and $m = n{-}k$ non-individual nodes of the kinds (4a), (4b) and (4d) satisfy (*$*$*).*

The latter check is done via $\mathsf{NR}(m, S, C)$. *Its disjuncts* φ_1 *to* φ_4 *correspond to the four possible ways in which these m nodes can be found among the nodes of types (4a), (4b) and (4d), viz.:*
(φ_1) m successors of w satisfy ($*$*),*
(φ_2) w itself and (at least) $m{-}1$ successors of w satisfy ($*$*),*
(φ_3) the predecessor of w and (at least) $m{-}1$ successors of w satisfy ($*$*), or*
(φ_4) w itself, the predecessor of w, and (at least) $m{-}2$ successors of w satisfy ($*$*).*

Next, the following transitions for each $(\!|\langle\mathsf{root}\rangle^k, S, C|\!)$ *in* Q_{roots} *check whether k indi-*

vidual nodes satisfy (∗):

$$\delta_D(\langle\!|\langle \mathsf{root}\rangle^k, S, C|\!\rangle, \sigma) = \bigvee_{N\subseteq \mathsf{N_I}(D),\, |N|=k} \Big(\bigwedge_{a\in N} ((\varepsilon, \uparrow_a^S) \vee (\langle \mathsf{root}\rangle, \langle\!|\wedge, \{a\}, C|\!\rangle)) \wedge \bigwedge_{a,b\in N,\, a\neq b} ([\mathsf{root}], \langle\!|\vee, \neg a, \neg b|\!\rangle) \Big)$$

Intuitively, such a transition checks whether there is some subset $N \subseteq \mathsf{N_I}(D)$ with cardinality k, such that the individuals in N are all interpreted as k different roots satisfying (∗). It is a big disjunction over the possible sets N. For each of them, the first conjunct checks that for each $a \in N$, the current node is related via S to an a-node that satisfies C, and the second conjunct checks that each pair of individuals $a, b \in N$ do not occur in the label of the same root, to ensure that the k individuals in N correspond to different individual nodes.

These transitions move to auxiliary states in $\langle\!|\vee, \neg a, \neg b|\!\rangle$ in Q_{roots} and $\langle\!|\wedge, \alpha, C|\!\rangle$ in Q_{bin}. For each $\langle\!|\vee, \neg a, \neg b|\!\rangle$ in Q_{Nom}, there is a transition

$$\delta_D(\langle\!|\vee, \neg a, \neg b|\!\rangle, \sigma) = (\varepsilon, \neg\{a\}) \vee (\varepsilon, \neg\{b\})$$

that checks that the current node does not contain both a and b, and for each $\langle\!|\wedge, \alpha, C|\!\rangle$ in Q_{bin} there is a transition

$$\delta_D(\langle\!|\wedge, \alpha, C|\!\rangle, \sigma) = (\varepsilon, \alpha) \wedge (\varepsilon, C)$$

that verifies that the current node satisfies both α and C.

The transitions for concepts $\leqslant n\, S.C$ are analogous. If $\leqslant n\, S.C$ is satisfied at a node w, then there must be some $k \leq n$ such that at most k individual nodes (4c) and at most $m = n-k$ non-individual nodes of the kinds (4a), (4b) and (4d) satisfy (†) w' is related to w by S and satisfies C. This is reflected in the following transition, for each $\leqslant n\, S.C$ in $Cl_{\mathsf{C}}(D)$:

$$\delta_D(\leqslant n\, S.C, \sigma) = \bigvee_{0\leq k\leq \min(n, |\mathsf{N_I}(D)|)} \big((\varepsilon, \langle\!|[\mathsf{root}]^k, S, C|\!\rangle) \wedge \mathsf{NR}'(n-k, S, C)\big)$$

where $\mathsf{NR}'(m, S, C)$ is a shortcut for the following formula that verifies whether at most m non-individual nodes of the kinds (4a), (4b) and (4d) satisfy (†). The check for (†) is done in its contrapositive form. That is, to check whether at most m nodes satisfy (†), we actually check whether all but m nodes satisfy the negation of (†), which is (‡) w' is not related to w via S or does not satisfy C:

- *for $2 \leq m \leq n$, $\mathsf{NR}'(m, S, C) = \varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4$ and*

$$\begin{aligned}
\varphi_1 &= (\varepsilon, {\sim}S_{\mathsf{Self}}) \wedge (\varepsilon, {\sim}C) \wedge (\varepsilon, \mathsf{Inv}({\sim}S)) \wedge (-1, {\sim}C) \wedge \big(\textstyle\bigwedge_{a\in \mathsf{N_I}(D)} (-1, \neg\{a\})\big) \\
&\quad \wedge([m], \langle\!|\vee, {\sim}S, {\sim}C|\!\rangle), \\
\varphi_2 &= (\varepsilon, \mathsf{Inv}({\sim}S)) \wedge (-1, {\sim}C) \wedge \big(\textstyle\bigwedge_{a\in \mathsf{N_I}(D)} (-1, \neg\{a\})\big) \wedge ([m-1], \langle\!|\vee, {\sim}S, {\sim}C|\!\rangle), \\
\varphi_3 &= (\varepsilon, {\sim}S_{\mathsf{Self}}) \wedge (\varepsilon, {\sim}C) \wedge \big(\textstyle\bigwedge_{a\in \mathsf{N_I}(D)} (\varepsilon, \neg\{a\})\big) \wedge ([m-1], \langle\!|\vee, {\sim}S, {\sim}C|\!\rangle), \\
\varphi_4 &= ([m-2], \langle\!|\vee, {\sim}S, {\sim}C|\!\rangle);
\end{aligned}$$

- $\mathsf{NR}'(1, S, C) = \varphi_1 \vee \varphi_2 \vee \varphi_3$ *with φ_1, φ_2 and φ_3 as above, and*
- $\mathsf{NR}'(0, S, C) = \varphi_1$ *with φ_1 as above.*

Again, the disjuncts φ_1 to φ_4 correspond to the four possible ways in which the at most m nodes satisfying (†) may be distributed among the nodes of types (4a), (4b) and (4d):

- *(φ_1) they are all successors of w, hence the predecessor of w, w itself and all but m successors of w satisfy (‡),*
- *(φ_2) they include w itself and up to $m-1$ successors of w, hence the predecessor of w and all but $m-1$ of its successors satisfy (‡),*
- *(φ_3) they include the predecessor of w and up to $m-1$ successors of w, hence w itself and all but $m-1$ of its successors satisfy (‡), or*
- *(φ_4) they include w, its predecessor, and up to $m-2$ of its successors, hence $m-2$ successors of w satisfy (‡).*

Next, the following transitions for each $(\!|[\mathsf{root}]^k, {\sim}S, {\sim}C|\!)$ in Q_{roots} check whether at most k individual nodes satisfy (†):

$$\delta_D((\!|[\mathsf{root}]^k, {\sim}S, {\sim}C|\!), \sigma) = \bigvee_{N \subseteq \mathsf{N_I}(D),\, |N|=k} \Big(\bigwedge_{a \notin N} ((\varepsilon, \uparrow_a^{\sim S}) \vee (\langle \mathsf{root} \rangle, (\!|\wedge, \{a\}, {\sim}C|\!))) \Big)$$

Intuitively, such a transition checks whether there is some subset $N \subseteq \mathsf{N_I}(D)$ with cardinality k, such that only the roots corresponding to individuals in N satisfy (†), or equivalently, every individual node that is not an a-node with $a \in N$ satisfies (‡). It is a big disjunction over the possible sets N. For each of them, there is a conjunction that checks that for each $a \notin N$, either the current node has no a-node as S successor, or the a-nodes do not satisfy C. Note that is this case, it does not matter whether the k individuals nodes are different.

The transitions move to auxiliary states $(\!|\wedge, \alpha, C|\!)$ and $(\!|\vee, \alpha, C|\!)$ in Q_{bin}. For the former ones the transitions were already described. For each $(\!|\vee, \alpha, C|\!)$ in Q_{bin} there is a transition

$$\delta_D((\!|\vee, \alpha, C|\!), \sigma) = (\varepsilon, \alpha) \vee (\varepsilon, C)$$

that verifies that the current node satisfies α or C.

5. *The above transitions decompose all concepts and roles until they reach states corresponding to possibly negated atomic expressions and special symbols, and then $\mathbf{A}_D$ checks whether they are in the label of the current node. For each $s \in \Theta(D)$ there are transitions:*

$$\delta_D(s, \sigma) = \begin{cases} \textbf{\textit{true}}, & \text{if } s \in \sigma, \\ \textbf{\textit{false}}, & \text{if } s \notin \sigma, \end{cases} \qquad \delta_D(\neg s, \sigma) = \begin{cases} \textbf{\textit{true}}, & \text{if } s \notin \sigma, \\ \textbf{\textit{false}}, & \text{if } s \in \sigma. \end{cases}$$

- *$G_D = (\emptyset, \{\forall R^*.C \mid \forall R^*.C \in Cl_{\mathsf{C}}(D)\}, Q_D)$ is the acceptance condition. As we show below, this condition ensures that each concept $\exists R^*.C$ is eventually realized.*

The automaton $\mathbf{A}_D$ accepts the representation of the canonical models of D. The converse holds in a slightly weaker form: every forest accepted by $\mathbf{A}_D$ represents an ind-model of D provided that it is *individual unique*. Hence $\mathscr{L}(\mathbf{A}_D) \neq \emptyset$ implies that D has an ind-model because, as we show below, if $\mathscr{L}(\mathbf{A}_D)$ accepts some forest, then it accepts an individual unique one. We actually show something slightly stronger, that will be useful when we extend our algorithm to query answering in the next chapter: $\mathbf{A}_D$ accepts an individual unique forest whose branching degree is bounded by b_D.

Proposition 3.3.8 *Let D be a normal $\mathcal{ZOIQ}$ concept. Then the following hold for $\mathbf{A}_D$:*

1. *If $\mathcal{I}$ is a canonical model of D, then $\mathbf{F}_{\mathcal{I}} \in \mathscr{L}(\mathbf{A}_D)$.*
2. *For every D-forest $\mathbf{F}$, $\mathbf{F} \in \mathscr{L}(\mathbf{A}_D)$ implies $\mathcal{I}_{\mathbf{F}} \models_{\mathsf{ind}} D$.*
3. *If $\mathscr{L}(\mathbf{A}_D) \neq \emptyset$, then $\mathbf{F} \in \mathscr{L}(\mathbf{A}_D)$ for some D-forest $\mathbf{F}$ whose branching degree is bounded by b_D.*

Hence $\mathscr{L}(\mathbf{A}_D) \neq \emptyset$ if D has a canonical model, and D has an ind-model if $\mathscr{L}(\mathbf{A}_D) \neq \emptyset$.

Proof. For the first item, assume $\mathcal{I}$ is a canonical model of D and $\mathbf{F}_{\mathcal{I}} = \langle \Delta^{\mathcal{I}}, L_{\mathcal{I}} \rangle$ its forest encoding. To show that there is an accepting run $\langle T_r, r \rangle$ of $\mathbf{A}_D$ on $\mathbf{F}_{\mathcal{I}}$, we use a well-founded adorned pre-model $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ (which exists by Lemma 3.2.10) to guide the selection of a set of atoms for constructing a run, starting from the root.

Formally, one shows easily show the following, for each pair $(w, q) \in \Delta^{\mathcal{I}} \times Q_D$:

(a) If $q = s \in \Theta(D)$ and $s \in L(w)$, or $q = \neg s$ for some $s \in \Theta(D)$ and $s \notin L(w)$, there is a finite $\Delta^{\mathcal{I}} \times Q_D$-labeled tree $\langle T, r \rangle$ whose root t is labeled $r(t) = (w, q)$, and where each node satisfies condition 2 in Definition 3.3.5. (in fact, the root t is the only node of T).

(b) If S is a role and

   (i) $q = S \in Cl_{\mathsf{R}}(D)$ and $w \in \mathsf{succ}(u)$ for some u with $S \in \theta(u, w)$, or

   (ii) $q = S_{\mathsf{Self}} \in Q_{\mathsf{Self}}$ and $S \in \theta(w, w)$, or

   (iii) $q = \uparrow_a^S \in Q_{\uparrow}$ and $S \in \theta(w, a^{\mathcal{I}})$,

   then there is a finite $\Delta^{\mathcal{I}} \times Q_D$-labeled tree $\langle T, r \rangle$ whose root t has label $r(t) = (w, q)$, and where each node satisfies condition 2 in Definition 3.3.5. The proof is a straightforward structural induction on the role S.

(c) If $q = C \in Cl_{\mathsf{C}}(D)$ is a concept and $C \in \theta(w)$, then there is a $\Delta^{\mathcal{I}} \times Q_D$-labeled tree $\langle T, r \rangle$ such that its root t has label $r(t) = (w, C)$, each node in T satisfies condition 2 in Definition 3.3.5, and for every infinite path P of T_r, there is some star concept $C' \in Cl_{\mathsf{C}}(C)$ such that

   (i) $r(t_0) = (w_0, C')$ for some $t_0 \in P$,

   (ii) for every descendant $t_i \in P$ of t_0 with $r(t_i) = (w_i, C_i)$ and $C_i \in Cl_{\mathsf{C}}(C_{\mathcal{K}})$, $C_i \in Cl_{\mathsf{C}}(C')$,

   (iii) if C is $\exists R^*.C'$, then $\exists R^*.C'$ is regenerated from w_i to w_{i+1} for each $t_i, t_{i+1} \in P$ with $r(t_i) = (w_i, \exists R^*.C')$ and $r(t_{i+1}) = (w_{i+1}, \exists R^*.C')$.

   Intuitively, the claim follows from the close correspondence between the transitions in Table 3.6 and the definition of the $\rightsquigarrow$ relation in Definition 3.2.8. It can be shown inductively that the tree $\langle T, r \rangle$ can be constructed by moving always from the current $r(t) = (w, C)$ to some $r(t') = (w', C')$ such that $(w, C) \rightsquigarrow (w', C')$, possibly adding additional auxiliary nodes in between. Such a run always decomposes a concept into less complex concepts, except in the case of the star nodes which may be regenerated and thus repeated along the branches.

Using (a) to (c), it is easy to show that a full run of $\mathbf{A}_D$ over $\mathbf{F}_{\mathcal{I}}$ exists. The run $\langle T_r, r \rangle$ is built starting from the root ε, and setting $r(\varepsilon) = (c_0, q_0)$. Then, to correctly execute the initial transition, the root has children as follows:

- a child j_c with $r(j_c) = (c, D)$ for each root $c \in \mathsf{roots}(\Delta^{\mathcal{I}})$.
- a child j_a for each $a \in \mathsf{N_I}(D)$, with $r(j_a) = (a^{\mathcal{I}}, \{a\})$,

- for each root $c \in \mathsf{roots}(\Delta^{\mathcal{I}})$, each $a \in \mathsf{N_I}(D)$ and each $s \in \Theta(D)$, a child $j_{c,a,s}$ with $r(j_{c,a,s}) = (c, (\!|a, s|\!))$ if $s \in L(a^{\mathcal{I}})$, and $r(j_{c,a,s}) = (c, (\!|a, \neg s|\!))$ otherwise, and
- for each root $c \in \mathsf{roots}(\Delta^{\mathcal{I}})$, each $a \in \mathsf{N_I}(D)$ and each $C \in Cl_{\mathsf{C}}(D)$, a child $j_{c,a,C}$ with $r(j_{c,a,C}) = (c, (\!|a, C|\!))$ if $C \in \theta(a^{\mathcal{I}})$, and $r(j_{c,a,C}) = (c, (\!|a, \sim C|\!))$ otherwise.

Such a tree can be easily completed into a run. To see that it is accepting, suppose towards a contradiction that there is an infinite path P of T_r such that the least i such that $G_i \cap \mathsf{Inf}(\langle P, r\rangle)$ is not empty is odd. This is only possible if $i = 3$, and there are no states of the form $\forall R^*.C$ in $\mathsf{Inf}(\langle P, r\rangle)$. Then by item (c.iii), there is some concept $\exists R^*.C \in \mathsf{Inf}(\langle P, r\rangle)$ that is regenerated from w_i to w_{i+1} for each $t_i, t_{i+1} \in P$ with $r(t_i) = (w_i, \exists R^*.C)$ and $r(t_{i+1}) = (w_{i+1}, \exists R^*.C)$, but this contradicts the fact that $\langle \mathcal{I}, \theta, \mathsf{ch}\rangle$ is well founded. This concludes the proof of item 1 in the claim.

To show the second item, we first consider a D-forest $\mathbf{F}$ and an arbitrary $\Delta^{\mathcal{I}_\mathbf{F}} \times Q_D$-labeled tree $\langle T, r\rangle$ where each node satisfies condition 2 in Definition 3.3.5. Then we can show, similarly as above, that for every pair $(w, q) \in \Delta^{\mathcal{I}} \times Q_D$ such that there is some $t \in T$ with $r(t) = (w, q)$, the following hold:

(a) If $q = s \in \Theta(D)$, then $s \in L(w)$, and if $q = \neg s$ for some $s \in \Theta(D)$, then $s \notin L(w)$.

(b) If $q = S \in Cl_{\mathsf{R}}(D)$, then $S \in \theta(w \cdot -1, w)$; if $q = S_{\mathsf{Self}} \in Q_{\mathsf{Self}}$, then $S \in \theta(w, w)$; and $q = \uparrow_a^S \in Q_\uparrow$, then $S \in \theta(w, a^{\mathcal{I}_\mathbf{F}})$. This is shown by a straightforward structural induction on the role S.

(c) If $q = C \in Cl_{\mathsf{C}}(D)$ is a concept, then $\mathcal{I}_\mathbf{F}, w \models_{\mathsf{pre}} C$. Furthermore, if C is of the form $\exists R^*.C'$, then it is eventually realized for w, or there is an infinite path P in T such that $C \in \mathsf{Inf}(\langle P, r\rangle)$ and $\mathsf{Inf}(\langle P, r\rangle) \subseteq Cl_{\mathsf{C}}(C)$. This can be easily shown by induction on C, following the definition of δ_D and Table 3.2, and using the previous items.

By the initial transitions 1 in Definition 3.3.7, for each $a \in \mathsf{N_I}(D)$ there is some $c \in \mathsf{roots}(\mathcal{I}_\mathbf{F})$ such that the run $\langle T, r\rangle$ has nodes $i, j \in \mathbb{N}$ with $r(i) = (c, \{a\})$ and $r(j) = (c, D)$. Then, by c, we have $\mathcal{I}_\mathbf{F}, c \models_{\mathsf{pre}} \{a\}$ and $\mathcal{I}_\mathbf{F}, c \models_{\mathsf{pre}} D$, which implies $\mathcal{I}_\mathbf{F}, a^{\mathcal{I}_\mathbf{F}} \models_{\mathsf{pre}} D$. The rest of the proof is standard. Since the run is accepting, by item c, there is no path $P \in T$ and no concept $\exists R^*.C$ such that $\exists R^*.C \in \mathsf{Inf}(\langle P, r\rangle)$, and every other state occurring infinitely often is a subconcept of $\exists R^*.C$, that is $\mathsf{Inf}(\langle P, r\rangle) \subseteq Cl_{\mathsf{C}}(\exists R^*.C)$. Using the standard techniques, one can show that this implies the existence of a choice function ch that does not generate a concept $\exists R^*.C$ infinitely often; the detailed proof relies on the existence of a *memoryless strategy* [EJ91, BLMV08].

Finally, to show 3, we show that if $\mathbf{A}_D$ accepts some $\mathbf{F}$, then it accepts some $\mathbf{F}'$ that is b_D-ary and individual unique. Roughly, if F has two subtrees whose roots c and c' have the same labels and satisfy the same concepts in the closure, then a run of $\mathbf{A}_D$ on $\mathbf{F}$ visiting both can be modified into a run visiting only one of them. Then all nodes that are not needed to satisfy some $\langle n\rangle$ transition can be dropped, and the resulting forest has the desired features.

More precisely, consider two root nodes c_1 and c_2 of $\mathbf{F}$. In what follows, we call a *partial run tree* a $\Delta^{\mathbf{F}} \times Q_D$-labeled tree where each node satisfies condition 2 in Definition 3.3.5 and each infinite path satisfies the acceptance condition. Assume $(*)$: $L(c_1) = L(c_2)$ and that for every $C \in Cl_{\mathsf{C}}(D)$, there is a partial run tree whose root is labeled $r(c_1, C)$ iff there is a partial run tree whose root is labeled $r(c_2, C)$. Intuitively, $(*)$ implies that the two equally labeled roots are not distinguishable by the automaton if they are visited in some state $C \in Cl_{\mathsf{C}}(D)$. Inspecting the transition function δ_D it is not hard to verify that this indistinguishability extends to all states of $\mathbf{A}_D$. That is, for every $q \in Q_D$, if a run has as subtree a partial run tree T_t with $r(t) = r(c_2, q)$, then it can be modified by replacing T_t with a partial run tree T'_t with $r(t) = r(c_1, q)$. One can then show using the standard techniques (namely, the existence of a

strategy) that if there is an accepting run of $\mathbf{A}_D$ on $\mathbf{F}$, then there is a run where no node in the tree T_{c_2} is ever visited. Further, if there is an accepting run of $\mathbf{A}_D$ on $\mathbf{F}$, then by the initial transition 1 in Definition 3.3.7 every pair c_1 and c_2 of a-nodes for each $a \in \mathsf{N_I}(D)$ satisfies $(*)$. Then it follows that there is an accepting run of $\mathbf{A}_D$ on $\mathbf{F}$ where for each a, only one a-node and its successors are visited. Finally, if there is such an accepting run over $\mathbf{F}$, then there is also an accepting run over the forest $\mathbf{F}'$ obtained by dropping each w that its not required to satisfy a $\langle \mathsf{root} \rangle$ or a $\langle n \rangle$ transition. Clearly, $\mathbf{F}'$ is individual unique, and its branching degree is bounded by b_D because $\mathbf{A}_D$ never needs more than b_D nodes to satisfy a $\langle n \rangle$ transition. This concludes the proof of the claim. □

From this and Propositions 3.1.5 and 3.2.4, we obtain:

Corollary 3.3.9 *Let $\mathcal{K}$ be a knowledge base in $\mathcal{ZOIQ}$ that enjoys the canonical model property (or in particular, that is in $\mathcal{ZIQ}$, $\mathcal{ZOI}$, or $\mathcal{ZOQ}$). Then we can construct from $\mathcal{K}$ (in time linearly bounded by $||\mathcal{K}||$) a concept $C_\mathcal{K}$ such that $\mathscr{L}(\mathbf{A}_{C_\mathcal{K}}) \neq \emptyset$ iff $\mathcal{K}$ is satisfiable.*

3.4 Complexity of Deciding Satisfiability

The following bounds on the size of $\mathbf{A}_D$ are important for obtaining from our reduction to automata emptiness an upper bound for deciding KB satisfiability.

Lemma 3.4.1 *Let D be a normal $\mathcal{ZOIQ}$ concept. Then the following hold for $\mathbf{A}_D$:*

1. *the number of states $|Q_D|$ is polynomial in $||D||$,*
2. *the alphabet size $|\Sigma(\mathbf{A}_D)|$ and the counting bound b_D are at most single exponential in $||D||$, and*
3. *the parity index of $\mathbf{A}_D$ is fixed.*

Proof. For the third item, simply observe that the parity index of $\mathbf{A}_D$ is 3. It is also clear that $b_D = \max(\{n \mid \geqslant n\, S.C \in \mathit{Cl}_\mathsf{C}(D)\} \cup \{0\})$ is bounded by $2^{||D||}$ (if the numbers are encoded in binary).

To estimate the bounds for $|\Sigma(\mathbf{A}_D)|$ and $|Q(\mathbf{A}_D)|$, we first observe that $|\mathsf{N_{CI}}(D)|$, $|\mathsf{N_R}(D)|$, $|\overline{\mathsf{N_R}}(D)|$, $|\mathsf{N_I}(D)|$ are all linearly bounded in $||D||$, and so are $|\mathit{Cl}(D)|$, $|\mathit{Cl}_\mathsf{C}(D)|$, and $|\mathit{Cl}_\mathsf{R}(D)|$. The result is then a consequence of the following simple estimates:

- $|\Sigma_D| = 2^{|\Theta(D)|}$, and $|\Theta(D)| = |\mathsf{N_{CI}}(D)| + |\overline{\mathsf{N_R}}(D)| + |\mathsf{N_R}(D)| + (|\overline{\mathsf{N_R}}(D)| \times |\mathsf{N_I}(D)|)$. It thus follows that $|\Sigma(\mathbf{A}_D)| \leq 2^{O(||D||^2)}$.
- $|Q_D| = 1 + |\mathit{Cl}(D)| + (2 \times |\Theta(D)|) + |Q_{\mathsf{N_I}}| + |Q_{\mathsf{Self}}| + |Q_\uparrow| + |Q_{\mathsf{Nom}}| + |Q_{\mathsf{roots}}| + |Q_{\mathsf{bin}}|$, where

$$
\begin{aligned}
|Q_{\mathsf{N_I}}| &= 2 \cdot |\mathsf{N_I}(D)| \cdot (|\mathit{Cl}_\mathsf{C}(D)| + |\Theta(D)|) \\
|Q_{\mathsf{Self}}| &\leq |\mathit{Cl}_\mathsf{R}(D)| \\
|Q_\uparrow| &\leq |\mathsf{N_I}(D)| \cdot |\mathit{Cl}_\mathsf{R}(D)| \\
|Q_{\mathsf{Nom}}| &= |\mathsf{N_I}(D)|^2 \\
|Q_{\mathsf{roots}}| &\leq 2 \cdot |\mathsf{N_I}(D)| \cdot |\mathit{Cl}_\mathsf{C}(D)|^2 \\
|Q_{\mathsf{bin}}| &\leq 2 \cdot |\mathit{Cl}_\mathsf{C}(D)|^2 + 2 \cdot |\mathsf{N_I}(D)| \cdot |\mathit{Cl}_\mathsf{C}(D)|
\end{aligned}
$$

Hence it is easy to see that $|Q_D| = O(||D||^3)$.

□

In this way we obtain we obtain our first main result: an ExpTime upper bound for the KB satisfiability of all $\mathcal{ZOIQ}$ knowledge bases that enjoy the canonical model property. This is worst-case optimal.

Theorem 3.4.2 *KB satisfiability in $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$ is* ExpTime*-complete.*

Proof. For the upper bound, by Corollary 3.3.9 it suffices to decide whether $\mathscr{L}(\mathbf{A}_{C_\mathcal{K}})$ is non-empty. By Lemma 3.4.1 and Theorem 3.3.6, this can be decided in time single exponential in $||C_\mathcal{K}||$, and thus single exponential in $||\mathcal{K}||$. A matching hardness result is known already for much weaker DLs, e.g. $\mathcal{ALC}$ [BCM$^+$03]. □

3.5 Related Work and Discussion

The construction of the automaton $\mathbf{A}_D$ for a concept D and some of the proof techniques in this chapter are inspired by [BLMV08], where tight upper bounds for the satisfiability problem of two enriched μ-calculi were established. The basic μ-calculus μL extends classical propositional Modal Logic with fixed-point operators, resulting in an expressive logic that subsumes most modal, dynamic and temporal logics [Koz83]. Many reasoning problems for the basic DL $\mathcal{ALC}$ and its extension with regular role expressions (in particular, concept and TBox satisfiability, and other reasoning problems that are not related to the ABox) can be reduced to formula satisfiability in μL. The so called *enriched μ-calculi* in turn enrich the basic μ-calculus with different combinations of backward modalities [Var98], graded modalities [KSV02], and nominals [SV01, BLMV08], which are the modal logic counterpart of the description logic constructors inverse roles $\mathcal{I}$, number restrictions $\mathcal{Q}$, and nominals $\mathcal{O}$: (i) the hybrid graded μ-calculus supports graded modalities and nominals, and is thus close to $\mathcal{ZOQ}$; (ii) the full graded μ-calculus supports backward and graded modalities, similarly to $\mathcal{ZIQ}$; and (iii) the full hybrid μ-calculus supports backwards and graded modalities, similarly to $\mathcal{ZOI}$. The fixed point operators of present in μ-calculi are strictly more expressive than the regular expressions of $\mathcal{ZOIQ}$ and its sublogics [Koz83], while the latter have features, such as Boolean role expressions and self concepts, which are not naturally expressible in the enriched μ-calculi. ABox assertions can be simulated in the hybrid μ-calculi using nominals, but neither in the basic μL nor in its non-hybrid enriched versions. The formula satisfiability problem for the three calculi mentioned above can be decided in ExpTime [SV01, BLMV08], while the fully enriched μ-calculus that simultaneously allows all three constructors is undecidable [BP04].

The ExpTime upper bounds (under binary encoding of numbers) for the hybrid graded and full graded μ-calculi were obtained employing FEAs in [BLMV08]. Our technique is similar, but while they do two separate reductions, we give a single automata construction that simultaneously takes into account nominals, inverses, and counting. This requires us to combine in a novel way techniques that had been used separately for the different combinations of these three constructs, together with techniques for Boolean roles and concepts of the form $\exists S.\mathsf{Self}$ [CDGL02, Ort08]. Another significant difference is the following. Each canonical model of a $\mathcal{ZOIQ}$ concept D induces a partition of the individual names into equivalence classes such that all the individuals in a class are interpreted as the same domain element and satisfy the same atomic concepts. In [BLMV08], the authors use the notion of a *guess* to refer to a partition of the individual names in $\mathsf{N_I}(D)$ into classes, together with a subset of the atomic concepts in $\mathsf{N_{CI}}(D)$ associated to each class. Then they show how to construct an automaton $\mathbf{A}_{D,g}$ for a given guess g, such that the language of $\mathbf{A}_{D,g}$ is non-empty if there is a canonical model of D inducing the

guess g. This provides an algorithm for the satisfiability of D, since one can effectively traverse all possible guesses and do an emptiness test for each of the respective automata.

In contrast, instead of building an automaton for a specific guess, we build an automaton $\mathbf{A}_D$ that directly verifies the existence of a canonical model, and checks that it 'contains a correct guess', that is, that all individuals interpreted as the same domain element satisfy exactly the same atomic concepts. Although this requires additional states and makes some of the transitions more involved, it seems to be more convenient for our purposes. The query answering algorithm that we describe in the next chapter relies heavily on the fact that there is one automaton that can effectively verify on its own whether there exists a forest that represents a model of the KB.

The application of automata in DLs is not novel, and in fact they are widely recognized as a powerful tool for obtaining optimal complexity bounds, specially for ExpTime-complete logics. For DLs with regular expressions, in the style of the $\mathcal{Z}$ family, automata techniques inspired in PDL (Propositional Dynamic Logic, see [HKT00]) became popular since the work of [Sch91, DGL94, DG95]. An illustrative example can be found in [CDG03], where an optimal ExpTime upper for the DL $\mathcal{ALCFI}_{reg}$ (the extension of $\mathcal{ALCI}$ with regular role expressions and role functionality, which is a restricted form of number restrictions) is shown by an automata reduction related to the one we have described. Since reasoning in: (i) $\mathcal{ALCQI}_{reg}$, which extends $\mathcal{ALCFI}_{reg}$ with full qualified number restrictions, (ii) $\mathcal{ALCQO}_{reg}$, which is similar but has nominals and no inverses, and (iii) $\mathcal{ALCIO}_{reg}$, that has nominals and inverses but no number restrictions, can all be reduced to simpler logics for which automata algorithms are available, ExpTime bounds are also known for these logics [DGL94, DG95]. However, they only hold if the numbers in the number restrictions are encoded in unary. Most of the existing algorithms did not support additional constructors such as self concepts and Boolean role constructors, although it was conjectured that they could be incorporated. Our findings confirm this conjecture.

Automata have also been employed to obtain tight upper bounds for logics that do not support regular role expressions. For example, an optimal ExpTime upper bound for $\mathcal{ALCQI}b$, the extension of $\mathcal{ALCI}$ with qualified number restrictions and safe Boolean role expressions, was obtained using tree automata in [Tob01]. In order to obtain an optimal bound even when numbers are encoded in binary, Tobies represents canonical models with polynomial branching via binary trees, and does suitable book-keeping in the automaton to decide if a tree represents a model. As reasoning in $\mathcal{SHIQ}$ can be polynomially reduced to reasoning in $\mathcal{ALCQI}b$, the ExpTime upper bound extends to $\mathcal{SHIQ}$ as well. For the DL $\mathcal{SHOQ}$, which is similar to $\mathcal{SHIQ}$ but does not support inverse roles and supports nominals, an ExpTime upper bound under unary encoding of numbers was obtained in [Gli07, GHS08]. A matching bound under binary encoding of numbers was conjectured, but to our knowledge it remain to be proved, and the best known bound was a NExpTime one inherited from $\mathcal{C}_2$, the two variable fragment of first-order logic with counting quantifiers, [PH05]. For the DL $\mathcal{SHOI}$, that supports both inverses and nominals but does not allow for number restrictions, an upper bound was shown in [Hla04] by means of a *tableau system*, an abstract description of a (tableau-based) algorithm that implies the existence of an ExpTime automata algorithm [BHLW03].

In the automata we have described, the transitions that ensure satisfaction of the number restrictions are novel and differ from all previous approaches. Our transitions are more involved than those in [BLMV08] for two reasons. First, to simultaneously accommodate all constructors of $\mathcal{ZOIQ}$, our definition of canonical models (Def. 3.2.2) must include (4a) to (4d) above, while in the logics considered in [BLMV08] it suffices with either just (4a) and (4c) or just (4a) and (4b). Second, as we are not building an automaton for a specific guess, verifying which roots of F take part in the satisfaction of a number restriction is more complicated, and special care is needed to ensure that roots interpreting more than one nominal are not counted more than once. The transitions differ also from those in [CDGL02, CEO07, Ort08, CEO09a], which use

non-graded automata and count the successors of w one-by-one; this requires exponentially many states if n is encoded in binary.

Another distinguishing feature of our approach is the use of a parity condition in the automaton. For simpler DLs like the ones in the $\mathcal{SH}$ family, it is enough to consider a simpler *looping automaton* in which no acceptance condition is given and every infinite run is accepting. This is done, for example, in [Tob01] and [Gli07]. However, looping automata can not ensure the satisfaction of the star concepts in the $\mathcal{Z}$ family. The presence of alternating fixed points in the more powerful μ-calculi calls for a parity condition; in our setting an intermediate solution could be used. In fact, we could employ a *Büchi automaton* as done in [CDGL02, CDG03], which is more powerful than looping automaton but less than a parity one. A Büchi condition is given as a single set G of states, and a run is accepting if for each infinite path some state in G occurs infinitely often. The motivation for choosing a parity condition will be clearer in the next section, where we rely on automata operations like complementation under which Büchi automata are not closed.

A natural question is whether the technique can be generalized to other DLs. One natural direction would be to extend it to logics with arbitrary fixed-points rather than regular expressions. This may be easy, since we are already employing parity automata. Other extensions within DLs should also be feasible as long as they do not have a significant effect on the canonical model property. Intuitively, if it is possible to prove a canonical model property where canonical models still satisfy 1, 2 and 3 in Definition 3.2.2, and 4 can be adapted in a simple way, then the technique may be easy to adapt. In contrast, accommodating constructors for which a canonical model in the style of Definition 3.2.2 can not be ensured seems much harder, and may not be possible in many cases.

Decidability of full $\mathcal{ZOIQ}$ remains open. Our automata-based algorithm supports simultaneously $\mathcal{O}$, $\mathcal{I}$ and $\mathcal{Q}$, but it does not provide an algorithm for reasoning in full $\mathcal{ZOIQ}$ because Proposition 3.2.4 does not hold for arbitrary $\mathcal{ZOIQ}$ concepts. In fact, the combination of these three constructs causes the loss of the canonical model property and makes reasoning NExpTime-hard already in $\mathcal{ALCIOQ}$, that is, in the absence of number restrictions, self concepts, regular and Boolean role expressions, and role inclusion axioms [Tob00]. $\mathcal{SHOIQ}$, that additionally supports transitive roles and role inclusions is known to be decidable in NExpTime. This follows from an encoding into $\mathcal{C}_2$ [Tob01], together with the NExpTime upper bound for $\mathcal{C}_2$ under unary [PST00] and even binary [PH05] encoding of numbers.

There are also other algorithms for $\mathcal{SHOIQ}$ that do not provide optimal complexity bounds and use, for example, tableau [HS05] and resolution methods [KM08]. It is not clear whether these algorithms can be easily extended to full $\mathcal{ZOIQ}$ (star concepts are the only feature of the $\mathcal{Z}$ family that has not been shown to be expressible in $\mathcal{C}_2$) but their extension to a logic with fixed points is not feasible: as shown in [BP04], $\mathcal{ALCIOQ}$ extended with fixed points is undecidable.

Chapter 4

Reasoning about Queries using Automata

In this chapter we turn to reasoning about queries, and show that automata on infinite trees can also be successfully exploited in this context. We reduce the problem of deciding the entailment of a given P2RPQs over a $\mathcal{ZOIQ}$ knowledge base enjoying the canonical model property, to the emptiness test of an automaton. The technique exploits the algorithm presented in the previous chapter, and builds on the ideas of Calvanese et. al. [CDGLV00] to obtain from it a procedure for query entailment. Roughly, the idea is to first construct an automaton that verifies whether there exists a match for the given query in an interpretation. Then we complement this automaton and intersect it with the automaton that recognizes the models of a KB that we developed in Chapter 3. The result is an automaton whose language is empty iff there is a model of the KB where the query has no match. This technique was employed in [CDGLV00] to decide containment of two queries (w.r.t. an empty KB), and uses automata on infinite words. Here we do a similar construction but using automata on infinite trees, which poses some additional difficulties. In particular, to implement the necessary automata operations, we need to consider a few different kinds of automata and apply transformations between them.

The reduction to automata emptiness allows us to decide the entailment of a P2RPQ over any given $\mathcal{ZOIQ}$ knowledge base enjoying the canonical model property in 2ExpTime. In this way, we do not only push significantly the decidability of query answering in expressive DLs, but also obtain tight complexity bounds. Further, since in the DLs with nominals (and regular role expressions), containment of P2RPQs can be reduced to entailment, we obtain the first decidability and complexity results for query containment. Specifically, we obtain (with unary encoding of numbers), an optimal 2ExpTime upper bound for P2RPQ entailment in $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$, and for P2RPQ containment in $\mathcal{ZOQ}$ and $\mathcal{ZOI}$. For $\mathcal{ZIQ}$, the same bound holds for containment of CQs in P2RPQs. This seems particularly interesting if one considers that P2RPQs allow a form of recursion, and that decidability results for recursive queries are very limited. In fact, this is to our knowledge the first result of this kind for expressive DLs.

The rest of the chapter is organized as follows. The core technique for query answering is presented in Section 4.1. Then, in Section 4.2, we obtain some complexity results from our automata algorithm. Related work and some discussion can be found in Section 4.3.

4.1 Query Entailment via Automata

Throughout this section we assume a given P2RPQ q and a KB $\mathcal{K}$ in $\mathcal{ZOIQ}$, such that $\mathcal{K}$ enjoys the canonical model property (that is, if $\mathcal{K}$ is satisfiable then it has a canonical model). In particular, $\mathcal{K}$ can be any KB in $\mathcal{ZIQ}$, $\mathcal{ZOQ}$ or $\mathcal{ZOI}$. We reduce $\mathcal{K} \not\models q$ to the emptiness test of an automaton. For simplicity, we assume that the set $\mathsf{Atoms}(q)$ of atoms occurring in q contains only role atoms of the form $R(v, v')$ with $v, v' \in \mathsf{N_V}$, and that all vocabulary symbols occurring in q occur also in $\mathcal{K}$, i.e., $\mathsf{N_C}(q) \subseteq \mathsf{N_C}(\mathcal{K})$, $\mathsf{N_R}(q) \subseteq \mathsf{N_R}(\mathcal{K})$, and $\mathsf{N_I}(q) \subseteq \mathsf{N_I}(\mathcal{K})$. These assumptions are without loss of generality (see Section 2.2).

It follows from Proposition 3.2.4 that to decide whether $\mathcal{K} \not\models q$, we only need to decide whether $C_\mathcal{K}$ has a canonical model in which q has no match, where $C_\mathcal{K}$ is the concept from Definition 3.1.2. We build an automaton $\mathbf{A}_{C_\mathcal{K} \not\models q}$ that accepts the representations of canonical models of $C_\mathcal{K}$ in which q has no match. Thus deciding query entailment reduces to testing $\mathbf{A}_{C_\mathcal{K} \not\models q}$ for emptiness. Roughly, $\mathbf{A}_{C_\mathcal{K} \not\models q}$ is obtained by intersecting two automata: the automaton $\mathbf{A}_{C_\mathcal{K}}$ defined in Chapter 3 that accepts canonical models of $C_\mathcal{K}$, and an automaton that accepts interpretations in which q has no match. As a preliminary step, we first construct a FEA $\mathbf{A}_q$ that accepts interpretations in which q has a match. It will then be possible to obtain $\mathbf{A}_{C_\mathcal{K} \not\models q}$ by applying some operations to $\mathbf{A}_q$ and $\mathbf{A}_{C_\mathcal{K}}$.

4.1.1 Representing Query Matches

Recall that q is of the form $q = \exists \vec{v}.\varphi(\vec{v})$, and that $\mathsf{N_V}(q)$ denotes the set of variables in $\vec{v}$. In what follows, we use the term *extended interpretation* to refer to a pair of a canonical interpretation and a binding for q.

Definition 4.1.1 *A* extended interpretation *is a pair $(\mathcal{I}, \pi)$ consisting of a canonical interpretation $\mathcal{I}$ and a binding $\pi : \mathsf{VI}(q) \to \Delta^\mathcal{I}$ for q.*

As in Definition 2.2.4, for $\mathcal{J} = (\mathcal{I}, \pi)$, we write $\mathcal{J} \models q$ if π is a match for q in $\mathcal{I}$.

An extended interpretation can be represented as a labeled forest. The representation is like the one in Definition 3.3.1, but we additionally include the variable v in the label of the node $\pi(v)$, and a marker $v^\downarrow$ in the label of the individual node that is an ancestor of $\pi(v)$ (recall that in a canonical interpretation, every node is a descendant of some node interpreting an individual). Forests representing extended interpretations are labeled with the alphabet $\Sigma_{\mathcal{K},q}$, which enriches $\Sigma_{C_\mathcal{K}}$ with the variables in $\mathsf{N_V}(q)$ and the markers of the form in $v^\downarrow$.

Definition 4.1.2 ($\Sigma_{\mathcal{K},q}$, q-binding forests) *Recall $\Theta(C_\mathcal{K})$ and $\Sigma_\mathcal{K}$ from Definition 3.3.1. We let $\mathsf{N_V}^\downarrow(q) = \{v^\downarrow \mid v \in \mathsf{N_V}(q)\}$, and define*

$$\Theta(\mathcal{K}, q) = \Theta(C_\mathcal{K}) \cup \mathsf{N_V}(q) \cup \mathsf{N_V}^\downarrow(q), \qquad \text{and} \qquad \Sigma_{\mathcal{K},q} = 2^{\Theta(\mathcal{K},q)}.$$

*For a $\Sigma_{\mathcal{K},q}$-labeled forest $\mathbf{F} = \langle F, L\rangle$, its $\Sigma_\mathcal{K}$-*restriction *is the forest $\mathbf{F}' = \langle F, L'\rangle$ that results from restricting the labeling L to the alphabet $\Sigma_\mathcal{K}$. The individual nodes of $\mathbf{F}$ are the ones of $\mathbf{F}'$, and $\mathbf{F}$ is individual unique if $\mathbf{F}'$ is.*

A $\Sigma_{\mathcal{K},q}$-labeled forest $\mathbf{F} = \langle F, L\rangle$ is called q-binding *if*

1. *for each $v \in \mathsf{VI}(q)$ there is exactly one $w \in T$ with $v \in L(w)$, called the* binding of v,
2. *for each marker $v^\downarrow \in \mathsf{N_V}^\downarrow(q)$, there is exactly one $w \in T$ with $v^\downarrow \in L(w)$, and w is an individual node and an ancestor of the binding of v.*

By $\pi_\mathbf{F}$ we denote the function $\mathsf{VI}(q) \to T$ that maps each $v \in \mathsf{VI}(q)$ to its binding.

*If $\mathbf{F}$ is q-binding and individual unique, we call it a $(q, C_\mathcal{K})$-*forest.

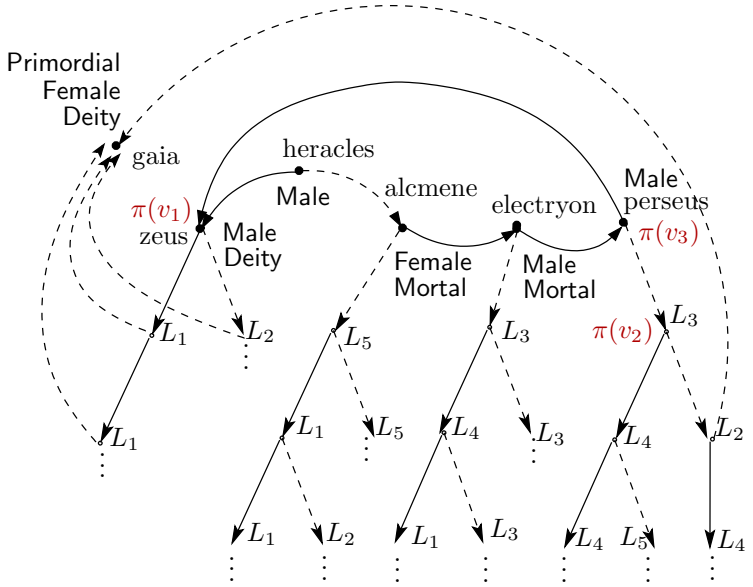

Figure 4.1: A match π for g_q in the canonical model for $\mathcal{K}_g$

Every $(q, C_\mathcal{K})$-forest represents an extended interpretation, and every extended interpretation is represented by some $(q, C_\mathcal{K})$-forest.

Definition 4.1.3 *A $(q, C_\mathcal{K})$-forest* $\mathbf{F}$ represents *the extended interpretation* $\mathcal{J}_\mathbf{F} = (\mathcal{I}_\mathbf{F}, \pi_\mathbf{F})$.

The forest representation *of an extended interpretation $\mathcal{J} = (\mathcal{I}, \pi)$ is the $(q, C_\mathcal{K})$-forest $\mathbf{F}_\mathcal{J} = \langle F, L\rangle$ whose $\Sigma_\mathcal{K}$-restriction is $\mathbf{F}_\mathcal{I}$ and such that*

- *for each $v \in \mathsf{N_V}(q)$ and each $w \in F$, $v \in L(w)$ iff $w = \pi(v)$,*
- *for each $v \in \mathsf{N_V}(q)$ and each $c \in \mathsf{roots}(F)$, $v^\downarrow \in L(c)$ iff $\pi(v)$ is a descendant of c.*

Example 4.1.4 *We consider the following query (is a minor (Boolean) variation of q_1^{theo} from Example 2.2.2).*

$$q_g = \exists v_1, v_2, v_3.\ \mathsf{hasParent}^* \circ \mathsf{hasParent}^{-*}(v_1, v_2) \wedge \mathsf{hasParent}^-(v_1, v_3) \wedge \mathsf{hasParent}^-(v_2, v_3) \wedge \mathsf{Male}(v_1) \wedge \mathsf{Female}(v_2) \wedge (\neg\mathsf{Deity}(v_1) \vee \neg\mathsf{Deity}(v_2))$$

Recall the interpretation $\mathcal{I}_g$ given in Example 3.2.3. The match $\pi(v_1) = \text{zeus}^{\mathcal{I}_g} = 2$, $\pi(v_2) = \text{perseus}^{\mathcal{I}_g} \cdot 1 = 6 \cdot 1$ and $\pi(v_3) = \text{perseus}^{\mathcal{I}_g} = 6$ for q_g in $\mathcal{I}_g$ is depicted in Figure 4.1.

The forest representation of the extended interpretation $(\mathcal{I}_g, \pi)$ is shown in Figure 4.2. It extends the forest representation of Figure 3.3 with the variable names v_1, v_2 and v_3, and the markers $v_1^\downarrow$, $v_2^\downarrow$ and $v_3^\downarrow$.

4.1.2 Recognizing Query Matches using Automata

We now construct a FEA $\mathbf{A}_q$ that verifies whether a $(q, C_\mathcal{K})$-forest represents an interpretation where there is a match for q. More specifically, $\mathbf{A}_q$ accepts an individual unique $\Sigma_{\mathcal{K},q}$-labeled forest $\mathbf{F}$ iff it is q-binding and $\mathcal{J}_\mathbf{F} \models q$. We define $\mathbf{A}_q$ as the intersection of two automata: (i) $\mathbf{A}_V$ accepts an individual unique $\Sigma_{\mathcal{K},q}$-labeled forest iff it is q-binding, that is, if it is a $(q, C_\mathcal{K})$-forest, and (ii) assuming that $\mathbf{F}$ is q-binding, $\mathbf{A}_\pi$ accepts $\mathbf{F}$ iff $\mathcal{J}_\mathbf{F} \models q$.

We first define a FEA $\mathbf{A}_V$ that verifies whether a given individual unique $\Sigma_{\mathcal{K},q}$-labeled forest is q-binding.

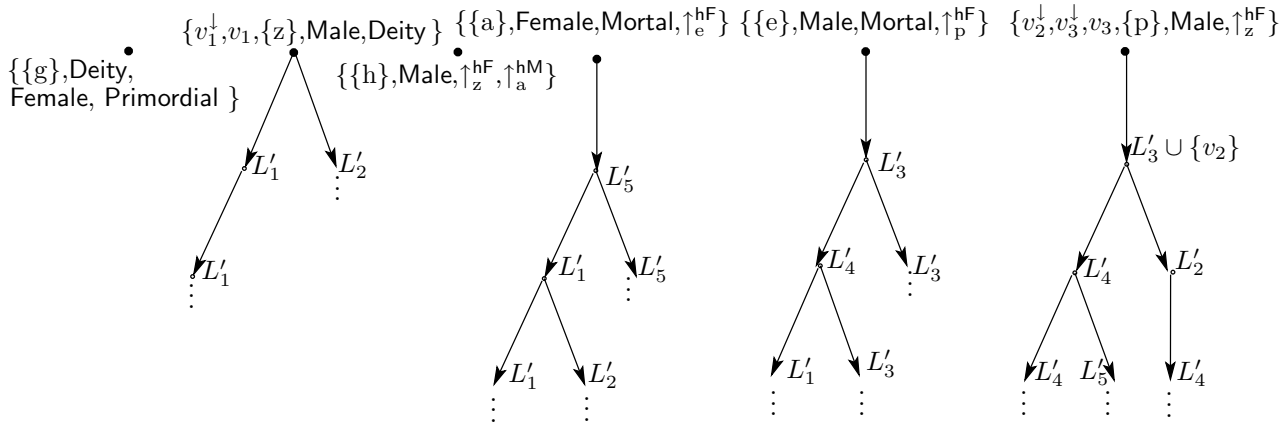

Figure 4.2: The forest representation of an extended interpretation

Definition 4.1.5 *The FEA* $\mathbf{A}_V = \langle \Sigma_V, b_V, Q_V, \delta_V, q_0^V, G_V \rangle$ *is defined as follows:*

- $\Sigma_V = \Sigma_{\mathcal{K},q}$, *that is,* $\mathbf{A}_V$ *runs over* $\Sigma_{\mathcal{K},q}$*-labeled forests.*
- *The counting bound of* $\mathbf{A}_V$ *is* $b_V = 1$.
- *The state set is defined as follows:*

$$\begin{aligned} Q_V \;=\; & \{q_0^V\} \cup \{v^+, v^-, v, \neg v, v^{\downarrow}, \neg v^{\downarrow}, \mid v \in \mathsf{N_V}(q)\} \cup \\ & \{(\!|v^+, v^{\downarrow}|\!), (\!|v^-, \neg v^{\downarrow}|\!) \mid v \in \mathsf{N_V}(q)\} \cup \\ & \{(\!|a, v^{\downarrow}|\!) \mid a \in \mathsf{N_I}(D), v \in \mathsf{N_V}(q)\}. \end{aligned}$$

  The state v^+ *is used for checking that the tree rooted at the current node contains* one *node labeled* v*, and the state* v^- *for checking that it does not contain any node labeled* v*. The states* v*,* $\neg v$*,* $v^{\downarrow}$*, and* $\neg v^{\downarrow}$ *are used simply for verifying whether the respective symbol is in the label of the current node, and the other auxiliary states are used for simultaneously checking some combinations of conditions at the roots. They are explained below.*

- *The transition function* $\delta_V : Q_V \times \Sigma_{\mathcal{K},q} \to \mathcal{B}(\mathbf{D}_1 \times Q_V)$ *is as follows:*

  1. *First, for each* $\sigma \in \Sigma_{\mathcal{K},q}$*, transitions from the initial state:*

$$\begin{aligned} \delta_D(q_D^0, \sigma) \;=\; & \textstyle\bigwedge_{v^{\downarrow} \in \mathsf{N_V}^{\downarrow}(q)} \big(\bigvee_{a \in \mathsf{N_I}(D)} (\langle \mathsf{root} \rangle, (\!|a, v^{\downarrow}|\!)) \big) \wedge \\ & \textstyle\bigwedge_{v^{\downarrow} \in \mathsf{N_V}^{\downarrow}(q)} ([\mathsf{root}], (\!|v^+, v^{\downarrow}|\!)) \vee ([\mathsf{root}], (\!|v^-, \neg v^{\downarrow}|\!)) \end{aligned}$$

     The first big conjunction ensures that each marker $v^{\downarrow}$ *occurs in some individual node, and hence in exactly one individual node if the input forest is nominal unique. The second conjunct ensures that for each root* $c \in \mathsf{roots}(F)$*, either a)* c *is labeled* $v^{\downarrow}$ *and the tree* T_c *rooted at it contains* one *node labeled* v*, or b)* c *is* not *labeled* $v^{\downarrow}$ *and* T_c *does* not *contain a node labeled* v.

  2. *The automaton moves to a state* $(\!|a, v^{\downarrow}|\!)$ *to verify that the label of a root node contains* $\{a\}$ *and* $v^{\downarrow}$*. Hence there is a transition, for each* $\sigma \in \Sigma_{\mathcal{K},q}$ *and each* $(\!|a, v^{\downarrow}|\!) \in Q_V$*:*

$$\delta_V((\!|a, v^{\downarrow}|\!), \sigma) \;=\; (\varepsilon, \{a\}) \wedge (\varepsilon, v^{\downarrow})$$

3. *The automaton moves to a root node c and a state $(\!|v^+, v^{\downarrow}|\!)$ to verify (1a) that c is labeled $v^{\downarrow}$ and the tree T_c rooted at it contains* one *node labeled v. It moves to c and a state $(\!|v^-, \neg v^{\downarrow}|\!)$ to verify that (1b) that c is* not *labeled $v^{\downarrow}$ and T_c does* not *contain a node labeled v. Hence there are transitions, for each $\sigma \in \Sigma_{\mathcal{K},q}$ and each $(\!|v^+, v^{\downarrow}|\!)$, $(\!|v^-, \neg v^{\downarrow}|\!)$ in Q_V:*

$$\begin{aligned} \delta_V((\!|v^+, v^{\downarrow}|\!), \sigma) &= (\varepsilon, v^+) \wedge (\varepsilon, v^{\downarrow}) \\ \delta_V((\!|v^-, \neg v^{\downarrow}|\!), \sigma) &= (\varepsilon, v^-) \wedge (\varepsilon, \neg v^{\downarrow}) \end{aligned}$$

4. *Next, $\mathbf{A}_V$ moves to a node w and a state v^+ to check that there is one node labeled v inside the subtree T_w rooted at w, and to a state v^- to check that are no nodes labeled v in T_w. Hence, for each $v \in \mathsf{N_V}(q)$ and each $\sigma \in \Sigma_{\mathcal{K},q}$, there are transitions:*

$$\begin{aligned} \delta_V(v^+, \sigma) &= \big((\varepsilon, v) \wedge ([0], v^-)\big) \vee \big((\varepsilon, \neg v) \wedge (\langle 1\rangle, v^+) \wedge ([1], v^-)\big), \\ \delta_V(v^-, \sigma) &= (\varepsilon, \neg v) \wedge ([0], v^-). \end{aligned}$$

*Intuitively, if there is exactly one node labeled v in T_w, there are are two possibilities, captured by the two disjuncts in the first transition. Either (*i*) w is labeled v and there are no nodes labeled v anywhere in the subtrees rooted at its successors, or (*ii*) w is not labeled v, it has a child w' such that the subtree rooted at w' contains one node labeled v, and there are no other nodes labeled v below w. The second transition ensures that w itself is not labeled v, and moves to v^- in all children of w to recursively make sure that there are no nodes labeled v in the respective subtrees.*

5. *Finally, for each $\sigma \in \Sigma_{C_\mathcal{K}}$ and each $s \in \mathsf{N_V}(q) \cup \mathsf{N_V}^{\downarrow}(q) \cup \mathsf{N_I}(C_\mathcal{K})$, there are transitions to check whether the symbol s is in the labels of the current node or not:*

$$\delta_V(s, \sigma) = \begin{cases} \textbf{\textit{true}}, & \text{if } s \in \sigma \\ \textbf{\textit{false}}, & \text{if } s \notin \sigma \end{cases}, \quad \delta_V(\neg s, \sigma) = \begin{cases} \textbf{\textit{true}}, & \text{if } s \notin \sigma \\ \textbf{\textit{false}}, & \text{if } s \in \sigma \end{cases}.$$

- $G_V = (\emptyset, \{v^- \mid v \in \mathsf{N_V}(q)\}, Q_V)$.

  Only the states of the form v^+ and v^- can occur infinitely often in the runs. In the former case, if there is some path P where v^+ repeats forever, then the automaton is searching for a node in P that contains the variable v in its label, but never finds it. That is, the variable v is not found in $\mathbf{F}$. In this case the parity condition is violated and the automata does not accept $\mathbf{F}$ because it is not q-binding. On the other hand, infinite runs where only v^- occurs are accepted because they are q-binding: each marker and each variable are found once, and then they do not occur in the rest of the finite and infinite paths of the tree.

The following lemma holds by the construction of $\mathbf{A}_V$ and can be proved in the standard way:

Lemma 4.1.6 *Let $\mathbf{F}$ be an individual unique $\Sigma_{\mathcal{K},q}$-labeled forest. Then $\mathbf{F} \in \mathscr{L}(\mathbf{A}_V)$ iff $\mathbf{F}$ is q-binding.*

To define the FEA $\mathbf{A}_\pi$ that verifies whether a binding is actually a match for q, we first introduce q-concepts. We use the variables in $\mathsf{N_V}(q)$ as atomic concepts, and use a *q-concept* C_α for each query atom $\alpha = R(v, v')$, such that C_α is satisfied at some root of an extended interpretation $(\mathcal{I}, \pi)$ iff $\mathcal{I}, \pi \models \alpha$.

Definition 4.1.7 *A* q-concept *C is defined like a regular $\mathcal{ZOIQ}$ concept, but allows also the elements of $\mathsf{VI}(Q)$ in place of atomic concepts.*

Similarly to standard concepts in standard interpretations, a q-concept C is interpreted in an extended interpretation $\mathcal{J} = (\mathcal{I}, \pi)$ as a set $C^{\mathcal{J}} \subseteq \Delta^{\mathcal{I}}$:

- *if $C = B$ for some atomic $\mathcal{ZOIQ}$ concept B, $C^{\mathcal{J}} = B^{\mathcal{I}}$,*
- *if $C = v$ for some $v \in \mathsf{Vl}(q)$, then $C^{\mathcal{J}} = \{\, \pi(v) \,\}$, and*
- *$\mathcal{J}$ is inductively extended to complex q-concepts as in Definition 2.1.24, i.e., like for a regular interpretation.*

We define a q-concept C_α for each α occurring in q as follows:

Definition 4.1.8 *For each $\alpha = R(v, v')$ in $\mathsf{Atoms}(q)$, let*

$$C_\alpha = \exists R^*_{\mathcal{K}}.(v \sqcap \exists R.v').$$

where $R_{\mathcal{K}} = \bigcup_{p \in \mathsf{N_R}(\mathcal{K})} p \cup p^-$.

The closures $Cl_{\mathsf{C}}(C_\alpha)$ and $Cl_{\mathsf{R}}(C_\alpha)$ of q-concepts are defined as for regular concepts (in particular, $Cl_{\mathsf{C}}(C_\alpha)$ includes the variables v, v' as atomic concepts). We use the following notation:

$$Cl_{\mathsf{C}}(q) = \textstyle\bigcup_{\alpha \in \mathsf{Atoms}(q)} Cl_{\mathsf{C}}(C_\alpha),$$
$$Cl_{\mathsf{R}}(q) = \textstyle\bigcup_{\alpha \in \mathsf{Atoms}(q)} Cl_{\mathsf{R}}(C_\alpha).$$

In an extended interpretation, the satisfaction of a query atom α is correctly captured by the satisfaction of the q-concept C_α at some individual node.

Lemma 4.1.9 *For every extended interpretation $\mathcal{J} = (\mathcal{I}, \pi)$ and atom $\alpha = R(v, v') \in \mathsf{Atoms}(q)$, we have $\mathcal{I}, \pi \models \alpha$ iff there is some $a \in \mathsf{N_I}(C_{\mathcal{K}})$ such that $a^{\mathcal{I}} \in (C_\alpha)^{\mathcal{J}}$.*

By this Lemma, we only need to verify that $C_{\alpha_1}, \ldots, C_{\alpha_k}$ hold at some root for a set of query atoms $\{\alpha_1, \ldots, \alpha_k\}$ that make the query q true. The satisfaction of the concepts C_{α_i} is verified by an automaton $\mathbf{A}_\pi$ that decomposes them via transitions analogous to those of $\mathbf{A}_{C_{\mathcal{K}}}$. Modulo the initial transition from the root node, $\mathbf{A}_\pi$ and $\mathbf{A}_{C_{\mathcal{K}}}$ are very similar.

Definition 4.1.10 *We define the FEA $\mathbf{A}_\pi = (\Sigma_\pi, b_{C_{\mathcal{K}}}, Q_\pi, \delta_\pi, q_0^\pi, G_\pi)$ as follows.*

- *$\Sigma_\pi = \Sigma_{\mathcal{K},q}$.*
- *The counting bound is $b_\pi = \max(\{n \mid \geqslant n\, S.C \in Cl_{\mathsf{C}}(q)\} \cup \{0\})$.*
- *Q_π is described in Table 4.1. It is analogous to Q_D, but it contains one additional state set Q_α and it is defined using $Cl_{\mathsf{C}}(q)$ and $Cl_{\mathsf{R}}(q)$ instead of $Cl_{\mathsf{C}}(D)$ and $Cl_{\mathsf{R}}(D)$, $\Theta(\mathcal{K}, q)$ instead of $\Theta(D)$, and $\mathsf{N_I}(C_{\mathcal{K}})$ instead of $\mathsf{N_I}(D)$.*
- *The transition function $\delta_\pi : Q_\pi \times \Sigma_{\mathcal{K},q} \to \mathcal{B}(\mathbf{D}_{b_\pi} \times Q_\pi)$ contains the following transitions. We group them similarly as for δ_D in Definition 3.3.7, all transitions are analogous except for those in the first group.*

  1. *From the initial state there is a transition for each $\sigma \in \Sigma_{\mathcal{K},q}$:*

  $$\delta_\pi(q_0, \sigma) \quad = \quad B_\varphi$$

  where B_φ results from $\varphi(\vec{v})$ by replacing each atom α with $(\langle \mathsf{root} \rangle, (\!|C_\alpha|\!))$. This transition ensures that a set of atoms $At = \{\alpha_1, \ldots, \alpha_k\}$ that make the query true are satisfied at the individual nodes.

$$
\begin{array}{rcl}
Q_{\alpha} & = & \{(\!|C_{\alpha}|\!) \mid \alpha \in \mathsf{Atoms}(q)\} \\
Q_{\mathsf{Self}} & = & \{S_{\mathsf{Self}} \mid S \in Cl_{\mathsf{R}}(q)\} \\
Q_{\uparrow} & = & \{\uparrow_a^S \mid S \in Cl_{\mathsf{R}}(q), a \in \mathsf{N_I}(C_{\mathcal{K}})\} \\
Q_{\mathsf{roots}} & = & \{(\!|\langle \mathsf{root} \rangle^k, S', C'|\!), (\!|[\mathsf{root}]^k, S', C'|\!), \mid 0 \leq k \leq |\mathsf{N_I}(C_{\mathcal{K}})|, S \in \mathbf{S}, C \in \mathbf{C}\} \\
Q_{\mathsf{Nom}} & = & \{(\!|\vee, \neg a, \neg b|\!) \mid a, b \in \mathsf{N_I}(C_{\mathcal{K}}), a \neq b\} \\
Q_{\mathsf{bin}} & = & \{(\!|\circ, S, C|\!) \mid \circ \in \{\wedge, \vee\}, S \in \mathbf{S}, C \in \mathbf{C}\} \cup \\
 & & \{(\!|\circ, \alpha, C|\!) \mid \circ \in \{\wedge, \vee\}, \alpha = \{a\} \text{ or } \alpha = \neg\{a\}, a \in \mathsf{N_I}(C_{\mathcal{K}}), C \in \mathbf{C}\}
\end{array}
$$

Table 4.1: State set $Q_{\pi} = \{q_{\pi}^0\} \cup Cl_{\mathsf{C}}(q) \cup Cl_{\mathsf{R}}(q) \cup \{s, \neg s \mid s \in \Theta(\mathcal{K}, q)\} \cup Q_{\alpha} \cup Q_{\mathsf{Self}} \cup Q_{\uparrow} \cup Q_{\mathsf{Nom}} \cup Q_{\mathsf{roots}} \cup Q_{\mathsf{bin}}$. Here, $\mathbf{S}$ and $\mathbf{C}$ respectively denote the set of of all simple roles S, $\sim S$ and of all concepts C, $\sim C$ such that $\geqslant n\, S.C \in Cl_{\mathsf{C}}(q)$.

To this aim, when $\mathbf{A}_{\pi}$ is at a root node c and in a state $(\!|C_{\alpha}|\!)$ for some $\alpha \in \mathsf{Atoms}(q)$, it verifies whether c is an individual node and it satisfies C_{α}, via the following transitions for each $\sigma \in \Sigma_{\mathcal{K},q}$:

$$\delta_{\pi}((\!|C_{\alpha}|\!), \sigma) = (\bigvee_{a \in \mathsf{N_I}(C_{\mathcal{K}})} (\varepsilon, \{a\})) \wedge (\varepsilon, C_{\alpha}).$$

Then, to check that C_{α} is satisfied, $\mathbf{A}_{\pi}$ has transitions similar to those of $\mathbf{A}_{C_{\mathcal{K}}}$. That is, for each $\sigma \in \Sigma_{\mathcal{K}}$, there are transitions that:

2. *recursively decompose the concepts using states in $Cl_{\mathsf{C}}(q)$, as in item 2 of δ_D;*
3. *recursively decompose roles using states in $Cl_{\mathsf{R}}(q)$, Q_{Self} and $Q_{\uparrow}$, as in item 3 of δ_D;*
4. *verify the satisfaction of the number restrictions, using the states in Q_{Nom}, Q_{roots} and Q_{bin}, as in item 4 of δ_D; and*
5. *check symbol occurrences in node labels, for each $s \in \Theta(\mathcal{K}, q)$, as in item 5 of δ_D.*

- $G_{\pi} = (\emptyset, \{\forall R^*.C \mid \forall R^*.C \in Cl_{\mathsf{C}}(q)\}, Q_{\pi})$, *analogously as in $\mathbf{A}_D$.*

The automaton correctly verifies whether a given $(q, C_{\mathcal{K}})$-forest represents an extended interpretation where q has a match.

Lemma 4.1.11 *The following hold:*

1. *For every extended interpretation $\mathcal{J}$, if $\mathcal{J} \models q$ then $\mathbf{F}_{\mathcal{J}} \in \mathscr{L}(\mathbf{A}_{\pi})$.*
2. *For every $(q, C_{\mathcal{K}})$-forest $\mathbf{F}$, $\mathbf{F} \in \mathscr{L}(\mathbf{A}_{\pi})$ implies $\mathcal{J}_{\mathbf{F}} \models q$.*

Proof. The proof is very similar to the proof of Proposition 3.3.8. For the first item, we take an extended interpretation $\mathcal{J} = (\mathcal{I}, \pi)$ such that $\mathcal{J} \models q$. By definition, there is a set of atoms $At \subseteq \mathsf{Atoms}(q)$ that make the query true and such that $\mathcal{I}, \pi \models \alpha$ for each $\alpha \in At$. By Lemma 4.1.9, for each α there is some individual $a_{\alpha} \in \mathsf{N_I}(C_{\mathcal{K}})$ such that $(a_{\alpha})^{\mathcal{I}} \in (C_{\alpha})^{\mathcal{I}}$, and this implies that $\mathcal{I}$ is an ind-model of the concept $C_q = \bigsqcap_{\alpha \in At}(\neg\{a_{\alpha}\} \sqcup C_{\alpha})$. We can use a well founded adorned premodel $\langle \mathcal{I}, \theta, \mathsf{ch} \rangle$ of C_q to guide the construction of a run of $\mathbf{A}_{\pi}$ over $\mathbf{F}$, as in the proof of item 1 in Proposition 3.3.8.

For the second item, we consider a $(q, C_{\mathcal{K}})$-forest $\mathbf{F}$. We can show, exactly as for item 1 in Proposition 3.3.8, that in a $\Delta^{\mathcal{I}_{\mathbf{F}}} \times Q_{\pi}$-labeled tree $\langle T_r, r \rangle$ where each node satisfies condition 2

in Definition 3.3.5, whenever there is some $t \in T_r$ with $r(t) = (w, C)$ and $C \in Cl_{\mathsf{C}}(q)$, then $\mathcal{I}_{\mathbf{F}}, w \models_{\mathsf{pre}} C$. Moreover, if C is of the form $\exists R^*.C'$, then it is eventually realized in every accepting run of $\mathbf{A}_\pi$ over $\mathbf{F}$, hence $\mathcal{I}_{\mathbf{F}}, w \models_{\mathsf{pre}} C$ implies $w \in C^{\mathcal{I}_{\mathbf{F}}}$. So, by the initial transitions in item 1 of the definition above, for each $\alpha \in At$ there is some $c \in \mathsf{roots}(\mathcal{I}_{\mathbf{F}})$ and some $a \in \mathsf{N_I}(C_{\mathcal{K}})$ such that $c \in \{a\}^{\mathcal{I}_{\mathbf{F}}}$ and $c \in (C_\alpha)^{\mathcal{I}_{\mathbf{F}}}$, and it follows from Lemma 4.1.9 that $\mathcal{J}_{\mathbf{F}} \models q$. □

Finally, the automaton $\mathbf{A}_q$ is defined as the intersection of $\mathbf{A}_V$ and $\mathbf{A}_\pi$. Building the intersection of two FEAs (with the same alphabet) is trivial: we simply take the maximal counting bound, the disjoint union (denoted $\uplus$) of their state sets and of their transition functions. The parity condition is obtained by taking the disjoint union of the sets with same index, and we add a fresh initial state and a transition from it to the initial states of both automata, for every symbol in the alphabet.

Definition 4.1.12 *The FEA* $\mathbf{A}_q = \langle \Sigma_q, b_q, Q_q, \delta_q, q_0^V, G_q \rangle$ *is defined as follows:*

- $\Sigma_q = \Sigma_{\mathcal{K},q}$.
- *The counting bound of* $\mathbf{A}_q$ *is* $b_q = \max(b_V, b_\pi) = \max(\{n \mid \geqslant n\, S.C \in Cl_{\mathsf{C}}(q)\} \cup \{1\})$.
- *The state set is* $Q_q = \{q_0^q\} \uplus Q_V \uplus Q_\pi$.
- *The transition function* $\delta_q : Q_q \times \Sigma_{\mathcal{K},q} \to \mathcal{B}(\mathbf{D}_{b_q} \times Q_q)$ *is:*

$$\bigcup_{\sigma \in \Sigma_{\mathcal{K},q}} \{\ (q_0^q, \sigma) = (\varepsilon, q_0^V) \wedge (\varepsilon, q_0^\pi)\ \} \ \uplus\ \delta_V \ \uplus\ \delta_\pi$$

- $G_q = (\emptyset, \{v^- \mid v \in \mathsf{N_V}(q)\} \uplus \{\forall R^*.C \mid \forall R^*.C \in Cl_{\mathsf{C}}(q)\}, Q_q)$.

From the construction of $\mathbf{A}_q$ and Lemmas 4.1.6 and 4.1.11, we obtain:

Lemma 4.1.13 *The following hold for* $\mathbf{A}_q$*:*

1. *If* $\mathcal{J}$ *is an extended interpretation and* $\mathcal{J} \models q$*, then* $\mathbf{F}_{\mathcal{J}} \in \mathscr{L}(\mathbf{A}_q)$.
2. *If* $\mathbf{F}$ *is individual unique and* $\mathbf{F} \in \mathscr{L}(\mathbf{A}_q)$*, then* $\mathbf{F}$ *is a* $(q, C_{\mathcal{K}})$*-forest and* $\mathcal{J}_{\mathbf{F}} \models q$.

The following bounds will be important to derive our complexity results. By convention, $||\mathcal{K}, q||$ denotes the combined size $||\mathcal{K}|| + ||q||$ of the strings encoding $\mathcal{K}$ and q. Observe that the alphabet and the set of states of $\mathbf{A}_q$ depend both on $||q||$ and $||\mathcal{K}||$.

Lemma 4.1.14 *For* $\mathbf{A}_q$*, the following hold:*

1. *the number of states* $|Q_q|$ *is polynomial in* $||\mathcal{K}, q||$,
2. *the alphabet size* $|\Sigma_{\mathcal{K},q}|$ *is single exponential in* $||\mathcal{K}, q||$,
3. *if the numbers are encoded in unary, the counting bound* b_D *is linear in* $||q||$*, and*
4. *the parity index of* $\mathbf{A}_q$ *is fixed.*

Proof. For the fourth item, simply observe that the parity index of $\mathbf{A}_q$ is 3. It is also clear that if we assume unary encoding of numbers, then $b_q = \max(\{n \mid \geqslant n\, S.C \in Cl_{\mathsf{C}}(q)\} \cup \{1\})$ is bounded by $||q||$.

The result is then a consequence of the following simple estimates:

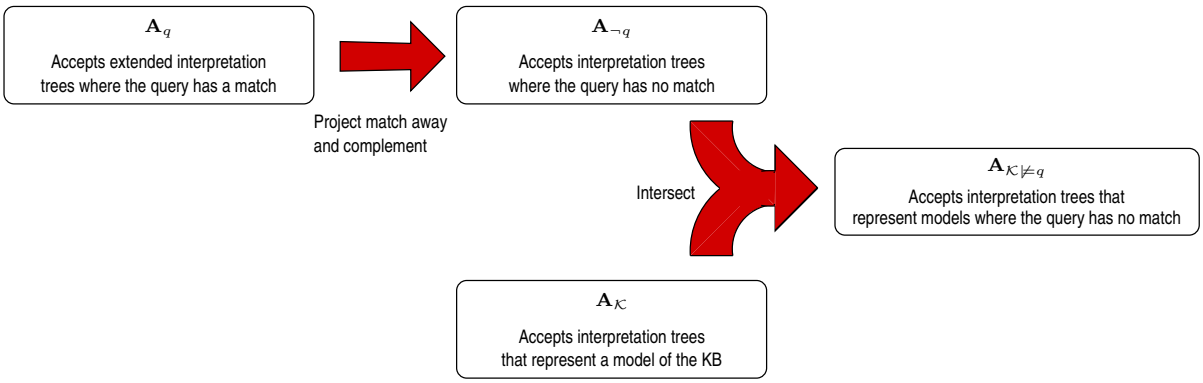

Figure 4.3: Overview of the automata algorithm for Query Entailment

- $|\Sigma_{\mathcal{K},q}| = 2^{|\Theta(\mathcal{K},q)|}$, and $|\Theta(\mathcal{K},q)| = |\Theta(C_\mathcal{K})| + 2 \cdot |\mathsf{N_V}(q)|$. Since $|\Theta(C_\mathcal{K})|$ is quadratic in $||\mathcal{K}||$ (see Lemmas 3.1.5 and 3.4.1) and $\mathsf{N_V}(q)$ is linear in $||q||$, then $|\Theta(\mathcal{K},q)|$ is polynomial (in fact, quadratic) in $||\mathcal{K},q||$, and $|\Sigma_{\mathcal{K},q}|$ is single exponential in $||\mathcal{K},q||$.

- By definition, $|Q_q| = 1 + |Cl_\mathsf{C}(q)| + |Cl_\mathsf{R}(q)| + (2 \times |\Theta(\mathcal{K},q)|) + |Q_\alpha| + |Q_\mathsf{Self}| + |Q_\uparrow| + |Q_\mathsf{Nom}| + |Q_\mathsf{roots}| + |Q_\mathsf{bin}|$, where

$$\begin{aligned}
|Q_\alpha| &= |\mathsf{Atoms}(q)| \\
|Q_\mathsf{Self}| &\leq |Cl_\mathsf{R}(q)| \\
|Q_\uparrow| &\leq |\mathsf{N_I}(C_\mathcal{K})| \cdot |Cl_\mathsf{R}(q)| \\
|Q_\mathsf{Nom}| &= |\mathsf{N_I}(C_\mathcal{K})|^2 \\
|Q_\mathsf{roots}| &\leq 2 \cdot |\mathsf{N_I}(C_\mathcal{K})| \cdot |Cl_\mathsf{C}(q)|^2 \\
|Q_\mathsf{bin}| &\leq 2 \cdot |Cl_\mathsf{C}(q)|^2 + 2 \cdot |\mathsf{N_I}(C_\mathcal{K})| \cdot |Cl_\mathsf{C}(q)|
\end{aligned}$$

  We know that $|\Theta(\mathcal{K},q)|$ is quadratic in $||\mathcal{K},q||$, and that $\mathsf{N_I}(C_\mathcal{K})$ is linearly bounded by $||\mathcal{K}||$. Clearly $\mathsf{Atoms}(q)$, $Cl_\mathsf{C}(q)$, $Cl_\mathsf{R}(q)$ are linear in $||q||$. Therefore $|Q_q|$ is polynomial in $||\mathcal{K},q||$.

□

4.1.3 Reducing Query Entailment to Automata Emptiness

The automata $\mathbf{A}_q$ we just defined and the automata $\mathbf{A}_{C_\mathcal{K}}$ from Chapter 3 are the main ingredients for building $\mathbf{A}_{C_\mathcal{K} \not\models q}$. Roughly, the automaton $\mathbf{A}_q$ accepts the representations of models where there is a match for q. It actually accepts forests labeled over the extended alphabet $\Sigma_{\mathcal{K},q}$ that uses the variables in the node labels to represent matches explicitly. By projecting the query variables from $\mathbf{A}_q$'s alphabet, we obtain an automaton that accepts the same forests, but restricted to the alphabet $\Sigma_{C_\mathcal{K}}$. They represent the interpretations in which q has a match, no matter where it is. We complement this automaton to obtain an automaton $\mathbf{A}_{\neg q}$ that accepts an interpretation exactly when there is no match for q in it. Finally, we intersect $\mathbf{A}_{C_\mathcal{K}}$ and $\mathbf{A}_{\neg q}$ to obtain $\mathbf{A}_{C_\mathcal{K} \not\models q}$. Figure 4.3 gives a general overview of the query entailment algorithm. Each of the steps will be discussed in detail below.

The special features of enriched automata have been useful so far, allowing us to avoid more complicated automata constructions. However, the operations that we use to obtain $\mathbf{A}_{C_\mathcal{K} \not\models q}$ from

$\mathbf{A}_q$ and $\mathbf{A}_{C_\mathcal{K}}$ work on simpler automata models. We first get rid of root and graded transitions, to rely on automata for which more results are available in the literature. We also need to get rid of two-wayness and alternation, as projection and complementation are only possible in one-way non-deterministic automata. We now introduce three increasingly simple automata models that will be used in the construction of $\mathbf{A}_{C_\mathcal{K} \not\models q}$.

In what follows, we refer to each component Σ, Q, b, etc. of an automaton $\mathbf{A}$ by $\Sigma(\mathbf{A})$, $Q(\mathbf{A})$, $b(\mathbf{A})$ etc., and we denote its parity index by $\mathsf{ind}(\mathbf{A})$.

Two-way Graded Alternating Tree Automata (2GATA)

First, we exploit the fact that root transitions can be easily removed from FEAs. A 2GATA is defined exactly as FEA, but there are no transitions using $\langle \mathsf{root} \rangle$ or $[\mathsf{root}]$.

Definition 4.1.15 ([BLMV08]) *For $b > 0$, let $\mathbf{D'}_b = \{-1, \varepsilon\} \cup \{\langle 0 \rangle, \dots, \langle b \rangle\} \cup \{[0], \dots, [b]\}$. A* two-way graded alternating parity tree automaton (2GATA) *is a FEA with transition function $\delta : Q \times \Sigma \to \mathcal{B}(\mathbf{D'}_b \times Q)$,*

2GATAs run on proper trees (that is, trees with root ε), rather than on forests. Their runs are defined essentially as for FEAs, and they always start at the root ε.

Definition 4.1.16 (run, acceptance) *A* run *of a 2GATA $\mathbf{A} = \langle \Sigma, b, Q, \delta, q_0, G \rangle$ over a labeled tree $\langle T, L \rangle$ is a $T \times Q$-labeled tree $\langle T_r, r \rangle$ such that*

1. *$r(c) = (\varepsilon, q_0)$ for the root c of T_r, and*
2. *for every $t \in T_r$ with $r(t) = (w, q)$ there is some $W \subseteq \mathbf{D'}_b \times Q$ such that W makes $\delta(q, L(w))$ true and, for all $(d, q') \in W$:*
   - *if $d \in \{-1, \varepsilon\}$, then $w{\cdot}d$ is a node in T and there is some $j \in \mathbb{N}$ such that $t{\cdot}j \in T_r$ and $r(t{\cdot}j) = (w{\cdot}d, q')$;*
   - *if $d = \langle n \rangle$, then there is some $M \subseteq \mathsf{succ}(w)$ with $|M| > n$ such that, for each $z \in M$, there is some $j \in \mathbb{N}$ such that $t{\cdot}j \in T_r$ and $r(t{\cdot}j) = (z, q')$; and*
   - *if $d = [n]$, then there is some $M \subseteq \mathsf{succ}(w)$ with $|M| \leq n$ such that, for each $z \in \mathsf{succ}(w) \setminus M$, there is some $j \in \mathbb{N}$ such that $t{\cdot}j \in T_r$ and $r(t{\cdot}j) = (z, q')$.*

As for FEAs, the run $\langle T_r, r \rangle$ is accepting *if, for each infinite path P of T_r, there is an* even *i such that $\mathsf{Inf}(\langle P, r \rangle) \cap G_i \neq \emptyset$ and $\mathsf{Inf}(\langle P, r \rangle) \cap G_{i-1} = \emptyset$. A 2GATA $\mathbf{A}$* accepts *a labeled tree $\langle T, L \rangle$ if there exists an accepting run of $\mathbf{A}$ over $\langle T, L \rangle$. The set $\mathscr{L}(\mathbf{A})$ of all trees accepted by $\mathbf{A}$ is the* language of $\mathbf{A}$.

From FEAs to 2GATAs

To recognize forests using 2GATAs, we transform them into trees by adding a suitable root.

Definition 4.1.17 *For $\Sigma = \Sigma_\mathcal{K}$ or $\Sigma = \Sigma_{\mathcal{K},q}$, the* tree encoding *of a Σ-labeled forest $\mathbf{F} = \langle F, L \rangle$, denoted $\mathsf{tree}(\mathbf{F})$, is the $(\Sigma \uplus \{\mathrm{root}\})$-labeled tree $\langle T, L' \rangle$ such that $T = \{\varepsilon\} \uplus F$, $L'(w) = L(w)$ for all $w \neq \varepsilon$, and $L'(\varepsilon) = \{\mathrm{root}\}$.*

We say that a $\Sigma_\mathcal{K}$-labeled tree $\mathbf{T}$ is individual unique *iff $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ for an individual unique $\Sigma_\mathcal{K}$-labeled forest $\mathbf{F}$, that is, if $L(\varepsilon) = \{\mathrm{root}\}$ and removing ε from it results in an individual unique $\Sigma_\mathcal{K}$-labeled forest.*

As shown in [BLMV08], every FEA $\mathbf{A}$ can be reduced to a 2GATA $\mathbf{A'}$ that accepts a tree iff it is the tree encoding of a forest accepted by $\mathbf{A}$.

Lemma 4.1.18 ([BLMV08]) *For each FEA* $\mathbf{A}$*, it is possible to construct a 2GATA* $\mathbf{A}'$ *such that*

1. $\mathbf{T} \in \mathscr{L}(\mathbf{A}')$ *iff* $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ *for some* $\mathbf{F} \in \mathscr{L}(\mathbf{A})$*, and*
2. $\mathbf{A}'$ *and* $\mathbf{A}$ *have the same parity index and counting bound, and* $|Q(\mathbf{A}')|$ *is linearly bounded in* $|Q(\mathbf{A})|$*.*

Two-way Alternating Tree Automata (2ATA)

Now we define the *two-way alternating parity tree automata* (2ATAs) over infinite trees introduced in [Var98]. A 2ATA is similar to a FEA, but it runs on trees of a fixed arity b, and instead of having graded transitions with $\langle n \rangle$ and $[n]$ that allow it to move to at least n or all but n successors of the current node w, it has transitions with directions $d \in \{1, \ldots, b\}$ that move to specific successors $w{\cdot}d$ of w.

Definition 4.1.19 *A* two-way alternating parity tree automaton *(2ATA) running over infinite* Σ*-labeled* k*-trees is a tuple* $\mathbf{A} = \langle \Sigma, Q, \delta, q_0, G \rangle$*, where the alphabet* Σ*, the set of states* Q*, the initial state* $q_0 \in Q$*, and the (parity) acceptance condition* $G = (G_1, \ldots, G_n)$ *with* $G_1 \subseteq G_2 \subseteq \cdots \subseteq G_n = Q$ *are as in FEAs (cf. Definition 3.3.4), and the transition function is* $\delta : Q \times \Sigma \to \mathcal{B}([-1..k] \times Q)$*, with* $[-1..k] = \{-1, \varepsilon, 1, \ldots, k\}$*.*

As usual, acceptance of 2ATAs is defined in terms of runs. They are similar to the runs of 2GATAs, but defined for proper trees of fixed arity rather than for arbitrary proper trees. A run for a 2ATA starts at the root. The directions -1 and ε in $[-1..k]$ indicate that the automaton moves up or stays at the current position, as for FEAs. A direction $d \in \{1, \ldots, k\}$ indicates that the automaton moves from the current node w to its child $w{\cdot}d$. Formally, 2ATA runs are defined as follows:

Definition 4.1.20 *Let* $\mathbf{A} = \langle \Sigma, Q, \delta, q_0, G \rangle$ *be a 2ATA running over* k*-ary trees. A* run (T_r, r) *of* $\mathbf{A}$ *over a* Σ*-labeled* k*-ary tree* (T, L) *is a* $(T \times Q)$*-labeled tree satisfying:*

1. $r(c) = (\varepsilon, q_0)$ *for the root* c *of* T_r*, and*
2. *for every* $t \in T_r$ *with* $r(t) = (w, q)$ *there is some* $W \subseteq [-1..k] \times Q$ *such that* W *makes* $\delta(q, L(w))$ *true and, for all* $(d, q') \in W$*,* $w{\cdot}d$ *is a node in* T *and there is some* $j \in \mathbb{N}$ *such that* $t{\cdot}j \in T_r$ *and* $r(t{\cdot}j) = (w{\cdot}d, q')$*.*

The notion of acceptance and the language of a 2ATA are defined as usual.

From 2GATAs to 2ATAs

We now show how to eliminate graded transitions from a 2GATA and transform it into a 2ATA. Since 2ATAs can only accept k-ary trees for a fixed k, we define the following notion:

Definition 4.1.21 *Given a proper tree* $\mathbf{T} = \langle T, L \rangle$ *and a bound* k *on its branching degree, we denote by* $\mathsf{ary}_k(\mathbf{T})$ *the* k*-ary tree* $\mathbf{T}' = \langle T', L' \rangle$ *obtained from* $\mathbf{T}$ *as follows. First, for each* $w \in T$*, let* f_w *be the bijection from* $\mathsf{succ}(w)$ *to* $\{1, \ldots, |\mathsf{succ}(w)|\}$ *such that* $f_w(w \cdot i) \leq f_w(w \cdot j)$ *whenever* $i \leq j$*. To define* $\mathbf{T}' = \langle T', L' \rangle$*, we build* T' *inductively using a partial function* $g : T' \to T$ *as follows:*

- *For the root* ε *of* T'*,* $g(\varepsilon) = \varepsilon$*.*
- *Each node* $w \in T'$ *has successors* $w \cdot 1, \ldots, w \cdot k$*. If* $g(w)$ *is defined and there is some* $w' \in \mathsf{succ}(g(w))$ *with* $f_{g(w)}(w') = i$*, then* $g(w \cdot i) = w'$*. Otherwise,* $g(w \cdot i)$ *is undefined.*

For every $w \in T'$, $L'(w) = L(g(w))$ if $g(w)$ is defined, and $L(w) = \emptyset$ otherwise.

For a forest $\mathbf{F}$, we use $\mathsf{tree}^b(\mathbf{F})$ as a shortcut $\mathsf{ary}_b(\mathsf{tree}(\mathbf{F}))$.

For every k, a 2GATA $\mathbf{A}$ can be transformed into a 2ATA $\mathbf{A}'$ that accepts exactly each k-ary tree encoding a tree of branching bounded by k that is accepted by $\mathscr{L}(\mathbf{A}')$.

Lemma 4.1.22 *Let $k \geq 1$. For each 2GATA $\mathbf{A}$ there is a 2ATA $\mathbf{A}^k$ such that*

1. *for every tree $\mathbf{T}$, $\mathbf{T} \in \mathscr{L}(\mathbf{A}^k)$ implies $\mathbf{T} \in \mathscr{L}(\mathbf{A})$,*
2. *for every tree $\mathbf{T} \in \mathscr{L}(\mathbf{A})$ whose branching degree is bounded by k, $\mathsf{ary}_k(\mathbf{T}) \in \mathscr{L}(\mathbf{A}^k)$, and*
3. *$\mathbf{A}$ and $\mathbf{A}^k$ have the same parity index and counting bound, and $|Q(\mathbf{A}^k)|$ is linearly bounded in $|Q(\mathbf{A})| \cdot b(\mathbf{A}) \cdot k$.*

Proof (sketch). Let $\mathbf{A} = \langle \Sigma, b, Q, \delta, q_0, G\rangle$. Then we define $\mathbf{A}^k = \langle \Sigma, Q \uplus Q', \delta', q_0, G'\rangle$, where

$$Q'=\{\langle i,q,j\rangle, [i,q,j] \mid q \in Q, 0 \leq i \leq b{+}1, 1 \leq j \leq k{+}1\}.$$

For each $\sigma \in \Sigma$, the transition function δ' is defined as follows. First, for all $q \in Q$, $\delta'(q,\sigma)$ is obtained from $\delta(q,\sigma)$ by replacing each $(\langle n\rangle, q)$ with $(\varepsilon, \langle n{+}1, q, 1\rangle)$ and each $([n], q)$ with $(\varepsilon, [n{+}1, q, 1])$. For $1{\leq}i{\leq}b{+}1$ and $1{\leq}j{\leq}k$, we define:

$$\begin{aligned}\delta'(\langle i,q,j\rangle,\sigma) &= ((j,q) \wedge (\varepsilon, \langle i{-}1, q, j{+}1\rangle)) \vee (\varepsilon, \langle i, q, j{+}1\rangle)\\ \delta'([i,q,j],\sigma) &= ((j,q) \wedge (\varepsilon, [i, q, j{+}1])) \vee (\varepsilon, [i{-}1, q, j{+}1])\end{aligned}$$

and additionally, we have:

$$\begin{aligned}\delta'(\langle 0,q,j\rangle,\sigma) &= \mathbf{true}, \quad \delta'([0,q,j],\sigma) = \mathbf{false}, \text{ for } 1 \leq j \leq k{+}1;\\ \delta'(\langle i,q,k{+}1\rangle,\sigma) &= \mathbf{false}, \ \delta'([i,q,k{+}1],\sigma) = \mathbf{true}, \text{ for } 1 \leq i \leq b{+}1.\end{aligned}$$

Intuitively, from state $\langle i,q,j\rangle$, a copy of $\mathbf{A}^k$ is sent off in state q, to at least i successor nodes starting from the j-th one. Similarly, from state $[i,q,j]$, no copy of $\mathbf{A}^k$ is sent off in state q, for at most $i{-}1$ successor nodes starting from the j-th one. Finally, if $G = (G_1, \ldots, G_{n-1}, Q)$, we have

$$G' = (G_1, \ldots, G_{n-1}, Q \uplus Q').$$

One can easily show that every tree $\mathbf{T}$ accepted by $\mathbf{A}^k$ is accepted by $\mathbf{A}$, and that if the branching degree of $\mathbf{T}$ is bounded by k and $\mathbf{T}$ is accepted by $\mathbf{A}$, then its k-ary version $\mathsf{ary}_k(\mathbf{T})$ is accepted by $\mathbf{A}^k$. □

Lemma 4.1.22 does not ensure equivalence between the 2GATA and the resulting 2ATA, since the 2GATA accepts arbitrary proper trees and the 2ATA only k-ary ones, but we will see that this is sufficient for our purposes.

One-way Non-deterministic Tree Automata (1NTA)

Standard non-deterministic (one-way) automata on infinite trees can be defined as particular 2ATAs, in which the transitions cannot use the directions 0 and -1, but instead the automaton always moves to the k successors of the current node and to a tuple of k states, one for each successor. Such a choice can be expressed as a formula in conjunctive form:

Definition 4.1.23 *A* one-way non-deterministic automaton *(1NTA) running over k-ary trees is a 2ATA $\mathbf{A} = \langle \Sigma, Q, \delta, q_0, G\rangle$ such that for every $q \in Q$ and every $\sigma \in \Sigma$, $\delta(q,\sigma)$ is of the form $((1, q_1^1) \wedge \cdots \wedge (k, q_k^1)) \vee \cdots \vee ((1, q_1^j) \wedge \cdots \wedge (k, q_k^j))$, with $j \geq 0$, and $q_\ell^i \in Q$ for each $1 \leq i \leq j$ and each $1 \leq \ell \leq k$.*

From 2ATAs to 1NTAs

The following result is well-known.

Theorem 4.1.24 ([Var98]) *Given a 2ATA* $\mathbf{A}$ *running on* k*-ary trees, it is possible to construct a 1NTA* $\mathbf{A}'$ *such that*

1. $\mathscr{L}(\mathbf{A}') = \mathscr{L}(\mathbf{A})$,
2. $|Q(\mathbf{A}')|$ *is single exponential in* $|Q(\mathbf{A})|$, *and*
3. $\mathsf{ind}(\mathbf{A}')$ *is linear in* $\mathsf{ind}(\mathbf{A})$.

Constructing the Automaton $\mathbf{A}_{\mathcal{K} \not\models q}$

Now we can transform $\mathbf{A}_{C_\mathcal{K}}$ and $\mathbf{A}_q$ into 1NTAs $\mathbf{A}^1_\mathcal{K}$ and $\mathbf{A}^1_q$, and rely on known results for 1NTAs to build $\mathbf{A}_{\mathcal{K} \not\models q}$. To eliminate the graded transitions we need a bound on the branching degree of forests representing (extended) interpretations, which is $k(\mathcal{K}) = \max(\mathsf{N_I}(C_\mathcal{K}), b_{C_\mathcal{K}})$ where $b_{C_\mathcal{K}} = |Cl_\mathsf{C}(C_\mathcal{K})| \cdot \max(\{n \mid \geqslant n\, S.C \in Cl_\mathsf{C}(C_\mathcal{K})\} \cup \{0\})$.

Lemma 4.1.25 *There is a 1NTA* $\mathbf{A}^1_\mathcal{K}$ *such that, assuming unary encoding of numbers,*

1. *if* $\mathcal{I}$ *is a canonical model of* $C_\mathcal{K}$, *then* $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_\mathcal{I}) \in \mathscr{L}(\mathbf{A}^1_\mathcal{K})$,
2. *if* $\mathbf{T} \in \mathscr{L}(\mathbf{A}^1_\mathcal{K})$, *then* $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ *for some* $C_\mathcal{K}$*-forest* $\mathbf{F}$ *such that* $\mathcal{I}_\mathbf{F} \models_{\mathsf{ind}} C_\mathcal{K}$,
3. *the number of states* $|Q(\mathbf{A}^1_\mathcal{K})|$ *is single exponential in* $||\mathcal{K}||$,
4. *the parity index* $\mathsf{ind}(\mathbf{A}^1_\mathcal{K})$ *is fixed, and*
5. $\mathbf{A}^1_\mathcal{K}$ *can be built in time single exponential in* $||\mathcal{K}||$,

Proof. Recall the automaton $\mathbf{A}_{C_\mathcal{K}}$ from Definition 3.3.7. By Lemma 4.1.18, there is a 2GATA $\mathbf{A}^\diamond_\mathcal{K}$ that accepts the forest encoding of each tree accepted by $\mathbf{A}_{C_\mathcal{K}}$. It has the same parity index and counting bound as $\mathbf{A}_{C_\mathcal{K}}$, and its state set $Q(\mathbf{A}^\diamond_\mathcal{K})$ is linearly bounded in $|Q_{C_\mathcal{K}}|$. Since $|Q_{C_\mathcal{K}}|$ is polynomial in $||\mathcal{K}||$, so is $Q(\mathbf{A}^\diamond_\mathcal{K})$. To transform $\mathbf{A}^\diamond_\mathcal{K}$ into a 2ATA, we consider $k(\mathcal{K})$ as above and take $\mathbf{A}^{k(\mathcal{K})}_q$ as constructed in Lemma 4.1.22. The parity index does not change. Since $k(\mathcal{K})$ is linearly bounded in $||\mathcal{K}||$, and $|Q(\mathbf{A}^{k(\mathcal{K})}_q)|$ is linearly bounded in $|Q(\mathbf{A}^\diamond_\mathcal{K})| \cdot b_{C_\mathcal{K}} \cdot k(\mathcal{K})$, then $|Q(\mathbf{A}^{k(\mathcal{K})}_q)|$ is polynomial in $||\mathcal{K}||$.

Since $\mathbf{A}^\diamond_\mathcal{K}$ accepts trees that are not individual unique, $\mathbf{A}^{k(\mathcal{K})}_\mathcal{K}$ also accepts trees that are not individual unique, and that do not properly represent a model of $C_\mathcal{K}$. To avoid distinguishing between individual unique and non individual unique trees as we did in Chapter 3, and having to prove the existence of the latter in the language of $\mathbf{A}^{k(\mathcal{K})}_\mathcal{K}$ and of other automata below, we exploit the fact that 2ATAs can easily enforce individual uniqueness, and transform $\mathbf{A}^{k(\mathcal{K})}_\mathcal{K}$ into an automaton $\mathbf{A}^u_\mathcal{K}$ that accepts exactly the nominal unique trees accepted by $\mathbf{A}^{k(\mathcal{K})}_\mathcal{K}$. We only need to add a new initial state q^u_0, and states a and $\neg a$ for each $a \in \mathsf{N_I}(C_\mathcal{K})$. Then we add a new initial transition from the root that checks that each individual only occurs at one level one node, and then executes the original transition of $\mathbf{A}^{k(\mathcal{K})}_\mathcal{K}$, and add transitions that check for the individual names at the labels (note that states a, $\neg a$ and the latter transitions were already defined in the automaton $\mathbf{A}_{C_\mathcal{K}}$ already, but they can always be added as (transitions from) fresh states). Let $\mathbf{A}^{k(\mathcal{K})}_\mathcal{K} = \langle \Sigma, Q, \delta, q_0, G\rangle$. Formally, we let $\mathbf{A}^u_\mathcal{K} = \langle \Sigma, Q_u, \delta_u, q^u_0, G_u\rangle$, where $Q_u = Q \uplus \{q^u_0\} \uplus \{a, \neg a \mid a \in \mathsf{N_I}(C_\mathcal{K})\}$. The transition function δ_u coincides with δ for the states in Q, and extends it with the following transitions:

$$\delta_u(q^u_0, \{\text{root}\}) = (\varepsilon, q_0) \wedge \bigwedge_{a \in \mathsf{N_I}(C_\mathcal{K})} \Big(\bigvee_{1 \leq i \leq k} ((i, a) \wedge \bigwedge_{1 \leq j \leq k, j \neq i} (j, \neg a))\Big)$$

and transitions, for each $\sigma \in \Sigma$:

$$\delta_u(a,\sigma) \;=\; \begin{cases} \mathbf{true}, & \text{if } a \in \sigma \\ \mathbf{false}, & \text{if } a \notin \sigma \end{cases} \quad , \quad \delta_u(\neg a,\sigma) \;=\; \begin{cases} \mathbf{true}, & \text{if } a \notin \sigma \\ \mathbf{false}, & \text{if } a \in \sigma \end{cases} .$$

If $G = (G_1, \dots, G_n = Q)$, we set $G_u = (G_1, \dots, G'_n = Q_u)$. Then $\mathbf{A}^u_{\mathcal{K}}$ accepts a tree $\mathbf{T}$ iff $\mathbf{T}$ is nominal unique and accepted by $\mathbf{A}^{k(\mathcal{K})}_{\mathcal{K}}$. The parity index of $\mathbf{A}^u_{\mathcal{K}}$ remains the same, and $|Q(\mathbf{A}^u_{\mathcal{K}})|$ is bounded by $|Q(\mathbf{A}^{k(\mathcal{K})}_{\mathcal{K}})| + 2 \cdot |\mathsf{N_I}(C_{\mathcal{K}})| + 1$, and is thus polynomial in $||\mathcal{K}||$.

Finally, let $\mathbf{A}^1_{\mathcal{K}}$ be a 1NTA that accepts the same trees as $\mathbf{A}^u_{\mathcal{K}}$, as described in Theorem 4.1.24. Then $|Q(\mathbf{A}^1_{\mathcal{K}})|$ is exponential in $||\mathcal{K}||$. Since the parity index of $\mathbf{A}^u_{\mathcal{K}}$ is fixed, so is the parity index of $\mathbf{A}^1_{\mathcal{K}}$. Each of the steps can be done in time single exponential in $||\mathcal{K}||$ (we note that the transformation from FEAs to 2GATAs described in [BLMV08] can be computed efficiently), so it is only left to show 2 and 1.

To show 1, consider a canonical model $\mathcal{I}$ of $C_{\mathcal{K}}$. By item 1 in Proposition 3.3.8, its forest encoding $\mathbf{F}_{\mathcal{I}}$ is accepted by $\mathbf{A}_{C_{\mathcal{K}}}$, and then by Lemma 4.1.18, $\mathsf{tree}(\mathbf{F}_{\mathcal{I}})$ is accepted by $\mathbf{A}^{\diamond}_{\mathcal{K}}$. Since the branching of $\mathbf{F}_{\mathcal{I}}$ is bounded by $k(\mathcal{K})$, Lemma 4.1.22 shows that $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_{\mathcal{I}}) \in \mathscr{L}(\mathbf{A}^{k(\mathcal{K})}_{\mathcal{K}})$, and since $\mathbf{F}_{\mathcal{I}}$ is individual unique, $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_{\mathcal{I}}) \in \mathscr{L}(\mathbf{A}^u_{\mathcal{K}})$. Finally, by Theorem 4.1.24, $\mathsf{tree}^{k}(\mathbf{F}_{\mathcal{I}}) \in \mathscr{L}(\mathbf{A}^1_{\mathcal{K}})$ as desired.

To show 2, if a tree $\mathbf{T}$ is accepted by $\mathbf{A}^1_{\mathcal{K}}$, then by Theorem 4.1.24 it is accepted by $\mathbf{A}^u_{\mathcal{K}}$. Hence it is individual unique and accepted by $\mathbf{A}^{k(\mathcal{K})}_{\mathcal{K}}$. So by Lemma 4.1.22 it is also accepted by $\mathbf{A}^{\diamond}_{\mathcal{K}}$. By Lemma 4.1.18, this implies that it $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ for some $\mathbf{F}$ accepted by $\mathbf{A}_{C_{\mathcal{K}}}$. Since $\mathbf{T}$ is individual unique then so is $\mathbf{F}$, and hence $\mathbf{F}$ is a $C_{\mathcal{K}}$-forest and $\mathcal{I}_{\mathbf{F}} \models_{\mathsf{ind}} C_{\mathcal{K}}$ by item 2 in Proposition 3.3.8. □

And similarly for the query automaton $\mathbf{A}_q$:

Lemma 4.1.26 *There is a 1NTA $\mathbf{A}^1_q$ such that, assuming unary encoding of numbers,*

1. *if $\mathcal{J}$ is an extended interpretation and $\mathcal{J} \models q$, then $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_{\mathcal{J}}) \in \mathscr{L}(\mathbf{A}^1_q)$,*
2. *if $\mathbf{T}$ is individual unique and $\mathbf{T} \in \mathscr{L}(\mathbf{A}^1_q)$, then $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ for some $(q, C_{\mathcal{K}})$-forest $\mathbf{F}$ such that $\mathcal{J}_{\mathbf{F}} \models q$,*
3. *the number of states $|Q_q|$ is single exponential in $||\mathcal{K}, q||$,*
4. *the parity index $\mathsf{ind}(\mathbf{A}_q)$ is fixed, and*
5. *$\mathbf{A}^1_q$ can be built in time single exponential in $||\mathcal{K}, q||$.*

Proof. The proof is analogous. We start from the automaton $\mathbf{A}_q$ from Definition 4.1.12. By Lemma 4.1.18, there is a 2GATA $\mathbf{A}^{\diamond}_q$ that accepts the forest encoding of each tree accepted by $\mathbf{A}_q$. It has the same parity index and counting bound as $\mathbf{A}_q$, and its state set $Q(\mathbf{A}^{\diamond}_q)$ is linearly bounded in $|Q_q|$. Since $|Q_q|$ is polynomial in $||\mathcal{K}, q||$, so is $Q(\mathbf{A}^{\diamond}_q)$. Then we take $\mathbf{A}^{k(\mathcal{K})}_q$ as in Lemma 4.1.22. The parity index remains the same. Since $k(\mathcal{K})$ is linearly bounded in $||\mathcal{K}, q||$, and $|Q(\mathbf{A}^{k(\mathcal{K})}_q)|$ is linearly bounded in $|Q(\mathbf{A}^{\diamond}_q)| \cdot b_q \cdot k(\mathcal{K})$, $|Q(\mathbf{A}^{k(\mathcal{K})}_q)|$ is polynomial in $||\mathcal{K}, q||$. Finally, $\mathbf{A}^1_q$ is a 1NTA that accepts the same trees as $\mathbf{A}^{k(\mathcal{K})}_q$ as in Theorem 4.1.24; $|Q(\mathbf{A}^1_q)|$ is exponential in $||\mathcal{K}, q||$, and the parity index of $\mathbf{A}^1_q$ remains fixed. As above, each of the steps can be done in time single exponential in $||\mathcal{K}, q||$.

To show 1, consider an extended interpretation $\mathcal{J}$ such that $\mathcal{J} \models q$. By Lemma 4.1.13, its forest encoding $\mathbf{F}_{\mathcal{J}}$ is accepted by $\mathbf{A}_q$ and then by Lemma 4.1.18, $\mathsf{tree}(\mathbf{F}_{\mathcal{J}})$ is accepted by $\mathbf{A}^{\diamond}_q$. Since $\mathbf{F}_{\mathcal{J}}$ is individual unique, the branching of $\mathsf{tree}(\mathbf{F}_{\mathcal{J}})$ is bounded by $k(\mathcal{K})$, and we have $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_{\mathcal{J}}) \in \mathscr{L}(\mathbf{A}^{k(\mathcal{K})}_q)$ by Lemma 4.1.22. So, by Theorem 4.1.24, $\mathsf{tree}^{k(\mathcal{K})}(F_{\mathcal{K}}) \in \mathscr{L}(\mathbf{A}^1_q)$.

To show 2, if an individual unique $\Sigma_{\mathcal{K},q}$-labeled tree $\mathbf{T}$ is accepted by $\mathbf{A}^1_q$, then it is accepted by $\mathbf{A}^{k(\mathcal{K})}_q$ by Theorem 4.1.24, and accepted by $\mathbf{A}^\diamond_q$ by Lemma 4.1.22. So, by Lemma 4.1.18, $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ for some $\mathbf{F} \in \mathscr{L}(\mathbf{A}_q)$. Since $\mathbf{T}$ is individual unique, then so is $\mathbf{F}$, and by Lemma 4.1.13 $\mathbf{F}$ is a $(q, C_\mathcal{K})$-forest and $\mathcal{J}_\mathbf{F} \models q$. □

Now we can use $\mathbf{A}^1_\mathcal{K}$ and $\mathbf{A}^1_q$ to construct $\mathbf{A}_{\mathcal{K} \not\models q}$. As anticipated, we do this in three steps: (A) we project $\mathbf{A}^1_q$ to the alphabet $\Sigma_\mathcal{K}$, (B) we complement it, and (C) we intersect the resulting automaton with $\mathbf{A}^1_\mathcal{K}$.

A. Projection We define the projection of a language to a restricted alphabet, which contains the restriction of each tree in it.

Definition 4.1.27 (Σ'-restriction, Σ'-projection) *For $\Sigma = 2^\Phi$ and $\Sigma' = 2^{\Phi'}$ where $\Phi' \subseteq \Phi$, and for any Σ-labeled tree $\mathbf{T} = (T, L)$, the* Σ'-restriction *of $\mathbf{T}$ is the Σ'-labeled tree $\mathbf{T}' = (T, L')$, where $L'(w) = L(w) \cap \Sigma'$ for every $w \in T$. The* Σ'-projection *of a set $\mathcal{L}$ of Σ-labeled trees is the set of containing the Σ'-restrictions of all trees in $\mathcal{L}$.*

A 1NTA can be easily modified to accept only its projection to some restricted alphabet.

Lemma 4.1.28 *For Σ and Σ' as above, for every 1NTA $\mathbf{A}$ running over k-ary Σ-labeled trees, $k \geq 1$, it is possible to construct a 1NTA $\mathbf{A}^{\Sigma'}$ with $|Q(\mathbf{A}^{\Sigma'})| \leq |Q(\mathbf{A})|$ and $\mathsf{ind}(\mathbf{A}^{\Sigma'}) \leq \mathsf{ind}(\mathbf{A})$ that accepts the Σ'-projection of $\mathscr{L}(\mathbf{A})$.*

Proof (sketch). To obtain $\mathbf{A}^{\Sigma'}$ from $\mathbf{A}$, simply change Σ to Σ' and the transition function to δ' as follows. For each $\sigma \in \Sigma'$ and each $q \in Q(\mathbf{A})$, $\delta'(q, \sigma) = \bigvee_{\sigma' \in \Xi(\sigma)} \delta'(q, \sigma')$, where $\Xi(\sigma) = \{\sigma' \in \Sigma \mid \sigma' \cap \Phi' = \sigma\}$. □

From $\mathbf{A}^1_q$ we can obtain a 1NTA that accepts the $\Sigma_\mathcal{K}$-labeled trees representing an interpretation where there is a match for the query.

Lemma 4.1.29 *There is a 1NTA $\mathbf{A}^\uparrow_q$ such that, assuming unary encoding of numbers,*

1. *if $\mathcal{I}$ is a canonical interpretation such that $\mathcal{I} \models q$, then $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_\mathcal{I}) \in \mathscr{L}(\mathbf{A}^\uparrow_q)$,*
2. *if $\mathbf{T} \in \mathscr{L}(\mathbf{A}^\uparrow_q)$ and $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ for some $C_\mathcal{K}$-forest $\mathbf{F}$, then $\mathcal{I}_\mathbf{F} \models q$,*
3. *the number of states $|Q(\mathbf{A}^\uparrow_q)|$ is single exponential in $||\mathcal{K}, q||$,*
4. *the parity index $\mathsf{ind}(\mathbf{A}_q)$ is fixed, and*
5. *$\mathbf{A}^\uparrow_q$ can be built in time single exponential in $||\mathcal{K}, q||$.*

B. Complementation The following bounds for automata complementation are given in [MS95].

Proposition 4.1.30 *For every 1NTA $\mathbf{A}$ running over k-ary trees, $k \geq 1$, it is possible to construct a 1NTA $\overline{\mathbf{A}}$ that accepts a k-ary Σ-labeled tree (T, L) iff $(T, L) \notin \mathscr{L}(\mathbf{A})$, and such that $|Q(\overline{\mathbf{A}})| \leq 2^{O(f(\mathbf{A}))}$ and $\mathsf{ind}(\overline{\mathbf{A}}) = O(f(\mathbf{A}))$, where $f(\mathbf{A}) = \mathsf{ind}(\mathbf{A}) \cdot |Q(\mathbf{A})| \cdot \log |Q(\mathbf{A})|$.*

Hence we can complement $\mathbf{A}^\uparrow_q$ to obtain a 1NTA that accepts the $\Sigma_\mathcal{K}$-labeled trees representing an interpretation if there is no match for the query in it. Note that this may cause an exponential blow-up in the number of states.

Lemma 4.1.31 *There is a 1NTA $\mathbf{A}_{\neg q}$ such that, assuming unary encoding of numbers,*

1. *if* $\mathcal{I}$ *is a canonical interpretation and* $\mathcal{I} \not\models q$*, then* $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_\mathcal{I}) \in \mathscr{L}(\mathbf{A}_{\neg q})$*,*
2. *if* $\mathbf{T} \in \mathscr{L}(\mathbf{A}_{\neg q})$ *and* $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ *for some* $C_\mathcal{K}$*-forest* $\mathbf{F}$*, then* $\mathcal{I}_\mathbf{F} \not\models q$*,*
3. *the number of states* $|Q(\mathbf{A}_{\neg q})|$ *is double exponential in* $||\mathcal{K}, q||$*,*
4. *the parity index* $\mathsf{ind}(\mathbf{A}_q)$ *is single exponential in* $||\mathcal{K}, q||$*, and*
5. $\mathbf{A}_{\neg q}$ *can be built in time double exponential in* $||\mathcal{K}, q||$*.*

C. Intersection The following bounds for the intersection of two 1NTAs are known:[1]

Lemma 4.1.32 *Given 1NTAs* $\mathbf{A}_1$ *and* $\mathbf{A}_2$*, it is possible to construct a 1NTA* $\mathbf{A}$ *such that* $\mathscr{L}(\mathbf{A}) = \mathscr{L}(\mathbf{A}_1) \cap \mathscr{L}(\mathbf{A}_2)$ *with*

$$\begin{aligned} \mathsf{ind}(\mathbf{A}) &= O(f(\mathbf{A}_1, \mathbf{A}_2)) \\ |Q(\mathbf{A})| &\leq 2^{O(f(\mathbf{A}_1,\mathbf{A}_2)^2)} \cdot f(\mathbf{A}_1, \mathbf{A}_2) \cdot |Q(\mathbf{A}_1)| \cdot |Q(\mathbf{A}_2)| \end{aligned}$$

where $f(\mathbf{A}_1, \mathbf{A}_2) = \mathsf{ind}(\mathbf{A}_1) + \mathsf{ind}(\mathbf{A}_2) + 1$*.*

To obtain the automaton $\mathbf{A}_{\mathcal{K} \not\models q}$, we intersect $\mathbf{A}^1_\mathcal{K}$ with $\mathbf{A}_{\neg q}$.

Lemma 4.1.33 *There is a 1NTA* $\mathbf{A}_{\mathcal{K} \not\models q}$ *such that, assuming unary encoding of numbers,*

1. *if* $\mathcal{I}$ *is a canonical model of* $C_\mathcal{K}$ *and* $\mathcal{I} \not\models q$*, then* $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_\mathcal{I}) \in \mathscr{L}(\mathbf{A}_{\mathcal{K} \not\models q})$*,*
2. *if* $\mathbf{T} \in \mathscr{L}(\mathbf{A}_{\mathcal{K} \not\models q})$*, then* $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ *for some* $C_\mathcal{K}$*-forest* $\mathbf{F}$ *such that* $\mathcal{I}_\mathbf{F} \models_{\mathsf{ind}} C_\mathcal{K}$ *and* $\mathcal{I}_\mathbf{F} \not\models q$*,*
3. *the number of states* $|Q(\mathbf{A}_{\mathcal{K} \not\models q})|$ *is double exponential in* $||\mathcal{K}, q||$*,*
4. *the parity index* $\mathsf{ind}(\mathbf{A}_q)$ *is single exponential in* $||\mathcal{K}, q||$*, and*
5. $\mathbf{A}_{\mathcal{K} \not\models q}$ *can be built in time double exponential in* $||\mathcal{K}, q||$*.*

Proof. If $\mathcal{I}$ is a canonical model of $C_\mathcal{K}$, then $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_\mathcal{I}) \in \mathbf{A}^1_\mathcal{K}$ by item 1 in Lemma 4.1.25, and if $\mathcal{I} \not\models q$, then $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_\mathcal{I}) \in \mathscr{L}(\mathbf{A}_{\neg q})$ by item 1 in Lemma 4.1.31, so $\mathsf{tree}^{k(\mathcal{K})}(\mathbf{F}_\mathcal{I}) \in \mathscr{L}(\mathbf{A}_{\mathcal{K} \not\models q})$ and 1 holds. For 2, assume $\mathbf{T} \in \mathscr{L}(\mathbf{A}_{\mathcal{K} \not\models q})$. Then $\mathbf{T} \in \mathscr{L}(\mathbf{A}^1_\mathcal{K})$, and by 2 in Lemma 4.1.25, $\mathbf{T} = \mathsf{tree}(\mathbf{F})$ for some $C_\mathcal{K}$-forest $\mathbf{F}$ such that $\mathcal{I}_\mathbf{F} \models_{\mathsf{ind}} C_\mathcal{K}$. So by item 2 in Lemma 4.1.31, $\mathcal{I}_\mathbf{T} \not\models q$. Items 3 to 5 follow from Theorem 4.1.32 and the analogous items in Lemmas 4.1.25 and 4.1.31. □

Therefore, $\mathbf{A}_{\mathcal{K} \not\models q}$ accepts some input tree if and only if $C_\mathcal{K}$ has a canonical model in which q has no match. Combined with Propositions 3.1.5 and 3.2.4, we thus obtain:

Theorem 4.1.34 *For every knowledge base* $\mathcal{K}$ *and P2RPQ* q *in* $\mathcal{ZOIQ}$ *such that* $\mathcal{K}$ *enjoys the canonical model property (or in particular,* $\mathcal{K}$ *is in* $\mathcal{ZIQ}$*,* $\mathcal{ZOQ}$ *or* $\mathcal{ZOI}$*), it holds that* $\mathcal{K} \models q$ *iff* $\mathscr{L}(\mathbf{A}_{\mathcal{K} \not\models q}) = \emptyset$*.*

4.2 Complexity of Reasoning with Queries

The reduction of query entailment to automata emptiness as in Theorem 4.1.34 gives a tight upper complexity bound for the problem, and also allows us to obtain some tight complexity bounds for query containment, as we show in this section.

[1] To our knowledge, these bounds have not been published. A tighter bound of $\mathsf{ind}(\mathbf{A}) = \mathsf{ind}(\mathbf{A}_1) + \mathsf{ind}(\mathbf{A}_2)$ and $|Q(\mathbf{A})| = f'(\mathbf{A}_1, \mathbf{A}_2)! \cdot f'(\mathbf{A}_1, \mathbf{A}_2) \cdot |Q(\mathbf{A}_1)| \cdot |Q(\mathbf{A}_2)|$, where $f'(\mathbf{A}_1, \mathbf{A}_2) = (\mathsf{ind}(\mathbf{A}_1) + \mathsf{ind}(\mathbf{A}_2))/2 + 1$, was confirmed through personal communication with Yoad Lustig and Nir Piterman, to whom we are very grateful.

4.2.1 Deciding Query Entailment

Testing a 1NTA for emptiness is feasible within the following bounds.

Proposition 4.2.1 ([KV98]) *Given a 1NTA* $\mathbf{A}$*, the non-emptiness problem is decidable in time* $O(|Q(\mathbf{A})|^{\mathsf{ind}(\mathbf{A})})$*.*

We thus obtain one of the main results of this chapter.

Theorem 4.2.2 *Given a KB* $\mathcal{K}$ *in* $\mathcal{ZIQ}$*,* $\mathcal{ZOQ}$*, or* $\mathcal{ZOI}$ *and a* $\mathcal{ZOIQ}$ *P2RPQ* q*, deciding* $\mathcal{K} \models q$ *is in* 2ExpTime *in the total size of* q *and* $\mathcal{K}$*, assuming unary number encoding in number restrictions.*

Proof. Follows from Lemma 4.1.33 and Proposition 4.2.1. □

This bound is worst case optimal. It was shown in [Lut07] that already answering CQs over $\mathcal{ALCI}$ is 2ExpTime-hard, and we will show in Chapter 7 that the same lower bound holds for CQs over $\mathcal{SH}$.

The 2ExpTime-upper bound can not pushed much further. As we showed recently, P2RPQ entailment is undecidable for $\mathcal{ZOIQ}$ [ORŠ10].

4.2.2 Deciding Query Containment

By Proposition 2.2.14, our results allow us to immediately derive upper bounds for the complexity of query containment $\mathcal{K} \models q_1 \subseteq q_2$ in the three sublogics of $\mathcal{ZOIQ}$. If $\mathcal{K}$ and q_1 are in $\mathcal{ZOI}$ or $\mathcal{ZOQ}$, then we can use nominals to rewrite the full query q_1 into a concept C_{q_1} and reduce the containment test to the entailment of q_2. In $\mathcal{ZIQ}$ this is not possible in general, but if q_1 is a $\mathcal{ZIQ}$ CQ, then we can rewrite each atom as an ABox assertion and reduce containment to entailment. Hence we obtain:

Theorem 4.2.3 *Deciding* $\mathcal{K} \models q_1 \subseteq q_2$ *is in* 2ExpTime *w.r.t. the total size of* q_1*,* q_2*, and* $\mathcal{K}$ *if:*

- $\mathcal{K}$ *is a* $\mathcal{ZOQ}$ *KB,* q_1 *a* $\mathcal{ZOQ}$ *P2RPQ, and* q_1 *a* $\mathcal{ZOIQ}$ *P2RPQs, or*
- $\mathcal{K}$ *is a* $\mathcal{ZOI}$ *KB,* q_1 *a* $\mathcal{ZOI}$ *P2RPQ, and* q_1 *a* $\mathcal{ZOIQ}$ *P2RPQs, or*
- $\mathcal{K}$ *is a* $\mathcal{ZIQ}$ *KB,* q_1 *is a* $\mathcal{ZIQ}$ *CQ, and* q_2 *is a P2RPQ.*

These are, to our knowledge, the first bounds for containment of queries with regular expressions, and hence with some form of recursion, over expressive DL constraints. We belive they can provide an interesting contribution to the very active research in databases that aims at identifying families of recursive queries for which containment is decidable [CDGV05]. While it is known that the results can not be extended to $\mathcal{ZOIQ}$ [ORŠ10], a natural question that arises is whether we can decide the containment of P2RPQs in $\mathcal{ZIQ}$, or in some fragment of it that supports both inverses and counting. Our most recent research in this direction suggests that a positive answer may be possible.

4.3 Related Work and Discussion

Tree automata have already been employed in DLs and in the closely related program logics in order to obtain tight complexity bounds for satisfiability problems (cf. Section 3.5 and references therein). In this chapter, we have shown that they can also be used for the more involved task of query answering over a DL knowledge base. The core idea of our query answering technique

is inspired by [CDGLV00], which uses automata on infinite words to obtain a tight upper bound for the containment of two P2RPQs over simple semi-structured databases (graphs), with no constraints. Similar ideas have been exploited in other papers to obtain other decidability and complexity results for queries, even with some form of recursion, over semi-structured data, e.g., in [CDGV05, CDGLV03]. However, these results do not consider rich constraint languages, as we do here. As we have seen, taking expressive DL constraints into account may require the use of richer automata models.

The automata-based approach in this thesis, was first explored for (a fragment of) $\mathcal{ZIQ}$ in [CEO07], and then extended to all the logics considered here in [CEO09b]. To our knowledge, no other automata-based approaches to query answering in expressive DLs have been proposed so far. The power of automata allows us to significantly push the boundary of decidability in 2ExpTime. Indeed, some 2ExpTime upper bounds had been obtained already using different techniques, for example, for $\mathcal{ALCHIQ}$ [HMS04], for $\mathcal{ALC}_{reg}$ [CDGL98, CDGL08], for $\mathcal{SHIQ}$ [GHLS08], and for $\mathcal{SHOQ}$ [GHS08]. The automata algorithm we have described subsumes all these results, and extends them to a more expressive query language, and to some expressive DLs not addressed before.

In some of the previous techniques, transitive roles are problematic. For instance, of the mentioned algorithms, only the ones in [GHLS08] and [GHS08] fully support transitive roles (the ones in [HMS04] and [OCE08] allow them in the KB but not in the query). Automata, in contrast, accommodate transitivity quite naturally. In fact, the automata technique allows us to easily deal with a very rich set of constructs in both the DL and the query. It is unclear whether other approaches offer such flexibility.

We discuss next other relevant approaches to query answering in expressive DLs.

4.3.1 The Rolling-up Technique

The core idea of the rolling-up technique is to reduce CQ answering to concept satisfiability. The approach goes back to the work of Calvanese et al. [CDGL98, CDGL08] where the authors proposed the notion of *tuple graph* as a way to characterize a potential 'shape' that a match for the given query can take in a canonical interpretation. Very roughly, in order to show $\mathcal{K} \not\models q$, one considers all tuple graphs, and represents each of them as a DL concept C, possibly in an extension $\mathcal{L}'$ of the DL $\mathcal{L}$ considered. A model of the knowledge base where all the concepts C have an empty extension represents a model where the query has no match. Hence, query entailment reduces to checking satisfiability of a possibly exponentially larger knowledge base, and possibly in a more expressive DL.

This approach has been called 'rolling-up' since the work of [HT00], where it was applied to answer CQs in $\mathcal{SH}$, under certain restrictions, and it is the technique used for $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ in [GHLS08, GHS08]. Arriving at a correct characterization of the relevant concepts C is not always easy, and it can require complicated query rewriting steps to overcome subtle difficulties that change from DL to DL. The correct characterization of the queries in the presence of transitive roles was elusive for some time, until the authors of [GHLS08] obtained, using this technique, a reduction of CQ entailment in $\mathcal{K}$ to satisfiability in the DL $\mathcal{SHIQ}^{\sqcap}$, which extends $\mathcal{SHIQ}$ with role intersection. This yields a decision procedure for 2ExpTime, which was later extended to $\mathcal{SHOQ}$ in [GHS08].

4.3.2 Modified Tableau in the Style of Carin

Another method for deciding query entailment that also appeared in 1998 is the one underlying the Carin algorithms [LR98a]. Carin was proposed as a *hybrid language* combining Description Logics with rules, and its basic reasoning tasks are solved using an algorithm for what the authors

call the *existential entailment problem*, which for the purposes of this discussion can be thought of as CQ entailment (it is in fact a simple generalization of containment of CQs in UCQs). The proposed solution to the query entailment problem is a modified version of an an existing tableau algorithm. Intuitively, a tableau algorithm tries to construct a finite structure that represents a model of a KB, and uses *blocking conditions* to ensure that, if the construction does not fail, then it terminates in finite time and the resulting structure represents a model of the knowledge base. The key idea of Levy and Rousset was that, using the query size as a parameter, the blocking conditions can be modified in such a way the expansion terminates when the structure represents a *set of models that are indistinguishable by the query*, and they showed that a finite representation of a counter-model can be obtained using the modified blocking conditions if the query is not entailed.

The original CARIN algorithm was proposed for a DL called $\mathcal{ALCNR}$, which is closely related to $\mathcal{ALCHQ}$. The technique was first extended to CQs in $\mathcal{SHIQ}$ [OCE06], but disallowing transitive roles from the query. Then it was extended to the larger class of positive queries without transitive roles, and to the DLs $\mathcal{SHIQ}$, $\mathcal{SHOQ}$, and $\mathcal{SHOI}$ [OCE08].

A major drawback of the method is that, since the size of the query is used as a bound on the size of the structures that the query can distinguish, there seems to be no simple way to correctly handle transitive roles in the query. Also, it builds on tableaux algorithms that are not worst-case optimal, hence no optimal bounds (w.r.t. combined complexity) can be easily obtained from it.

Chapter 5

Reasoning in the $\mathcal{SR}$ family

The automata techniques devised in the previous chapters can be also fruitfully exploited for reasoning in the $\mathcal{SR}$ family. The DL $\mathcal{SROIQ}$ was defined in Section 2.1.4. It is similar to $\mathcal{ZOIQ}$, but lacks Boolean and regular role expressions. Instead a $\mathcal{SROIQ}$ RBox $\mathcal{R}$ comprises *complex role inclusions (CRIAs)* $P_1 \circ \cdots \circ P_n \sqsubseteq P$, and *role property axioms* that state properties of roles such as (ir)reflexivity, asymmetry, and disjointness. Its sublogics $\mathcal{SRIQ}$, $\mathcal{SROQ}$, $\mathcal{SROI}$ are analogous to $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, $\mathcal{ZOI}$. $\mathcal{SROIQ}$ was introduced in [HKS06] as an extension of some well-known expressive DLs and proposed as the basis for the new OWL 2 standard. The motivation of $\mathcal{SROIQ}$ was to combine as many of the useful features of the existing logics into OWL 2, and to additionally support some other expressive means that on the one hand are desirable for applications such as ontology engineering, and that on the other hand have limited impact on the techniques employed by existing reasoners and hence be relatively easily incorporated. Analyzing them from the perspective of worst-case computational complexity, these extensions are not always harmless when compared with the DLs underlying the first generation of OWL. Adding complex role inclusions to the DLs of the $\mathcal{SH}$ family increases exponentially the worst case complexity of standard reasoning, making $\mathcal{SRIQ}$ 2ExpTime-hard and $\mathcal{SROIQ}$ 2NExpTime-complete [Kaz08]. Adapted tableaux algorithms for these logics are available, and in fact implemented in actual reasoners, but their worst case behavior is actually non-deterministic triple exponential. From the results in [Kaz08] it follows that reasoning is feasible in 2NExpTime for $\mathcal{SRIQ}$, $\mathcal{SROQ}$ and $\mathcal{SROI}$, but only ExpTime harness is known in general. Despite its importance for the Semantic Web, query answering in the $\mathcal{SR}$ family had not been attempted so far. Tight complexity bounds and query answering techniques are known only for the lightweight profiles OWL EL and OWL QL of OWL 2, whose DL counterparts do not contain the basic $\mathcal{ALC}$.

In this chapter, we improve the existing complexity bounds for standard reasoning in the $\mathcal{SR}$ family and obtain the first decidability and complexity results for query answering and query containment. We achieve this via the algorithm for the $\mathcal{Z}$ family in the previous chapters. To exploit our automata-based techniques for reasoning in (sublogics of) $\mathcal{SROIQ}$, we transform each $\mathcal{SROIQ}$ KB $\mathcal{K}$ into a $\mathcal{ZOIQ}$ KB $\mathcal{K}'$. The rewritten $\mathcal{K}'$ is satisfiable iff the original $\mathcal{K}$ is and, with a slight query reformulation, we can evaluate queries posed to $\mathcal{K}$ over the models of $\mathcal{K}'$. The reduction builds on the fact that, since the role inclusion axioms of $\mathcal{SROIQ}$ are subject to some regularity restrictions, it is possible to simulate them through regular expressions. Role property axioms, on the other hand, can be simulated in $\mathcal{ZOIQ}$ using Self concepts and Boolean

roles. Since the rewriting causes an (unavoidable) exponential blow-up, we can show that KB satisfiability in $\mathcal{SRIQ}$, $\mathcal{SROI}$, and $\mathcal{SROQ}$ can be solved in 2ExpTime (even when the numbers in the number restrictions are encoded in binary). This is known to be optimal for $\mathcal{SRIQ}$. We also show that P2RPQ entailment is decidable in 3ExpTime, and so is containment $q_1 \subseteq q_2$ if the DL has nominals or q_1 has no regular expressions. These are, to our knowledge, the first such bounds. In fact, they are the first decidability results in any expressive DL of the $\mathcal{SR}$ family, and hence in significant sublogics of OWL 2.

5.1 Reducing $\mathcal{SROIQ}$ to $\mathcal{ZOIQ}$

In this section, we describe the rewriting that transforms a $\mathcal{SROIQ}$ KB $\mathcal{K}$ into a $\mathcal{ZOIQ}$ KB $\Psi(\mathcal{K})$, allowing us to exploit the automata-based algorithms described in the previous chapter for reasoning in $\mathcal{SROIQ}$.

The *regularity* condition from Definition 2.1.33 was first introduced in [HKS05], it is known to play a crucial role in the decidability of $\mathcal{SROIQ}$. It ensures that all implications between roles can be described by a regular language. By using suitable $\mathcal{ZOIQ}$ roles to represent these implications, we can remove the CRIAs from the KB while preserving satisfiability.

As usual, we use (possibly subindexed) P to represent atomic roles in $\overline{\mathsf{N_R}}$, and R to represent role chains $P_1 \circ \cdots \circ P_n$. Recall from Definition 2.1.33 that for a given RBox $\mathcal{R}$, the relation $\sqsubseteq^{\mathcal{R}}$ contains the following pairs of roles:

$$\text{If } R \sqsubseteq^{\mathcal{R}} P, \text{ then } R^{\mathcal{I}} \subseteq P^{\mathcal{I}} \text{ for each model } \mathcal{I} \text{ of } \mathcal{R}. \tag{5.1}$$

If $\mathcal{R}$ is regular, then the set of all R such that $R \sqsubseteq^{\mathcal{R}} P$, when seen as words over the alphabet $\overline{\mathsf{N_R}}$, defines a regular language, and can thus be represented by a $\mathcal{ZOIQ}$ role. The following lemma simply rephrases some results of [HS04]:

Lemma 5.1.1 *Given a regular $\mathcal{SROIQ}$ RBox $\mathcal{R}$, we can construct, for each $P \in \overline{\mathsf{N_R}}(\mathcal{R})$, a $\mathcal{ZOIQ}$ role $R = R^{\mathcal{R}}(P)$ such that*

1. *$P^{\mathcal{I}} \subseteq (R^{\mathcal{R}}(P))^{\mathcal{I}}$ for every interpretation $\mathcal{I}$,*
2. *$\mathcal{I} \models \mathcal{R}$ implies $(R^{\mathcal{R}}(P))^{\mathcal{I}} \subseteq P^{\mathcal{I}}$ for every interpretation $\mathcal{I}$,*
3. *$R^{\mathcal{R}}(P)$ is a simple $\mathcal{ZOIQ}$ role whenever P is simple w.r.t. $\mathcal{R}$,*
4. *if $\mathcal{R}$ is a $\mathcal{SROQ}$ RBox, then $R^{\mathcal{R}}(P)$ is a $\mathcal{ZOQ}$ role, and*
5. *$||R^{\mathcal{R}}(P)||$ is at most single exponential in $||\mathcal{R}||$.*

Proof. For each P in the set $\overline{\mathsf{N_R}}(\mathcal{R})$ of roles occurring in $\mathcal{R}$, let

$$L_{\mathcal{R}}(P) = \{P_1 \cdot \ldots \cdot P_n \mid P_1 \circ \cdots \circ P_n \sqsubseteq^{\mathcal{R}} P\}$$

It is well known that $L_{\mathcal{R}}(P)$ is a regular language over the alphabet $\overline{\mathsf{N_R}}(\mathcal{R})$ [HKS06, Kaz08]. Furthermore, there is an effective procedure to obtain a regular expression whose language is $L_{\mathcal{R}}(P)$; this was first shown for a slightly weaker DL than $\mathcal{SRIQ}$ (see Lemma 1 in [HS04]), and it holds for $\mathcal{SROIQ}$ (see Proposition 10 in [HKS06]). The regular expression can be written as a $\mathcal{ZOIQ}$ role (using $\circ$ in the place of concatenation), which we denote $R^{\mathcal{R}}(P)$, it is obtained iteratively using the regular order $\prec$ from Definition 2.1.33. Very roughly, one starts from simple regular expressions, and then goes up the $\prec$ relation replacing roles with unions of the regular expressions obtained in previous steps. It there are no inverse roles p^- in the CRIAs in $\mathcal{R}$, then $R^{\mathcal{R}}(P)$ is a regular role expression over the alphabet $\mathsf{N_R}(\mathcal{R})$; this implies 4. If a role P is simple w.r.t. $\mathcal{R}$, that is, there is no R with $|R| \geq 2$ such that $R \sqsubseteq^{\mathcal{R}} P$, then each R in $L_{\mathcal{R}}(P)$ has

length one, and is just an atomic role in $\overline{\mathsf{N_R}}(\mathcal{R})$. Then the following simple $\mathcal{ZOIQ}$ role satisfies all the requirements of the claim, and 3 follows:

$$R^{\mathcal{R}}(P) = \bigcup_{P' \sqsubseteq^{\mathcal{R}} P \in \mathcal{R}} P'$$

It is also shown in [HS04] that the size of $R^{\mathcal{R}}(P)$ and the time required to build it are single exponential in $||\mathcal{K}||$ in the worst case; this shows 5. Finally, 1, 2 hold because $P \in L_{\mathcal{R}}(P)$ for each P, and $R \sqsubseteq^{\mathcal{R}} P$ implies $R^{\mathcal{I}} \subseteq P^{\mathcal{I}}$ in each model of $\mathcal{R}$. □

5.1.1 The Rewriting Ψ

The rewriting Ψ of $\mathcal{SROIQ}$ KBs into $\mathcal{ZOIQ}$ KBs exploits this lemma. In what follows, we assume a fixed given regular RBox $\mathcal{R}$, and for each $P \in \overline{\mathsf{N_R}}$, we let $R^{\mathcal{R}}(P)$ denote an arbitrary but fixed $\mathcal{ZOIQ}$ role as in Lemma 5.1.1 above. Note that for roles not occurring in $\mathcal{R}$, we can simply take $R^{\mathcal{R}}(P) = P$.

Rewriting concepts and TBoxes

By replacing each role P by the regular expression $R^{\mathcal{R}}(P)$, we ensure that all the implications between roles imposed by $\mathcal{R}$ are satisfied in the models of the rewritten KB (which, in fact, does not support CRIAs).

Definition 5.1.2 *For any $\mathcal{SROIQ}$ concept C, we denote by $\Psi_{\mathcal{R}}(C)$ the $\mathcal{ZOIQ}$ concept that results from replacing each role P in C with $R^{\mathcal{R}}(P)$. For a $\mathcal{SROIQ}$ TBox $\mathcal{T}$, we define $\Psi_{\mathcal{R}}(\mathcal{T}) = \{\Psi_{\mathcal{R}}(C) \sqsubseteq \Psi_{\mathcal{R}}(D) \mid C \sqsubseteq D \in \mathcal{T}\}$.*

This rewriting preserves equivalence w.r.t. the models of the RBox $\mathcal{R}$.

Lemma 5.1.3 *Let $\mathcal{I}$ be an interpretation such that $\mathcal{I} \models \mathcal{R}$. Then*

1. *$C^{\mathcal{I}} = (\Psi_{\mathcal{R}}(C))^{\mathcal{I}}$ for each $\mathcal{SROIQ}$ concept C, and*
2. *$\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I} \models \Psi_{\mathcal{R}}(\mathcal{T})$ for each $\mathcal{SROIQ}$ TBox $\mathcal{T}$.*

Proof. Trivial, since $\mathcal{I} \models \mathcal{R}$ implies $P^{\mathcal{I}} = (R^{\mathcal{R}}(P))^{\mathcal{I}}$ for every role P. □

We note that the assumption $\mathcal{I} \models \mathcal{R}$ is necessary only to show that $(\exists R^{\mathcal{R}}(P).C)^{\mathcal{I}} \subseteq (\exists P.C)^{\mathcal{I}}$ and $(\forall R.C)^{\mathcal{I}} \subseteq (\forall R^{\mathcal{R}}(P).C)^{\mathcal{I}}$ for every C.

Rewriting ABoxes

To rewrite ABoxes, we replace each $\mathcal{SROIQ}$ concept C by the corresponding $\mathcal{ZOIQ}$ concept $\Psi_{\mathcal{R}}(C)$ in every assertion of the form $C(a)$. To remove the *negated role membership assertions* $\neg P(a, b)$, which are not allowed in $\mathcal{ZOIQ}$, we use a fresh role name for each assertion, together with a BRIA that ensures that the fresh symbol is interpreted as desired.

Definition 5.1.4 *Given an $\mathcal{SROIQ}$ ABox $\mathcal{A}$, let*

- *$\Psi_{\mathcal{R}}(\mathcal{A})$ be the $\mathcal{ZOIQ}$ ABox obtained by replacing in $\mathcal{A}$:*
  - *each assertion $C(a)$ with $\Psi(C)(a)$, and*
  - *each assertion $\neg S(a, b)$ with $P_{\neg S}(a, b)$ for a fresh role name $P_{\neg S}$.*

- $\Psi'_{\mathcal{R}}(\mathcal{A})$ *be the RBox containing* $P_{\neg S} \cap R^{\mathcal{R}}(S) \sqsubseteq \mathsf{B}$ *for each* $\neg S(a,b)$ *in* $\mathcal{A}$.

Lemma 5.1.5 *Let* $\mathcal{I}$ *be an interpretation such that* $\mathcal{I} \models \mathcal{R}$. *Then, for every* $\mathcal{SROIQ}$ *ABox* $\mathcal{A}$, $\mathcal{I} \models \mathcal{A}$ *iff* $\mathcal{I} \models \Psi_{\mathcal{R}}(\mathcal{A})$ *and* $\mathcal{I} \models \mathcal{T}_{\mathcal{R}}(\mathcal{A})$.

Rewriting RBoxes

Recall that the $\mathcal{SROIQ}$ RBox $\mathcal{R}$ contains a set of CRIAs and a set of role property axioms $\mathcal{R}^{\mathrm{a}}$. When rewriting a KB $\mathcal{K}$, we guarantee the satisfaction of the CRIAs by rewriting all concepts in $\mathcal{K}$ as described in the above definition. To ensure the satisfaction of the assertions, we use a set of BRIAs and a set of CIAs.

Definition 5.1.6 $\Psi(\mathcal{R})$ *and* $\Psi'(\mathcal{R})$ *are the following* $\mathcal{ZOIQ}$ *TBox and RBox, respectively:*

$$\begin{array}{rcll} \Psi(\mathcal{R}) & = & \{\top \sqsubseteq \exists P.\mathsf{Self} & | \quad \mathsf{Ref}(P) \in \mathcal{R}\} \cup \\ & & \{\exists R^{\mathcal{R}}(S).\mathsf{Self} \sqsubseteq \bot & | \quad \mathsf{Irr}(S) \in \mathcal{R}\}, \\ \Psi'(\mathcal{R}) & = & \{R^{\mathcal{R}}(S) \cap R^{\mathcal{R}}(S') \sqsubseteq \mathsf{B} & | \quad \mathsf{Dis}(S,S') \in \mathcal{R}\} \cup \\ & & \{R^{\mathcal{R}}(S) \cap \mathsf{Inv}(R^{\mathcal{R}}(S)) \sqsubseteq \mathsf{B} & | \quad \mathsf{Asy}(S) \in \mathcal{R}\} \end{array}$$

where, or any simple role S, $\mathsf{Inv}(S)$ *denotes the role obtained by replacing each atomic role* P *occurring in* S *by its inverse* $\mathsf{Inv}(P)$.

Lemma 5.1.7 *For every interpretation* $\mathcal{I}$, $\mathcal{I} \models \mathcal{R}$ *iff* $\mathcal{I} \models \Psi(\mathcal{R})$ *and* $\mathcal{I} \models \Psi'(\mathcal{R})$.

Rewriting KBs

Now we are ready to put all the pieces together and define the rewritten KB $\Psi(\mathcal{K})$.

Definition 5.1.8 (KB rewriting) *Let* $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *be a* $\mathcal{SROIQ}$ *KB. Then*

$$\Psi(\mathcal{K}) = \langle \Psi_{\mathcal{R}}(\mathcal{A}), \Psi_{\mathcal{R}}(\mathcal{T}) \cup \Psi(\mathcal{R}), \Psi'_{\mathcal{R}}(\mathcal{A}) \cup \Psi'(\mathcal{R}) \rangle$$

Clearly, the models of $\mathcal{K}$ are models of $\Psi(\mathcal{K})$. The converse holds only in a slightly weaker form, as the CRIAs of $\mathcal{K}$ need not be satisfied in every model $\mathcal{I}$ of $\Psi(\mathcal{K})$. However, each $\mathcal{I}$ can be transformed into a model of the CRIAs by adding all implied pairs of individuals to the extension of the role names. Hence this rewriting provides the desired reduction.

Proposition 5.1.9 *Let* $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *be a* $\mathcal{SROIQ}$ *KB, and let* $\Psi(\mathcal{K})$ *be the* $\mathcal{ZOIQ}$ *KB obtained from rewriting* $\mathcal{K}$ *as in Definition 5.1.8. Then the following hold:*

1. $\mathcal{K}$ *is in* $\mathcal{SRIQ}$, $\mathcal{SROI}$, *or* $\mathcal{SROQ}$, *then* $\Psi(\mathcal{K})$ *is in* $\mathcal{ZIQ}$, $\mathcal{ZOI}$, *or* $\mathcal{ZOQ}$, *respectively.*
2. *Every model of* $\mathcal{K}$ *is a model of* $\Psi(\mathcal{K})$.
3. *If* $\Psi(\mathcal{K})$ *has a model, then* $\mathcal{K}$ *has a model. In particular,* $\mathcal{I}' \models \mathcal{K}$ *for each* $\mathcal{I}$ *with* $\mathcal{I} \models \Psi(\mathcal{K})$, *where* $\mathcal{I}'$ *is the interpretation that has* $P^{\mathcal{I}'} = (R^{\mathcal{R}}(P))^{\mathcal{I}}$ *for each role* P, *and that is identical to* $\mathcal{I}$ *otherwise.*
4. $||\Psi\mathcal{K}||$ *and the time needed to construct it are polynomially bounded in* $||\mathcal{A}||+||\mathcal{T}||+||R^{\mathcal{R}}_{\max}||$, *where* $R^{\mathcal{R}}_{\max}$ *denotes the longest* $R^{\mathcal{R}}(P)$, $P \in \overline{\mathsf{N_R}}(\mathcal{K})$.

Proof. The first item holds because the transformation does not introduce any nominals or number restrictions, and uses only role names from $\mathsf{N_R}(\mathcal{R})$ if there are no inverse roles p^- in the CRIAs in $\mathcal{R}$. The second and third item follow easily from Lemmas 5.1.3, 5.1.5 and 5.1.7. For the last item, it is easy to see that all steps of the rewriting Ψ are polynomial in the size of $\mathcal{A}$, $\mathcal{T}$ and $R^{\mathcal{R}}_{\max}$ (but the latter can be exponential in the size of $\mathcal{R}$ [HS03]). □

5.2 Deciding KB satisfiability

Due to Proposition 5.1.9, the automata algorithm in Chapter 4 can be used to decide the satisfiability of $\mathcal{SRIQ}$, $\mathcal{SROQ}$ and $\mathcal{SROI}$ knowledge bases, and of any $\mathcal{SROIQ}$ knowledge base that enjoys suitable canonical models. Yet the fact that $R^{\mathcal{R}}_{\max}$ may be exponential in $||\mathcal{R}||$ can cause $||\Psi(\mathcal{K})||$ to be exponential in $||\mathcal{K}||$, and result in a double-exponential decision procedure. If the blow-up in $R^{\mathcal{R}}(P)$ can be avoided, then we call a KB *sparse*.

Definition 5.2.1 *We call a $\mathcal{SROIQ}$ KB* canonical *if $\Psi(\mathcal{K})$ has the canonical model property.*

We call a $\mathcal{SROIQ}$ KB sparse *if it is possible to construct w for each $P \in \overline{\mathsf{N_R}}(\mathcal{K})$ a $\mathcal{ZOIQ}$ role $R(P)$ of size* polynomial *in $||\mathcal{K}||$ that satisfies the conditions of Lemma 5.1.1.*

For example, *simple* sets of CRIAs as defined in [HS04] are sparse. A set of CRIAs $\mathcal{R}$ is simple if $P_1 \circ S \sqsubseteq^{\mathcal{R}} S$ and $S \circ P_2 \sqsubseteq^{\mathcal{R}} S$ for some $S, P_1, P_2 \in \overline{\mathsf{N_R}}$, implies that there is no P' such that $P' \sqsubseteq^{\mathcal{R}} P_1$, $P' \sqsubseteq^{\mathcal{R}} P_2$, and $P' \circ S' \sqsubseteq^{\mathcal{R}} P'$ or $S' \circ P' \sqsubseteq^{\mathcal{R}} P'$ for some $S' \in \overline{\mathsf{N_R}}$.

If $\mathcal{K}$ is sparse, then $||\Psi(\mathcal{K})||$ is polynomial in $||\mathcal{K}||$, and we obtain the same complexity bounds as for the $\mathcal{ZOIQ}$ family. Clearly, these bounds are tight. So, from Lemma 5.1.1, Proposition 5.1.9, and Theorem 3.4.2, we obtain:

Theorem 5.2.2 *Deciding the satisfiability of a given canonical $\mathcal{SROIQ}$ KB $\mathcal{K}$ (or, in particular, of a $\mathcal{SRIQ}$, $\mathcal{SROQ}$ or $\mathcal{SROI}$ KB $\mathcal{K}$) is in* 2ExpTime. *If $\mathcal{K}$ is sparse, then the problem is* ExpTime*-complete.*

Notably, the upper bounds hold independently of the encoding of numbers in the number restrictions. To our knowledge, this is the best upper bound currently known, both in the general and in the sparse case, and it improves over the 2NExpTime and NExpTime bounds established in [Kaz08]. The $\mathcal{SRIQ}$ and $\mathcal{SROIQ}$ tableaux algorithms in [HKS05, HKS06], in contrast require at least non-deterministic double exponential time even for sparse KBs.

For $\mathcal{SRIQ}$ the bound is optimal; a matching lower bound was shown in [Kaz08].

Corollary 5.2.3 *Satisfiability of $\mathcal{SRIQ}$ KBs is* 2ExpTime*-complete.*

The precise complexity of $\mathcal{SROQ}$ and $\mathcal{SROI}$ remains open.

5.3 Deciding Query Entailment and Containment

Making again use of the rewriting above, we can reduce also query entailment and query containment in $\mathcal{SROIQ}$ to $\mathcal{ZOIQ}$. To this end, we rewrite a P2RPQ q over $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ into a query $\Psi_{\mathcal{R}}(q)$ over $\Psi(\mathcal{K})$, in such a way that query entailment is preserved.

Lemma 5.3.1 *Let $\mathcal{K}$ be a $\mathcal{SROIQ}$ KB and q a P2RPQ over $\mathcal{K}$, and let $\Psi_{\mathcal{R}}(q)$ be obtained from q by replacing each occurrence of a role P by $R^{\mathcal{R}}(P)$. Then $\mathcal{K} \models q$ iff $\Psi(\mathcal{K}) \models \Psi_{\mathcal{R}}(q)$.*

Note that the rewriting of q into $\Psi_{\mathcal{R}}(q)$ may introduce regular expressions, even if they were not originally present in q, i.e., our technique reduces $\mathcal{L}$ queries to $\mathcal{L}$ P2RPQs.

Proposition 5.3.2 *Let q be a P2RPQ over a $\mathcal{SROIQ}$ knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. Then $\mathcal{K} \models q$ iff $\Psi(\mathcal{K}) \models \Psi_{\mathcal{R}}(q)$.*

Proof. Given an interpretation $\mathcal{I}$, let $\mathcal{I}'$ be as in item 3 of Proposition 5.1.9. First observe that every match π for $\Psi_{\mathcal{R}}(q)$ in some $\mathcal{I}$ with $\mathcal{I} \models \mathcal{R}$ is also a match for q; hence $\mathcal{I} \models \Psi_{\mathcal{R}}(q)$ and $\mathcal{I} \models \mathcal{R}$ implies $\mathcal{I} \models q$. Similarly, a match for q in $\mathcal{I}'$ is also a match for q in $\Psi_{\mathcal{R}}(q)$, so $\mathcal{I}' \models q$ implies $\mathcal{I} \models \Psi_{\mathcal{R}}(q)$.

Now assume $\mathcal{K} \models q$, and consider any model $\mathcal{I}$ of $\Psi(\mathcal{K})$. Then $\mathcal{I}' \models \mathcal{K}$ by item 3 of Proposition 5.1.9. Thus $\mathcal{I}' \models q$ and $\mathcal{I} \models \Psi_{\mathcal{R}}(q)$, so $\Psi(\mathcal{K}) \models \Psi_{\mathcal{R}}(q)$. Conversely, suppose $\Psi(\mathcal{K}) \models \Psi_{\mathcal{R}}(q)$. Consider an arbitrary $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$. By item 2 of Proposition 5.1.9, we know that $\mathcal{I} \models \Psi(\mathcal{K})$. Therefore $\mathcal{I} \models \Psi_{\mathcal{R}}(q)$, and since $\mathcal{I} \models \mathcal{R}$, it follows $\mathcal{I} \models q$. □

Again, the size of a longest regular expression $R^{\mathcal{R}}_{\max}$ may exponentially influence the overall complexity of the algorithm. Although in the results of Chapter 4 we had to assume that the numbers in number restrictions were encoding in unary to obtain optimal bounds, the blow-up that we get w.r.t. the RBox is completely independent from the encoding of numbers and makes the succinctness of the encoding irrelevant. So, putting together Lemma 5.1.1, Propositions 5.1.9 and 5.3.2, and Theorem 4.2.2, we obtain:

Theorem 5.3.3 *For a given canonical $\mathcal{SROIQ}$ KB $\mathcal{K}$ (or, in particular, a $\mathcal{SRIQ}$, $\mathcal{SROQ}$ or $\mathcal{SROI}$ KB $\mathcal{K}$) and a $\mathcal{SROIQ}$ P2RPQ q , query entailment $\mathcal{K} \models q$ is decidable in* 3ExpTime. *If $\mathcal{K}$ is sparse and numbers are encoded in unary, then the problem is* 2ExpTime*-complete.*

The results also extend to query containment. Using Propositions 5.3.2 and Theorem 4.2.3, we obtain:

Theorem 5.3.4 *$\mathcal{K} \models q_1 \subseteq q_2$ is in* 3ExpTime *w.r.t. the total size of q_1, q_2, and $\mathcal{K}$ if:*

- *$\mathcal{K}$ is a $\mathcal{SROQ}$ KB, q_1 a $\mathcal{SROQ}$ P2RPQ, and q_1 a $\mathcal{SROIQ}$ P2RPQs, or*
- *$\mathcal{K}$ is a $\mathcal{SROI}$ KB, q_1 a $\mathcal{SROI}$ P2RPQ, and q_1 a $\mathcal{SROIQ}$ P2RPQs, or*
- *$\mathcal{K}$ is a $\mathcal{SRIQ}$ KB, q_1 is a $\mathcal{SRIQ}$ CQ, and q_2 is a P2RPQ.*

If $\mathcal{K}$ is sparse and the numbers are encoded in unary, then $\mathcal{K} \models q_1 \subseteq q_2$ is 2ExpTime*-complete.*

These are, to our knowledge, the first results showing the feasibility of query answering in the $\mathcal{SR}$ family, and more specifically, in any extension of $\mathcal{ALC}$ that supports complex role inclusions.

Finally, we point out one restricted case where we get an exponential improvement in complexity even if the knowledge base is not sparse. Suppose we are given a canonical $\mathcal{SROIQ}$ KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and a positive query q in $\mathcal{SROIQ}$ such that every role P that occurs in q is simple w.r.t. $\mathcal{R}$. Then $||\Psi(\mathcal{K})||$ may be exponential in $||\mathcal{K}||$, but $||\Psi(q)||$ is only polynomial in $||q||$. A simple inspection of the bounds in Lemma 4.1.14 for the size of the automaton $\mathbf{A}_{\Psi(q)}$ (which verifies the existence of query matches) shows that the number of states in $\mathbf{A}_{\Psi(q)}$ depends (polynomially) only on $||\Psi(q)||$ and on the number of vocabulary symbols occurring in $\Psi(\mathcal{K})$, i.e., on $|\mathsf{N_{CI}}(\Psi(\mathcal{K}))|$, $|\overline{\mathsf{N_R}}(\Psi(\mathcal{K}))|$, and $|\mathsf{N_I}(\Psi(\mathcal{K}))|$, hence it is polynomial in $||\Psi(q)|| + ||\mathcal{K}||$. After the automata operations we obtain an automaton $\mathbf{A}_{\Psi(\mathcal{K}) \not\models \Psi(q)}$ such that the number of states of $\mathbf{A}_{\Psi(\mathcal{K}) \not\models \Psi(q)}$ and the time required to construct it are double exponential in $||\Psi(q)|| + ||\mathcal{K}||$, while its parity index is single exponential in $||\Psi(q)|| + ||\mathcal{K}||$. This means that we can test $\mathbf{A}_{\Psi(\mathcal{K}) \not\models \Psi(q)}$ for emptiness in time that is double exponential in $||\Psi(q)|| + ||\mathcal{K}||$. Since $||\Psi(q)||$ is polynomial in $||q||$, this is also double exponential in $||q|| + ||\mathcal{K}||$, and we obtain:

Corollary 5.3.5 *Given a canonical $\mathcal{SROIQ}$ KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and a positive query q in $\mathcal{SROIQ}$ such that every role P that occurs in q is simple w.r.t. $\mathcal{R}$, the problem of deciding $\mathcal{K} \models q$ is feasible in double exponential time in $||\mathcal{K}, q||$.*

Since answering conjunctive queries is already 2ExpTime-hard for $\mathcal{ALCI}$, this is worst case optimal for $\mathcal{SRIQ}$ and $\mathcal{SROI}$. This is interesting, as entailment of queries with only simple roles is provably harder than standard reasoning for $\mathcal{SHIQ}$, but not for $\mathcal{SRIQ}$.

5.4 Related Work and Discussion

In this chapter,we have improved the existing upper bounds for KB satisfiability in all fragments of $\mathcal{SROIQ}$ that enjoy the canonical model property. This includes, in particular, the three maximal and incomparable sublogics $\mathcal{SRIQ}$, $\mathcal{SROI}$ and $\mathcal{SROQ}$. We have also shown that query entailment and containment are decidable in these logics for the very expressive class of P2RPQs (in the case of $\mathcal{SRIQ}$, containment was shown for CQs in P2RPQs only), and provided tight complexity bounds under some restrictions.

We hope that our results will contribute towards a better understanding of the computational complexity of the logics in the $\mathcal{SR}$ family, which has only been limitedly explored. For standard reasoning, a non-deterministic triple exponential bound for $\mathcal{SROIQ}$ and its sublogics can be easily obtained from the tableaux algorithm in the original $\mathcal{SROIQ}$ paper [HKS06], but for a long time the best known lower bounds were the ones inherited from the simpler $\mathcal{SH}$ family: $\mathcal{SROIQ}$ is NExpTime-hard, and its fragments $\mathcal{SRIQ}$, $\mathcal{SROQ}$ and $\mathcal{SROI}$ are ExpTime-hard. This big gap was significantly narrowed and partially closed by Kazakov, who showed that $\mathcal{SRIQ}$ is 2ExpTime-hard and $\mathcal{SROIQ}$ is 2NExpTime-complete [Kaz08]. A gap between 2ExpTime and 2NExpTime remained for $\mathcal{SRIQ}$, and a gap between ExpTime and 2NExpTime for $\mathcal{SROI}$ and $\mathcal{SROQ}$. We have closed the former, and narrowed the latter to 2ExpTime.

For query entailment and containment we have obtained the first decidability results, but left an open gap between 2ExpTime and 3ExpTime, for all three logics. Another problem that remains open is the feasibility of reasoning with queries in $\mathcal{SROIQ}$. The only decidability result known until now for an expressive DL that allows simultaneously for nominals, inverses and number restrictions, is the result for CQ entailment in $\mathcal{ALCHOIQ}$ in [GR09], which however does not give any elementary bound on the complexity of the problem. We note that it is not clear whether the undecidability of P2RPQs in $\mathcal{ZOIQ}$ extends to $\mathcal{SROIQ}$ as well. It was shown in [PH09] that conjunctive query entailment is decidable in the *guarded* two-variable fragment of first-order logic with counting quantifiers –which captures a large fragment of $\mathcal{SRIQ}$ but can not express nominals– but undecidable for the full two-variable fragment with counting quantifiers $\mathcal{C}_2$.

The translation of $\mathcal{SROIQ}$ into $\mathcal{ZOIQ}$ that we have exploited here is closely related to some known techniques. In particular, the the 2NExpTime upper bound in [Kaz08] is obtained by rewriting (also with an exponential blow-up) $\mathcal{SROIQ}$ into $\mathcal{C}_2$. That rewriting builds on a technique developed for a family of modal logics known as *grammar logics* [DdN05], and relies on essentially the same principles as our one. Indeed, it is not hard to see that if we consider KB in a suitable normal form that only allows universal concepts in GCIs of the form $C \sqsubseteq \forall P.A$ with A a concept name (such normal forms are well known; see, e.g. [KM08]), then to rewrite the TBox, we only need to replace P with $R^{\mathcal{R}}(P)$ in universal concepts $\forall P.A$. This reduction would closely resemble the one in [Kaz08]. Roughly, instead of using complex roles inside universals of the form $\forall R^{\mathcal{R}}(P).A$, he uses the finite automata representation of $R^{\mathcal{R}}(P)$, additional concept names A_s for each state s in the automaton, and concept inclusion axioms of the form $A_s \sqsubseteq A'_s$ to 'break down' the transitions of the automaton step by step. These kind of encodings, in turn, can be seen as generalization of the well known 'elimination of transitivity' used, e.g., in [Tob01, Mot06].

Finally, we remark that the reduction to $\mathcal{ZOIQ}$ can also be applied to KBs in the DL $\mathcal{SROIQb}$ that extends $\mathcal{SROIQ}$ with safe Boolean roles [RKH08a], hence Theorems 5.2.2, 5.3.3,

and 5.3.4 also apply to the corresponding extensions of the $\mathcal{SR}$ logics.

Chapter 6

Querying DLs with Inverse Roles

The automata-based technique we described in the first part of this thesis is very powerful: it allowed us to obtain very general algorithms, and a 2ExpTime upper bound for the complexity of query answering that is tight for a wide range of description logics. However, for some DLs, there are no matching lower bounds, and the automata technique does not give us much insight into the effect on the overall complexity of the various constructors. The automata approach is also not so adequate if we want to consider more fine-grained complexity measures such as *data complexity*, i.e., the complexity of query entailment when the query, the TBox, and the RBox are fixed, and only the ABox (i.e., the data) is given as an input (see Section 2.3.2). While the data complexity of query answering is known to be coNP-complete for many extensions of $\mathcal{ALC}$ [OCE08] and even for more expressive formalisms such as the two-variable fragment of First-Order Logic [PH09], our automata construction requires double exponential time and space even when all components except the ABox are fixed. It is not clear whether a better upper bound in data complexity—or in some other restricted setting of lower complexity—can be easily achieved with this kind of automata reduction. Instead we consider a different technique that seems better suited in this respect, and propose query answering algorithms based on *knots*.

Knots, as we discuss them here, were introduced in [ŠE07, EŠ10] for reasoning over logic programs with function symbols. They are an instance of the *mosaic* technique known from Modal Logics [Ném86, BdRV01]. Essentially, a knot is a small schematic labeled tree of depth ≤ 1, representing a *pattern* for a small subtree that can occur in a tree-shaped model of a knowledge base. Knots can be used as 'blocks' for building models, and models can be represented as sets of knots. Since only finitely many knots exist (over the signature of the KB), the representation of a possibly infinite model is always finite. To ensure that sets of knots correctly represent a model, it suffices to impose some simple conditions of two kinds: *local* conditions that apply to individual knots and deal with the internal consistency of the nodes in a knot, and *global* conditions ensure that instances of the knots in a set can be assembled together into models. Satisfiability testing of a KB can then be reduced to finding a knot set that satisfies the local and global conditions, and can usually be done using a simple elimination method. The method can even be extended to CQ answering: we mark each knot with a *set of (sub)queries* that cannot be mapped locally into the model part the knot describes, and global conditions on sets of marked knots ensure that a full countermodel for the query can be constructed from them.

In this chapter, we illustrate the approach on the DLs $\mathcal{ALCH}$ and $\mathcal{ALCHI}$, and obtain in a transparent way a worst-case optimal algorithm for answering unions of conjunctive queries

if	$C(a) \in \mathcal{A}$	then	$C \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$C_1 \sqsubseteq C_2 \in \mathcal{T}$	then	$\sim C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$C \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$\sim C \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C_1, C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$C_1 \sqcap C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C_1, C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$\exists P.C \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$\forall P.C \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C \in Cl_{\mathsf{C}}(\mathcal{K})$

Table 6.1: Concept closure $Cl_{\mathsf{C}}(\mathcal{K})$ of an $\mathcal{ALCHI}$ KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$

in both logics. More specifically, we present an algorithm to decide UCQ entailment over the basic DL $\mathcal{ALCH}$, and its extension with inverse roles $\mathcal{ALCHI}$. The algorithm yields a tight ExpTime upper bound in the $\mathcal{ALCH}$ case, and a tight 2-ExpTime upper bound for $\mathcal{ALCHI}$. It hence allows us to identify $\mathcal{ALCH}$ as an expressive description logic for which query answering is feasible in single exponential time, and thus not more expensive than satisfiability testing. It also gives a tight coNP upper bound for data complexity. We will see that the marking of the knots is simple and intuitive. It is flexible enough to easily extend to other DLs with different constructs, and allows for elegant refinements that yield optimal bounds even in the presence of very subtle sources of complexity, as we discuss later.

The chapter is organized as follows. First, in Section 6.1, we develop the notion of *canonical models* that we exploit in the query entailment algorithm. To describe the algorithm, we first reduce in Section 6.2 the general query answering problem to many instances of a simpler problem: namely, to decide query entailment over *simple* knowledge bases whose ABox consists of one concept membership assertion only. Then we introduce knots in Section 6.3 and show how they can be used to decide knowledge base satisfiability of simple knowledge bases, and describe the knot based technique for query answering in Section 6.4. We analyze its complexity in Section 6.5. Finally, related work is addressed in Section 6.6.

6.1 Canonical Models for $\mathcal{ALCHI}$

The algorithm for query entailment that we describe in this chapter exploits the fact that, to solve any given instance of the problem, we only need to consider some forest-shaped *canonical models* of the knowledge base. This is just a special instance of the canonical model property that we showed for the logics of the $\mathcal{Z}$ family in Chapter 3. Before defining canonical models for $\mathcal{ALCH}$ and $\mathcal{ALCHI}$, we adapt to these logics the notions of concept closure and concept and role types.

6.1.1 Syntactic Closure and Types

We start by defining the *concept closure* of a knowledge base $\mathcal{K}$, which contains all the concepts that are relevant for deciding its satisfiability. The definition of the concept closure is standard: it contains (the negation normal form of) all concepts occurring in the ABox and in the TBox of $\mathcal{K}$, and it is closed under subconcepts and their negations. It can be compared to Definition 3.2.1, but now we define it for a KB rather than for a concept, and the definition is simpler because we have less constructors.

Definition 6.1.1 (Concept closure) *The* concept closure $Cl_{\mathsf{C}}(\mathcal{K})$ *of an* $\mathcal{ALCHI}$ *KB* $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *is the smallest set of* $\mathcal{ALCHI}$ *concepts closed under the rules in Table 6.1.*

if	$C_1 \sqsubseteq C_2 \in \mathcal{T}$,	then	$\sim C_1 \sqcup C_2 \in \mathbf{t}$,		
if	$C \in Cl_{\mathsf{C}}(\mathcal{K})$,	then	$C \in \mathbf{t}$	iff	$\sim C \notin \mathbf{t}$,
if	$C_1 \sqcap C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$,	then	$C_1 \sqcap C_2 \in \mathbf{t}$	iff	$\{C_1, C_2\} \subseteq \mathbf{t}$,
if	$C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$,	then	$C_1 \sqcup C_2 \in \mathbf{t}$	iff	$\{C_1, C_2\} \cap \mathbf{t} \neq \emptyset$.

Table 6.2: Concept type $\mathbf{t} \subseteq Cl_{\mathsf{C}}(\mathcal{K})$

We also define the notions of *concept type* and *role type* for $\mathcal{ALCHI}$ KBs. A concept type is a maximal consistent set of concepts from $Cl_{\mathsf{C}}(\mathcal{K})$ that is compatible with the axioms in the TBox. Role types for $\mathcal{ALCHI}$ are just sets of roles closed under the role inclusions in the RBox.

Definition 6.1.2 (Concept and role types) *A* concept type *of an $\mathcal{ALCHI}$ KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ is a set $\mathbf{t} \subseteq Cl_{\mathsf{C}}(\mathcal{K})$ of concepts closed under the rules of Table 6.2.*

A role-type *of $\mathcal{K}$ is a set $\mathbf{r} \subseteq \overline{\mathsf{N_R}}(\mathcal{K})$ (that is, a subset of the set of role names occurring in $\mathcal{K}$ and their inverses) such that, for each $P_1 \sqsubseteq P_2 \in \mathcal{T}$, $P_1 \in \mathbf{r}$ implies $P_2 \in \mathbf{r}$ and $\mathsf{Inv}(P_1) \in \mathbf{r}$ implies $\mathsf{Inv}(P_2) \in \mathbf{r}$. For a role type $\mathbf{r}$, we define $\mathsf{Inv}(\mathbf{r}) = \{\mathsf{Inv}(P) \mid P \in \mathbf{r}\}$, that is, $\mathsf{Inv}(\mathbf{r})$ contains the inverse of each role in $\mathbf{r}$.*

The set of all concept types of $\mathcal{K}$ is denoted by $\mathsf{typesC}(\mathcal{K})$, and the set of all role types by $\mathsf{typesR}(\mathcal{K})$.

Now we can define canonical models, which are models with a forest-shaped domain. The roots are the interpretation of the individuals $\mathsf{N_I}(\mathcal{K})$ occurring in $\mathcal{K}$ (which in the case of $\mathcal{ALCHI}$ KBs are precisely the individuals in the ABox assertions) and they may be arbitrarily interrelated. Every other node of the forest can only be related via a role name to its successors and to its predecessor, and in the case of $\mathcal{ALCH}$, to its successors only. The definition of canonical models can be compared with the one for $\mathcal{ZOIQ}$ (Definition 3.2.2), but now we disallow relations between a non-root node and a root node, and between a non-root node and itself. We also relax the restriction of fixed arity to bounded branching. We use $|Cl_{\mathsf{C}}(\mathcal{K})|$ as a bound; this is sufficient for logics that do not support number restrictions.

Definition 6.1.3 ((one-way) canonical interpretation, canonical model) *An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is* canonical (for $\mathcal{K}$) *if:*

1. *$\Delta^{\mathcal{I}}$ is a forest with branching degree bounded by $|Cl_{\mathsf{C}}(\mathcal{K})|$,*
2. *$\mathsf{roots}(\Delta^{\mathcal{I}}) = \{a^{\mathcal{I}} \mid a \in \mathsf{N_I}(\mathcal{K})\}$,*
3. *$\mathcal{I}$ is connected, that is, for each pair $\{w, w'\} \subseteq \Delta^{\mathcal{I}}$ with $w' \in \mathsf{succ}(w)$ there is some $P \in \overline{\mathsf{N_R}}(\mathcal{K})$ such that $(w, w') \in P^{\mathcal{I}}$, and*
4. *for every pair $\{w, w'\} \in \Delta^{\mathcal{I}}$ such that $\{w, w'\} \not\subseteq \mathsf{roots}(\Delta^{\mathcal{I}})$ and $(w, w') \in p^{\mathcal{I}}$ for some role name p, either*
   a) $w' \in \mathsf{succ}(w)$, or
   b) $w \in \mathsf{succ}(w')$.

If for every pair of non-root nodes 4a holds (i.e., $(w, w') \in p^{\mathcal{I}}$ implies $w' \in \mathsf{succ}(w)$), then we call $\mathcal{I}$ a one-way *canonical interpretation.*

We call $\mathcal{I}$ a canonical model of $\mathcal{K}$ *if $\mathcal{I} \models \mathcal{K}$.*

The following well-known proposition can be derived by simply inspecting the proof of Proposition 3.2.12 or Proposition 3.2.13; it also follows, e.g., from the results in [GHLS08, OCE08].

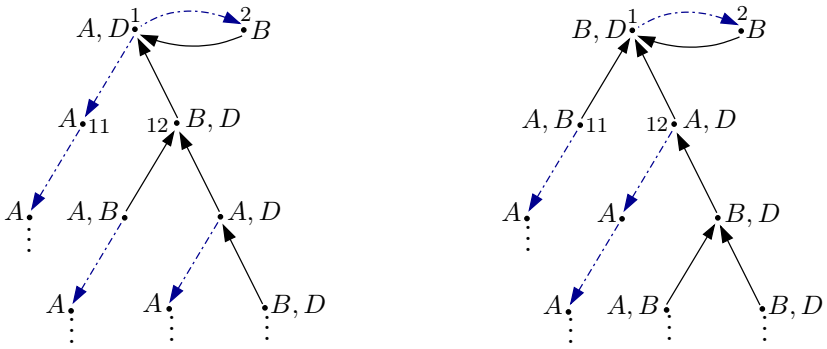

Figure 6.1: Two canonical models $\mathcal{I}_1$ and $\mathcal{I}_2$ for $\mathcal{K}_1$

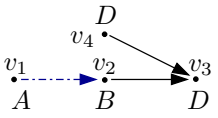

Figure 6.2: The query graph of q_1

Proposition 6.1.4 *Let $\mathcal{K}$ be a KB in $\mathcal{ALCHI}$. For every (positive) query q, if $\mathcal{K} \not\models q$ then there is a canonical model $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I} \not\models q$. Moreover, if $\mathcal{K}$ is in $\mathcal{ALCH}$, then $\mathcal{I}$ is one-way.*

Observe that, in particular, this implies that if $\mathcal{K}$ has only one individual, then we can always find a *tree-shaped* countermodel for any query that it does not entail. This will play an important role in the query answering algorithm we describe next.

Example 6.1.5 *In this chapter we use as a running example the knowledge base $\mathcal{K}_1 = \langle \mathcal{A}_1, \mathcal{T}_1, \emptyset \rangle$, where*

$$\begin{aligned} \mathcal{A}_1 = &\ \{D(a), B(b), p_1(a,b), p_2(b,a)\} \\ \mathcal{T}_1 = &\ \{D \sqsubseteq A \sqcup B, A \sqsubseteq \exists p_1.A, D \sqsubseteq \exists {p_2}^-.D, B \sqcap D \sqsubseteq \exists {p_2}^-.(A \sqcap B)\} \end{aligned}$$

Two (infinite) canonical models $\mathcal{I}_1$ and $\mathcal{I}_2$ for $\mathcal{K}_1$ are depicted in Figure 6.1 (only the roots and the first level nodes are explicitely named). Each node w is labeled with the concept names $\{A \in \mathsf{N_C}(\mathcal{K}) \mid w^{\mathcal{I}_1} \in A^{\mathcal{I}_1}\}$. The dashed blue arrows represent the role p_1 and the solid black arrows the role p_2. We have $a^{\mathcal{I}_1} = a^{\mathcal{I}_2} = 1$ and $b^{\mathcal{I}_1} = b^{\mathcal{I}_2} = 2$.

We consider the Boolean conjunctive query

$$q_1 = \ \{A(v_1), p_1(v_1, v_2), B(v_2), p_2(v_2, v_4), D(v_3), p_2(v_3, v_4), D(v_4)\}$$

The query graph of q_1 is depicted in Figure 6.2. A match for q_1 in $\mathcal{I}_1$ that has $\pi(v_1) = \pi(v_4) = 1$, $\pi(v_2) = 2$, and $\pi(v_3) = 12$ is depicted in Figure 6.3. In the depicted part of $\mathcal{I}_2$, however, there is no match for q_1 (note that there are no p_1 arcs from a node labeled A to a node labeled B), and in fact, $\mathcal{K}_1 \not\models q_1$.

6.2 From General to Simple KBs

In what follows, we will use q to denote CQs, an Q to denote UCQs. For the remainder of this section, we assume a fixed given KB $\mathcal{K}$ and a Boolean UCQ $Q = \exists \vec{v}.\varphi(\vec{x}, \vec{v})$ in $\mathcal{ALCHI}$, and

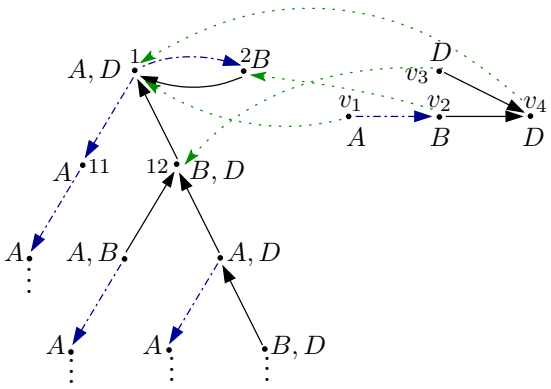

Figure 6.3: A match for q_1 in $\mathcal{I}_1$

describe an algorithm for deciding $\mathcal{K} \models Q$. As discussed in Section 2.2.2, the algorithm can also be used for query query answering (and yields the same complexity bounds). Recall that φ is a disjunction of conjunctions of atoms, and that we may assume without loss of generality that all atoms are of the form $A(v)$ or $P(v, v')$ with $\{v, v'\} \subseteq \{v_1, \ldots, v_n\} \subseteq \mathsf{N_V}$, $\vec{v} = v_1, \ldots, v_n$, $A \in \mathsf{N_C}$ and $P \in \overline{\mathsf{N_R}}$. That is, only concept names, atomic roles, and variables occur in Q. Note that atoms $p^-(v, v')$ can be equivalently rewritten as $p(v', v)$, hence every UCQ Q in $\mathcal{ALCHI}$ is equivalent to one in $\mathcal{ALCH}$ (in which only role names occur). Recall that we may identify a Boolean CQ q with the set of its atoms $\mathsf{Atoms}(q)$ and, similarly, we may treat a UCQ Q as a union of CQs, that is, a collection of sets of atoms. We call a UCQ *connected* if each $q \in Q$ is connected.

Our algorithm to decide $\mathcal{K} \models Q$ works in two steps: first we reduce deciding $\mathcal{K} \models Q$ to a set of calls to a procedure for deciding $\mathcal{K}' \models Q'$, where Q' is a connected UCQ and $\mathcal{K}'$ is a *simple* KB allowing only one individual and one single (concept membership) assertion in the ABox. In the second step, we employ a technique based on knots to decide $\mathcal{K}' \models Q'$. As we will see, knots are specially adequate for handling tree-shaped structures and thus well suited in the second step, as simple knowledge bases enjoy canonical models that are tree-shaped.

The first notion in the reduction is that of an *ABox completion*. In an ABox completion we assign concept types to individuals and role types to pairs of individuals, in such a way that the ABox and all the constraints imposed by the TBox and RBox are satisfied, excepting the existential restrictions.

Definition 6.2.1 (ABox completion) *An* ABox completion *for $\mathcal{K}$ is a function θ that maps each $a \in \mathsf{N_I}(\mathcal{K})$ to a concept type $\theta(a) \in \mathsf{typesC}(\mathcal{K})$ and each pair $(a, b) \in \mathsf{N_I}(\mathcal{K}) \times \mathsf{N_I}(\mathcal{K})$ to a role type $\theta(a, b) \in \mathsf{typesR}(\mathcal{K})$ such that*

1. *$\theta(a, b) = \mathsf{Inv}(\theta(b, a))$ for each $a, b \in \mathsf{N_I}(\mathcal{K})$;*
2. *for each $C(a) \in \mathcal{A}$, $C \in \theta(a)$,*
3. *for each $P(a, b) \in \mathcal{A}$, $P \in \theta(a, b)$, and*
4. *for each $a, b \in \mathsf{N_I}(\mathcal{K})$, $\forall P.C \in \theta(a)$ and $P \in \theta(a, b)$ imply $C \in \theta(b)$.*

Note that by taking a completion and looking only at one specific individual, we can obtain a *simple* KB with one individual only. We will refer to these KBs later.

Definition 6.2.2 ((Induced) simple knowledge base) *We call a knowledge base* simple *if its ABox is of the form $\{A(a)\}$ for some concept name A. For a completion θ and an individual a, the* knowledge base for a induced by θ *is the simple KB*

$$\mathcal{K}_{\theta(a)} = \langle \{A_0(a)\}, \mathcal{T} \cup \{A_0 \sqsubseteq C_{\theta(a)}\}, \mathcal{R} \rangle,$$

where A_0 is a distinguished concept name and $C_{\theta(a)} = \bigsqcap_{C \in \theta(a)} C$.

Example 6.2.3 *In the examples, we implicitly assume that concept types are closed under the rules of Table 6.2, and only mention the concept names and existential restrictions that occur positively in them (that is, negated concepts, conjunctions and disjunctions are omitted). Consider the following concept types and role types:*

$$\begin{array}{rcl} \mathbf{t}_B & = & \{B\} \\ \mathbf{t}_{A,D} & = & \{A, D, \exists p_1.A, \exists p_2{}^-.D\} \\ \mathbf{t}_{B,D} & = & \{B, D, \exists p_2{}^-.D, \exists p_2{}^-.(A \sqcap B)\} \end{array} \qquad \begin{array}{rcl} \mathbf{r}_{p_1} & = & \{p_1\} \\ \mathbf{r}_{p_2{}^-} & = & \{p_2{}^-\} \end{array}$$

Then the following θ_1 and θ_2 are completions for $\mathcal{K}_1$:

$$\begin{array}{rcl} \theta_1(a) & = & \mathbf{t}_{A,D} \\ \theta_1(b) & = & \mathbf{t}_B \\ \theta_1(a,b) & = & \mathbf{r}_{p_1} \\ \theta_1(b,a) & = & \mathbf{r}_{p_2{}^-} \end{array} \qquad \begin{array}{rcl} \theta_2(a) & = & \mathbf{t}_{B,D} \\ \theta_2(b) & = & \mathbf{t}_B \\ \theta_2(a,b) & = & \mathbf{r}_{p_1} \\ \theta_2(b,a) & = & \mathbf{r}_{p_2{}^-} \end{array}$$

Note that the completion θ_1 coincides with $\mathcal{I}_1$ (i.e., it describes the part of $\mathcal{I}_1$ that corresponds to the individuals) and θ_2 coincides with $\mathcal{I}_2$. The knowledge base for a induced by θ_2 is

$$\mathcal{K}_{\theta_2(a)} = \langle \{A_0(a)\}, \mathcal{T} \cup \{A_0 \sqsubseteq C_{\mathbf{t}_{B,D}}\}, \mathcal{R} \rangle,$$

where $C_{\mathbf{t}_{B,D}} = \bigsqcap_{C \in \mathbf{t}_{B,D}} C = \{B \sqcap D \sqcap \exists p_2{}^-.D \sqcap \exists p_2{}^-.(A \sqcap B) \sqcap \cdots\}$. In the rest of the examples we focus on the completion θ_2, and we denote $\mathcal{K}_{\theta_2(a)}$ by $\mathcal{K}_a$ and $\mathcal{K}_{\theta_2(b)}$ by $\mathcal{K}_b$.

Intuitively, every canonical model can be seen as an ABox completion, possibly with a collection of of trees attached to (the interpretation of) some individuals. Hence, to ensure that there is a canonical model where the query is falsified, i.e., a countermodel, it is sufficient to find an ABox completion and a set of trees that we can attach to it, such that no query match is possible. To achieve the latter we use a standard technique [BEL$^+$10, GHLS08].

A *match candidate* for an ABox completion and a CQ $q \in Q$, is a way to partially match q into the completion, and partition the rest into a set of subqueries that are assigned to the individuals. Intuitively, the partial match in the completion can become a full match for q only when we attach to each a a tree that has a match for the corresponding subqueries. Formally, a match candidate maps each variable of q to an individual a, or to a symbol $a_\downarrow$ that intuitively means 'inside the tree rooted at (the interpretation of) a'.

Definition 6.2.4 (Match candidate) *A* match candidate for a completion θ and a query $q \in Q$ *is a mapping $\xi : \mathsf{Vl}(q) \to \{a, a_\downarrow \mid a \in \mathsf{N_I}(\mathcal{K})\}$ such that*

- *if $\xi(v) = a$, and either $P(v, v') \in q$ or $P(v', v) \in q$, then either $\xi(v') = a_\downarrow$ or $\xi(v') \in \mathsf{N_I}(\mathcal{K})$,*
- *if $\xi(v) = a_\downarrow$, and either $P(v, v') \in q$ or $P(v', v) \in q$, then either $\xi(v') = a$ or $\xi(v') = a_\downarrow$, and*
- *for each $v, v' \in \mathsf{Vl}(q)$ such that $\{\xi(v), \xi(v')\} \in \mathsf{N_I}(\mathcal{K})$, $\{P \mid P(v, v') \in q\} \subseteq \theta(\xi(v), \xi(v'))$.*

For a match candidate ξ and $a \in \mathsf{N_I}(\mathcal{K})$, let $V_\xi(a) = \{v \in \mathsf{VI}(q) \mid \xi(v) \in \{a, a_\downarrow\}\}$. Furthermore, for each variable v such that $\xi(v) = a$, let q_v be the maximal connected subquery of $q|_{V_\xi(a)}$ such that $v \in \mathsf{VI}(q_v)$. We define

$$q|_v^\xi = q_v \cup \{A_0(v') \mid v' \in \mathsf{VI}(q_v), \xi(v') = a\}.$$

In order to decide $\mathcal{K} \not\models Q$, we need to decide whether there exists an ABox completion that can be extended into a model where there is no match for Q, that is, that we can extend it in such a way that every candidate match does not become a full match.

A *query annotation* for an ABox completion provides a selection of subqueries $\alpha(a)$ for each individual a. Intuitively, any candidate match for Q in this completion would require that some query in $\alpha(a)$ has a match in the tree T_a rooted at a to become a full match. Hence, if for each a there is some T_a where no q in $\alpha(a)$ has a match (i.e., all the queries in $\alpha(a)$ can be avoided for a), then there is a canonical model that extends this completion in which every match candidate fails, and hence there is no match for Q.

Definition 6.2.5 *Let θ be an ABox completion for $\mathcal{K}$ and let Q be a UCQ. A* query annotation *for θ and Q is a function α that assigns to each individual $a \in \mathsf{N_I}(\mathcal{K})$ a set of CQs such that:*

(a) $Q \subseteq \alpha(a)$,

(b) for every $q \in Q$ and for every match candidate ξ for θ and q, there is some $v \in \mathsf{VI}(q)$ such that $\xi(v) = a$ and $q|_v^\xi \in \alpha(a)$, and

(c) every $q \in \alpha(a)$ is such that $q = q|_v^\xi$ for some match candidate ξ and some $v \in \mathsf{VI}(q)$.

Note that, by definition, for every individual a and every query annotation α, each $q \in \alpha(a)$ is connected.

The following theorem follows from the results in [GHLS08, Lut08a].

Theorem 6.2.6 *$\mathcal{K} \not\models Q$ iff there is a completion θ and a query annotation α for θ and Q such that, for every $a \in \mathsf{N_I}(\mathcal{K})$, $\mathcal{K}_{\theta(a)} \not\models \alpha(a)$.*

In this way, we reduce the non-entailment of a CQ over arbitrary KBs to a collection of instances of the non-entailment problem of connected UCQs over simple KBs. To decide the latter, we employ knots.

Example 6.2.7 *There are many match candidates for the completion θ_2. For example, we can consider $\xi(v_1) = a$, $\xi(v_2) = b$, $\xi(v_3) = a_\downarrow$ and $\xi(v_4) = a$. Intuitively, it suggests a query match that binds v_2 to $b^{\mathcal{I}_2} = 2$. The rest of the variables are matched in the tree rooted at $a^{\mathcal{I}_2} = 1$ (more specifically, v_1 and v_2 to 1, and v_3 inside the tree), generating two disconnected subqueries for which entailment at $\mathcal{K}_a$ has to be tested. More specifically, the match candidate induces three subqueries*

$$\begin{aligned} q_1|_{v_1}^\xi &= \{A_0(v_1), A(v_1)\} \\ q_1|_{v_2}^\xi &= \{A_0(v_2), B(v_2)\} \\ q_1|_{v_3}^\xi = q_1|_{v_4}^\xi &= \{A_0(v_4), D(v_3), p_2(v_3, v_4), D(v_4)\} \end{aligned}$$

To show that $\mathcal{K} \not\models q_1$, we need to find a query annotation α such that $\mathcal{K}_a \not\models \alpha(a)$ and $\mathcal{K}_b \not\models \alpha(b)$. Furthermore, such a query annotation must contain one of the three queries above (since it must 'spoil' ξ). We can not have $q_1|_{v_2}^\xi \in \alpha(b)$ because $\mathcal{K}_b \models q_1|_{v_2}^\xi$, hence one of $q_1|_{v_1}^\xi$ or $q_1|_{v_3}^\xi$ must be in $\alpha(a)$. It is not hard to see that every other candidate match ξ' that has $\xi'(v) = b$ or $\xi'(v) = b_\downarrow$ can be spoiled with the subquery $q_1|_v^{\xi'}$ (because $q_1|_{v_2}^\xi$ is in fact the only subquery of q_1 entailed by

$\mathcal{K}_b$). Hence, if there is a query annotation that witnesses $\mathcal{K} \not\models q_1$, then there is one that only contains one of $q_1|^{\xi}_{v_1}$ or $q_1|^{\xi}_{v_3}$, and a set of queries q' such that each of them is of the form

$$q' = q_1 \cup \{A_0(v) \mid v \in V\}$$

for some (possibly empty) $V \subseteq \mathsf{Vl}(q_1)$ (note that if $\mathcal{K}_a \not\models q_1$, then $\mathcal{K}_a \not\models q'$ for each q'). We remark that, in this particular case, it suffices to show that $\mathcal{K}_a \not\models \alpha'_1(a) = \{q_1, \{A_0(v_1), A(v_1)\}\}$ to establish $\mathcal{K}_1 \not\models q_1$.

6.3 Reasoning in simple $\mathcal{ALCHI}$ KBs using Knots

In this section we introduce knots, and show how they can be employed to decide the existence of a canonical model for a simple KB. We adapt the technique to query answering in the next section.

6.3.1 Knots

Let $\mathcal{K} = \langle \{A_0(a)\}, \mathcal{T}, \mathcal{R} \rangle$ be a simple $\mathcal{ALCHI}$ KB. The aim of the *knot technique* is to obtain a finite representation of potentially infinite canonical models of $\mathcal{K}$, by decomposing them into a collection of small pieces. Each such piece is described by a *knot*, a schematic labeled tree of depth ≤ 1 with bounded branching, where nodes are labeled sets of concepts, and arcs are labeled with sets of roles. By restricting ourselves to only the *relevant* concepts and roles that occur in $\mathcal{K}$, we achieve that only finitely many distinct knots exist, and hence that models of $\mathcal{K}$ can be finitely represented. Formally, a knot is defined as follows:

Definition 6.3.1 (Knot) *A* knot *for $\mathcal{K}$ is a pair $(\mathbf{t}, \mathbf{S})$ that consists of a concept type $\mathbf{t} \in \mathsf{typesC}(\mathcal{K})$, called* root*, and a set $\mathbf{S}$ of* children*, which are pairs $(\mathbf{r}, \mathbf{t}')$ of a role type $\mathbf{r} \in \mathsf{typesR}(\mathcal{K})$ and a concept type $\mathbf{t}' \in \mathsf{typesC}(\mathcal{K})$, such that $|\mathbf{S}| \leq |Cl_{\mathsf{C}}(\mathcal{K})|$.*

A knot describes a possible node w in a canonical model of a simple KB $\mathcal{K}$ with all its successors, fixing the concepts that are satisfied at each node and the roles connecting the nodes. More specifically, it describes a node w that satisfies $\mathbf{t}$, and that has a successor w' related to w via the roles in $\mathbf{r}$, that satisfies the concepts in $\mathbf{t}'$, for each pair $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$.

Example 6.3.2 *Some knots for $\mathcal{K}_a$ are depicted in Figure 6.4. Five different concept types occur in these knots:*

$$\begin{aligned}
\mathbf{t}_A &= \{A, \exists p_1.A\} \\
\mathbf{t}_{A,B} &= \{A, B, \exists p_1.A\} \\
\mathbf{t}_{A,D} &= \{A, D, \exists p_1.A, \exists p_2{}^-.D\} \\
\mathbf{t}_{B,D} &= \{B, D, \exists p_2{}^-.D, \exists p_2{}^-.(A \sqcap B)\} \\
\mathbf{t}_{A_0,B,D} &= \{A_0, B, D, \exists p_2{}^-.D, \exists p_2{}^-.(A \sqcap B)\}
\end{aligned}$$

The role types that occur are $\mathbf{r}_{p_1} = \{p_1\}$ and $\mathbf{r}_{p_2{}^-} = \{p_2{}^-\}$ as in the previous example. Graphically, we represent a knot $\mathbf{k} = (\mathbf{t}_0, \mathbf{S})$ as a tree of depth at most one. The root is labeled $\mathbf{t}_0$ and it has a child labeled $\mathbf{t}'$ for each $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$. Instead of writing the role types as labels, we simply use downward dashed blue arrows for $\mathbf{r}_{p_1}$ and upward solid black arrows for $\mathbf{r}_{p_2{}^-}$.

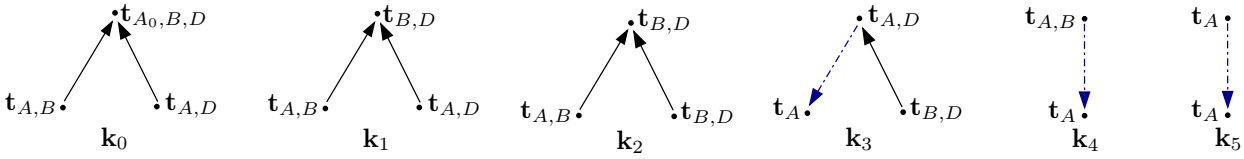

Figure 6.4: A set of knots $\mathbb{K}_a$ for $\mathcal{K}_a$

6.3.2 Satisfiability of Simple $\mathcal{ALCHI}$ KBs using Knots

Every canonical model of $\mathcal{K}$ can be decomposed into a finite knot set. Conversely, knots can be viewed as the 'building blocks' for a potential canonical model. Hence, to decide satisfiability of $\mathcal{K}$, we only need to decide whether there is a set of knots that represents a model of $\mathcal{K}$. To ensure that knots indeed represent a model, two kinds of conditions are imposed:

1. *local consistency conditions*, which apply to each knot individually and ensure that it represents an abstract domain element with its immediate successors in a canonical model, as required by the constraints given in the KB.
2. *global conditions*, which apply to the (whole) knot set and ensure that suitable instances of the knots in the set can be composed into full models.

We first define the local consistency conditions. They ensure that the node described by the root of the knot satisfies all the axioms in $\mathcal{T}$ and $\mathcal{R}$, and that there are no contradictions in it that could prevent it from occurring in a model.

Definition 6.3.3 (Knot consistency) *A knot* $(\mathbf{t}, \mathbf{S})$ *is* $\mathcal{K}$-consistent *if it obeys the following conditions:*

1. *if* $\exists P.C \in \mathbf{t}$*, then* $P \in \mathbf{r}$ *and* $C \in \mathbf{t}'$ *for some* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$*;*
2. *if* $\forall P.C \in \mathbf{t}$*, then* $C \in \mathbf{t}'$ *for all* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$ *with* $P \in \mathbf{r}$*; and*
3. *if* $\forall P.C \in \mathbf{t}'$ *for some* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$ *and* $\mathsf{Inv}(P) \in \mathbf{r}$*, then* $C \in \mathbf{t}$*.*

A consistent knot complies with all the constraints described in $\mathcal{T}$ and $\mathcal{R}$, but this does not yet guarantee that it can be part of a canonical model, as there could be existential restrictions at child nodes that cannot be expanded into a full model. We therefore also need a global condition which guarantees that such an expansion is always possible.

Definition 6.3.4 (Coherency of knot sets) *Given a knot set* $\mathbb{K}$*, a knot* $(\mathbf{t}, \mathbf{S}) \in \mathbb{K}$ *is* good in $\mathbb{K}$*, if for each* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$*, there is a knot* $(\mathbf{t}', \mathbf{S}') \in \mathbb{K}$ *for some* $\mathbf{S}'$*. We call* $\mathbb{K}$ $\mathcal{K}$-coherent *if (i) each knot* $(\mathbf{t}, \mathbf{S}) \in \mathbb{K}$ *is* $\mathcal{K}$*-consistent and good in* $\mathbb{K}$*, and (ii) there is a knot* $(\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}$ *with* $A_0 \in \mathbf{t}_0$*.*

The existence of a $\mathcal{K}$-coherent knot set precisely characterizes the satisfiability of $\mathcal{K}$:

Theorem 6.3.5 *$\mathcal{K}$ is consistent iff there exists a $\mathcal{K}$-coherent knot set.*

Proof. If we have a $\mathcal{K}$-coherent set of knots $\mathbb{K}$, then we can easily build a canonical model of $\mathcal{K}$: we start with a knot $(\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}$ with $A_0 \in \mathbf{t}_0$, repeatedly append suitable successor knots $(\mathbf{t}', \mathbf{S}')$ to each leaf of the tree with type $\mathbf{t}'$. Such a successor always exists because each knot is

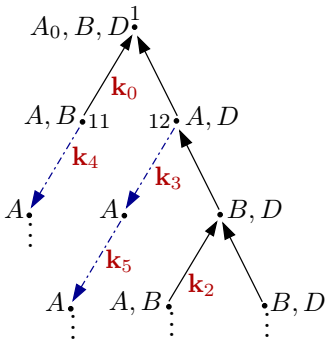

Figure 6.5: A canonical model $\mathcal{I}_a$ for $\mathcal{K}_a$

good in $\mathbb{K}$. A simple inspection of Definition 6.1.3 suffices to see that this construction results in a canonical model of $\mathcal{K}$.

Conversely, a tree-shaped canonical model $\mathcal{I}$ of $\mathcal{K}$ can be decomposed into a $\mathcal{K}$-coherent knot set in a straightforward way. We simply take a mapping ξ that assigns to each $w \in \Delta^{\mathcal{I}}$ a knot $\xi(w) = (\mathbf{t}_0, \mathbf{S})$ such that (i) $\mathbf{t}_0 = \{C \in \mathit{Cl}_{\mathsf{C}}(\mathcal{K}) \mid w \in C^{\mathcal{I}}\}$, and (ii) $\mathbf{S} = \bigcup_{w \cdot i \in \mathsf{succ}(w)} \{(\mathbf{r}_i, \mathbf{t}'_i)\}$, where $\mathbf{r}_i = \{P \mid (w, w \cdot i) \in P^{\mathcal{I}}\}$ and $\mathbf{t}'_i = \{C \in \mathit{Cl}_{\mathsf{C}}(\mathcal{K}) \mid w \cdot i \in C^{\mathcal{I}}\}$. The knot set $\mathbb{K}' = \{\xi(w) \mid w \in \Delta^{\mathcal{I}}\}$ is $\mathcal{K}$-coherent by construction. □

Example 6.3.6 *Recall the set of knots $\mathbb{K}_a$ shown in Figure 6.4. Every knot in $\mathbb{K}_a$ is $\mathcal{K}_a$-consistent and good in $\mathbb{K}_a$. Since the root of $\mathbf{k}_0$ contains A_0, the set is $\mathcal{K}_a$-coherent and witnesses the satisfiability of $\mathcal{K}$. In fact, the canonical model $\mathcal{I}_a$ of $\mathcal{K}_a$ shown in Figure 6.5 (which is essentially the subtree of $\mathcal{I}_2$ rooted at a) can be constructed from $\mathbb{K}_a$; the knot to which each model part corresponds is indicated in the figure.*

A simple algorithm for testing existence of a nonempty $\mathcal{K}$-coherent knot set, similar to the well-known type-elimination method due to Pratt [Pra79], is shown in Figure 6.6. Starting from the set of all knots that are consistent w.r.t. $\mathcal{K}$, the algorithm eliminates the knots that are not good. If no more knots need to be removed and a knot $(\mathbf{t}_0, \mathbf{S})$ with $A_0 \in \mathbf{t}_0$ remains then the set of knots is $\mathcal{K}$-coherent and witnesses the satisfiability of $\mathcal{K}$.

6.4 Query Answering by Knot Elimination

We are left with the task of deciding $\mathcal{K} \models Q$, where $\mathcal{K}$ is simple and Q is a connected UCQ. We again proceed in two steps. In the first step we give a knot-based algorithm for UCQ entailment in $\mathcal{ALCHI}$ that presupposes a restricted form of queries that are tree-shaped, and decides a special form of entailment that we call *directed* entailment. In a second second stage we reduce the standard entailment of arbitrary UCQs to this restricted setting.

6.4.1 Non-Entailment of a Set of Tree-shaped Queries

We start by introducing a restricted kind of CQs that are *tree-shaped*.

Definition 6.4.1 (tree-shaped queries) *A CQ q is* tree-shaped *if there is a bijection ψ between the variables in $\mathsf{Vl}(q)$ and the nodes of a proper tree (that is, a tree with root ε) such that $p(v, v') \in \mathsf{Atoms}(q)$ implies $\psi(v') \in \mathsf{succ}(\psi(v))$. We denote by v_w the variable v such that $\psi(v) = w$. For a variable v_w, we denote by $q|_{v_w}$ the tree-shaped query obtained by restricting q to the variables $v_{w \cdot w'}$, $w' \in \mathbb{N}^*$.*

Algorithm 1: satisfiability

Input: a simple KB $\mathcal{K} = \langle \{A_0(a)\}, \mathcal{T}, \mathcal{R} \rangle$
begin
  Compute the set $\mathbb{K}$ of all knots that are $\mathcal{K}$-consistent;
  repeat
    $\mathbb{K}' := \mathbb{K}$;
    $\mathbb{K} := \mathbb{K}' \setminus \{(\mathbf{t}, \mathbf{S}) \in \mathbb{K}' \mid (\mathbf{t}, \mathbf{S})$ is not good in $\mathbb{K}'\}$;
  until $\mathbb{K} \neq \mathbb{K}'$;
  if *there is some* $(\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}$ *with* $A_0 \in \mathbf{t}_0$ **then**
    return *"satisfiable"*
  else
    return *"unsatisfiable"*
end

Figure 6.6: The knot elimination algorithm for satisfiability of simple KBs.

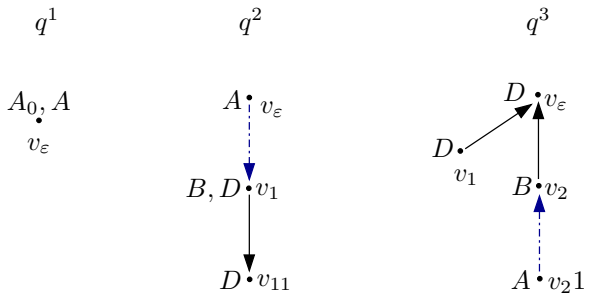

Figure 6.7: A set Q_a of tree-shaped queries

Note that the graph of every tree-shaped query is a tree whose root is the variable v_ε, and where each variable v_w has as successors variables of the form $v_{w \cdot i}$, $i \in \mathbf{N}$. Furthermore, by our naming convention, each variable $v_{w \cdot w'}$ is renamed to $v_{w'}$ in the restricted query $q|_{v_w}$.

Example 6.4.2 *We consider a set of tree-shaped queries* $Q_a = \{q^1, q^2, q^3\}$*; the (graphs of the) queries are depicted in Figure 6.7.*

For tree-shaped queries, we define a special kind of *directed entailment* in canonical interpretations.

Definition 6.4.3 (Directed Entailment) *Let* $\mathcal{I}$ *be a canonical interpretation, let* $w \in \Delta^{\mathcal{I}}$*, let* Q *a set of tree-shaped queries, and let* $q \in Q$*. Then we write:*

- $\mathcal{I} \models^{\downarrow} q[w]$ *if there exists a match* π *for* q *in* $\mathcal{I}$ *such that* $\pi(v_\varepsilon) = w$*, and for every* $r(v, v') \in q$ *we have* $\pi(v') = \pi(v) \cdot i$ *for some* $i \in \mathbf{N}$*;*
- $\mathcal{I} \models^{\downarrow} q$ *if* $\mathcal{I} \models^{\downarrow} q[w]$ *for some* $w \in \Delta^{\mathcal{I}}$*;*
- $\mathcal{I} \models^{\downarrow} Q$ *if* $\mathcal{I} \models^{\downarrow} q$ *for some* $q \in Q$*; and*
- $\mathcal{K} \models^{\downarrow} Q$ *if* $\mathcal{I} \models^{\downarrow} Q$ *for every canonical model* $\mathcal{I}$ *of the knowledge base* $\mathcal{K}$*.*

Observe that the matches used in Definition 6.4.3 are directed in the sense that we ignore the possible relations between a node and its parent in a canonical interpretation. Clearly, the existence of a directed match implies the existence of an ordinary match. The converse holds if the interpretation is one-way, but not in general. Hence in case $\mathcal{K}$ is an $\mathcal{ALCH}$ KB we have $\mathcal{K} \models Q$ iff $\mathcal{K} \models^{\downarrow} Q$ by Proposition 6.1.4. For $\mathcal{ALCHI}$ only the if direction can be ensured, but we show below how this kind of entailment can be used to decide general entailment.

The algorithm devised in this section decides, given an $\mathcal{ALCHI}$ KB $\mathcal{K}$ and a set of tree-shaped queries Q, whether $\mathcal{K} \models^{\downarrow} Q$. The following characterization of directed entailment is easy to establish.

Proposition 6.4.4 *Let $\mathcal{I}$ be a canonical model of an $\mathcal{ALCHI}$ KB $\mathcal{K}$, let $w \in \Delta^{\mathcal{I}}$, and let q be a tree-shaped query. Then $\mathcal{I} \not\models^{\downarrow} q[w]$ iff one of the following holds:*

(i) $\{A \mid A(v_\varepsilon) \in q\} \not\subseteq \{A \mid w \in A^{\mathcal{I}}\}$ *or*

(ii) there exists a variable $v_{\varepsilon \cdot i}$ of q such that for each child $w \cdot j \in \Delta^{\mathcal{I}}$ of w we have:

a) $\{r \mid r(v_\varepsilon, v_{\varepsilon \cdot i}) \in q\} \not\subseteq \{r \mid (w, w \cdot j) \in r^{\mathcal{I}}\}$, *or*
b) $\mathcal{I} \not\models^{\downarrow} q|_{v_{\varepsilon \cdot i}}[w \cdot j]$.

For the rest of the section, fix a simple $\mathcal{ALCHI}$ KB $\mathcal{K}$ and a set of tree-shaped queries Q. The aim of our algorithm is to decide whether $\mathcal{K} \models^{\downarrow} Q$ by using knots to verify the existence of a canonical model $\mathcal{I}$ of $\mathcal{K}$ with $\mathcal{I} \not\models^{\downarrow} Q$ (a *countermodel* for Q). Proposition 6.4.4 suggests that we can do this by extending knots with auxiliary information that enables us to track the satisfaction of conditions (i) and (ii) for each query q in Q at each node w of the tree.

Definition 6.4.5 (Tree subqueries, Marked knots) *Let $\mathsf{subq}(Q)$ denote the smallest set such that (i) $Q \subseteq \mathsf{subq}(Q)$, and (ii) $q|_{v_{\varepsilon \cdot i}} \in \mathsf{subq}(Q)$ for each $q \in \mathsf{subq}(Q)$ and each variable $v_{\varepsilon \cdot i}$ of q. A* Q-marked *knot is a tuple $(\mathbf{t}, \mathbf{S}, \nu)$ where $(\mathbf{t}, \mathbf{S})$ is a knot and $\nu : \{\mathbf{t}\} \cup \mathbf{S} \to 2^{\mathsf{subq}(Q)}$ is a* marking function *that assigns to each node in the knot a set of subqueries.*

Intuitively, every node in a Q-marked knot is labeled with the set of those tree subqueries of Q for which a (directed) match of the root should be *avoided* at that node. To capture this formally, we define additional local conditions.

Definition 6.4.6 (Query avoiding knots) *A Q-marked knot $(\mathbf{t}, \mathbf{S}, \nu)$ is* Q-avoiding, *if for each $q \in Q$, we have $q \in \nu(\mathbf{t})$, and one of the following holds:*

(i) $\{A \mid A(v_\varepsilon) \in q\} \not\subseteq \mathbf{t}$, *or*

(ii) there exists some variable $v_{\varepsilon \cdot i}$ such that for every $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$, it holds that either

a) $\{r \mid r(v_\varepsilon, v_{\varepsilon \cdot i}) \in q\} \not\subseteq \mathbf{r}$, *or*
b) $q|_{v_{\varepsilon \cdot i}} \in \nu((\mathbf{r}, \mathbf{t}'))$.

The above just mimics the conditions in Proposition 6.4.4.

Example 6.4.7 *A set $\mathbb{K}'_a$ of Q_a-avoiding marked knots is shown in Figure 6.8, they are a marked version of the knots in $\mathbb{K}_a$. For each knot $(\mathbf{t}, \mathbf{S}) \in \mathbb{K}_a$, we label $\mathbf{t}$ with its marking $\nu(\mathbf{t})$ and, for each child $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$, we label $\mathbf{t}'$ with $\nu(\mathbf{t}')$. For readability, we write $q^{1,2,3}$ instead of q^1, q^2, q^3. they are all Q_a-avoiding. For example, for $\mathbf{k}_0 = (\mathbf{t}_0, \mathbf{S}_0)$ with $\mathbf{t}_0 = \mathbf{t}_{A_0,B,D}$, we have $Q_a \subseteq \nu(\mathbf{t}_0)$. For q^1 and q^3, item (i) in Definition 6.4.6 holds. For q^2 we have $\{A \mid A(v_\varepsilon) \in q\} = \{D\} \subseteq \mathbf{t}_0$, but there is one variable, namely v_2, such that$q|_{v_3} \in \nu((\mathbf{r}, \mathbf{t}'))$ for each $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}_0$, hence (ii) holds.*

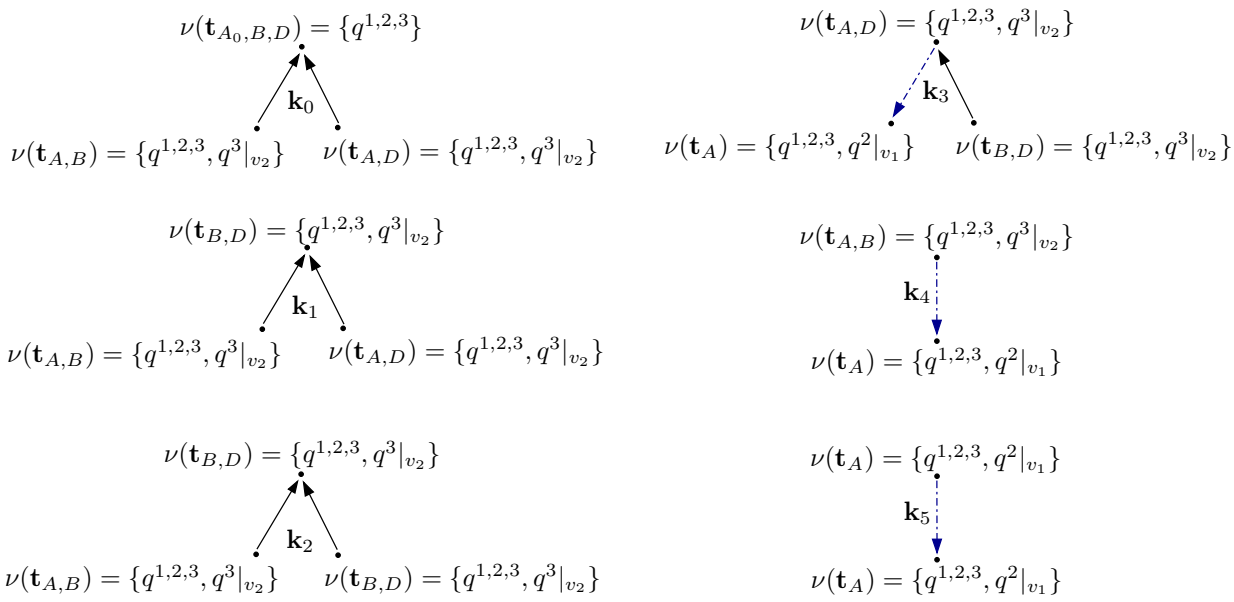

Figure 6.8: A set $\mathbb{K}'_a$ of Q_a avoiding knots

To make sure that a full countermodel can be constructed, the marking must be consistent across knots. This is achieved by extending the coherence condition to marked knots in the natural way.

Definition 6.4.8 (Coherency of marked knot sets) *For a set $\mathbb{K}$ of Q-marked knots, we call $(\mathbf{t}, \mathbf{S}, \nu) \in \mathbb{K}$* good in $\mathbb{K}$ *if for each $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$, there is some $(\mathbf{t}', \mathbf{S}', \nu') \in \mathbb{K}$ such that $\nu((\mathbf{r}, \mathbf{t}')) = \nu'(\mathbf{t}')$. Then $\mathbb{K}$ is* $\mathcal{K}$-coherent *if:*

- *for each $(\mathbf{t}, \mathbf{S}, \nu) \in \mathbb{K}$,*
  - *$(\mathbf{t}, \mathbf{S})$ is $\mathcal{K}$-consistent,*
  - *$(\mathbf{t}, \mathbf{S}, \nu)$ is Q-avoiding and good in $\mathbb{K}$, and*
- *there is some $(\mathbf{t}_0, \mathbf{S}, \nu) \in \mathbb{K}$ such that $A_0 \in \mathbf{t}_0$.*

We then obtain the following characterization of directed entailment.

Proposition 6.4.9 *$\mathcal{K} \not\models^{\downarrow} Q$ iff there exists a $\mathcal{K}$-coherent set of Q-marked knots.*

This can be shown arguing as in the proof of Theorem 6.3.5, additionally using Proposition 6.4.4.

Example 6.4.10 *The knots in $\mathbb{K}'_a$ are marked versions of $\mathcal{K}_a$-consistent knots, they are all Q_a-avoiding, and they are all good in $\mathbb{K}_a$. Hence the set is $\mathcal{K}_a$-coherent and it witnesses that $\mathcal{K}_a \not\models Q_a$.*

The existence of $\mathcal{K}$-coherent set of Q-marked knots can be decided with the *knot elimination* algorithm presented in Figure 6.9.

The following properties of the algorithm will be important to establish our complexity results in Section 6.5. In particular the second claim, which may seem trivial, will be useful when we analyze the data complexity of the algorithm.

Algorithm 2: countermodel

Input: a simple KB $\mathcal{K} = \langle\{A_0(a)\}, \mathcal{T}, \mathcal{R}\rangle$, a set of tree-shaped queries Q

begin

Compute the set $\mathbb{K}$ of all Q-marked knots $(\mathbf{t}, \mathbf{S}, \nu)$ for $\mathcal{K}$ such that $(\mathbf{t}, \mathbf{S})$ is $\mathcal{K}$-consistent and $(\mathbf{t}, \mathbf{S}, \nu)$ is Q-avoiding ;

repeat

$\mathbb{K}' := \mathbb{K}$;

$\mathbb{K} := \mathbb{K}' \setminus \{(\mathbf{t}, \mathbf{S}, \nu) \in \mathbb{K}' \mid (\mathbf{t}, \mathbf{S}, \nu)$ is not good in $\mathbb{K}\}$;

until $\mathbb{K} \neq \mathbb{K}'$;

if *there is* $(\mathbf{t}_0, \mathbf{S}, \nu) \in \mathbb{K}$ *with* $A_0 \in \mathbf{t}_0$ **then**

return *" a counter model exists"*

else

return *"a counter model does not exist"*

end

Figure 6.9: The knot elimination algorithm for directed entailment of tree-shaped queries.

Theorem 6.4.11 *Given a simple KB $\mathcal{K} = \langle\{A_0(a)\}, \mathcal{T}, \mathcal{R}\rangle$ and a set of tree-shaped queries Q,*

1. *$\mathcal{K} \models^{\downarrow} Q$ can be decided in time exponential in $||\mathcal{K}|| + ||Q||$, and*
2. *if $|Cl_{\mathsf{C}}(\mathcal{K})|$, $|\overline{\mathsf{N_R}}(\mathcal{K})|$ and $||Q||$ are bounded by some constant, $\mathcal{K} \models^{\downarrow} Q$ can be decided in constant time.*

Proof. Let $c = |Cl_{\mathsf{C}}(\mathcal{K})|$, let $t = |\overline{\mathsf{N_R}}(\mathcal{K})|$, and let $u = |\mathsf{subq}(Q)|$. As $|\mathsf{typesC}(\mathcal{K})|$, i.e., the number of distinct concept types for $\mathcal{K}$, is bounded by 2^c, and $|\mathsf{typesC}(\mathcal{K})|$, i.e., the number of distinct role types for $\mathcal{K}$, is bounded by 2^t. Hence the number of distinct knots for $\mathcal{K}$ is bounded by $2^c{\cdot}(2^t{\cdot}2^c)^c = 2^{c+tc+c^2}$. The number of distinct subsets of $\mathsf{subq}(Q)$ that can be used as 'markers' by a marking function ν is bounded by 2^u, so the number of distinct Q-marked knots for $\mathcal{K}$ is bounded by $k = 2^c{\cdot}2^u{\cdot}(2^t{\cdot}2^c{\cdot}2^u)^c \leq 2^{2c(c+t+u)}$. It is easy to see that c and t are linear in $||\mathcal{K}||$. The number $u = |\mathsf{subq}(Q)|$ of tree-shaped subqueries of Q is linearly bounded by the number of queries in Q times the maximal number or variables in each query, and is thus polynomial in $||Q||$. Hence the number k of distinct Q-marked knots for $\mathcal{K}$ is single exponential in $||\mathcal{K}|| + ||Q||$. Checking $\mathcal{K}$-consistency and Q-avoidance are clearly polynomial in $||\mathcal{K}|| + ||Q||$, hence constructing the set $\mathbb{K}$ in the first step of the algorithm is feasible in time exponential in $||\mathcal{K}|| + ||Q||$. In the subsequent 'clean-up' stage, each test is feasible in linear time in $||\mathbb{K}|| \leq k$. Each knot is removed at most once and never introduced again, hence the algorithm stops after at most k steps. Checking for the existence of a knot containing A_0 is linear in $||\mathbb{K}||$, so the algorithm terminates in time that is exponential in $||\mathcal{K}|| + ||Q||$.

The second item is now trivial: if c, t and u are bounded by some constant, then the number k of distinct Q-marked knots is also bounded by a constant, and the whole algorithm requires only constant time. □

6.4.2 From Standard Entailment to Directed Entailment

We now show how Theorem 6.4.11 can be used to derive tight complexity bounds for standard entailment of arbitrary UCQs over simple KBs. Consider a simple KB $\mathcal{K} = \langle\{A_0(a)\}, \mathcal{T}, \mathcal{R}\rangle$, and an arbitrary UCQ Q. Recall that by Proposition 6.1.4, checking $\mathcal{K} \not\models Q$ is equivalent to ensuring that none of the tree-shaped canonical models of $\mathcal{K}$ admits a match for a query $q \in Q$. The

existence of a match for q in such a model can be characterized in terms of a set of tree-shaped queries that can be obtained from q by unifying variables and inverting role atoms.

Definition 6.4.12 *A tree-shaped CQ q' is a* tree rewriting *of a CQ q if there is a surjective map $\phi : \mathsf{VI}(q) \to \mathsf{VI}(q')$ such that*

(i) $A(v) \in q$ iff $A(\phi(v)) \in q'$, and

(ii) $P(v, v') \in q$ iff $P(\phi(v), \phi(v')) \in q'$ or $\mathsf{Inv}(P)(\phi(v'), \phi(v)) \in q'$.

Let $\mathsf{TRew}(q)$ denote the set of all tree rewritings of q. For a UCQ Q, we let $\mathsf{TRew}(Q) = \bigcup_{q \in Q} \mathsf{TRew}(q)$.

The following is easy to prove.

Proposition 6.4.13 *If $\mathcal{I}$ is a canonical model of a simple KB $\mathcal{K}$ and q is a connected CQ in $\mathcal{ALCHI}$, then $\mathcal{I} \models q$ iff $\mathcal{I} \models^{\downarrow} \mathsf{TRew}(q)$.*

Proof. It is easy to verify that a directed match for a query $q' \in \mathsf{TRew}(q)$ is also a match for q. On the other hand, any match π for q in $\mathcal{I}$ gives rise to a rewriting q_π of q as follows:

- $\mathsf{VI}(q_\pi) = \{v_w \mid \exists v \in \mathsf{VI}(q) : \pi(v) = w\}$, i.e., there is a variable v_w for each $w \in \Delta^{\mathcal{I}}$ that is in the range of π;
- the concept atoms of q_π are $\{A(v_w) \mid \exists\, A(v) \in q : \pi(v) = w\}$;
- the role atoms are $\{P(v_w, v_{w \cdot i}) \mid \exists\ P(v, v') \in q : \pi(v) = w \text{ and } \pi(v') = w \cdot i\} \cup \{\mathsf{Inv}(P)(v_w, v_{w \cdot i}) \mid \exists\, P(v, v') \in q : \pi(v') = w \text{ and } \pi(v) = w \cdot i\}$

Note that in general, as π does not need to be surjective, q_π may have less variables than q. Since q is connected and $\mathcal{I}$ is a canonical model, q_π is obviously tree-shaped.Moreover, the construction of q_π ensures that $\mathcal{I} \models^{\downarrow} q_\pi$, i.e., there is a *directed* match for q_π in $\mathcal{I}$. □

The following is a direct consequence of the proposition above and the canonical model property (Proposition 6.1.4).

Proposition 6.4.14 *For a simple KB $\mathcal{K}$ and a connected UCQ Q in $\mathcal{ALCHI}$, $\mathcal{K} \models Q$ iff $\mathcal{K} \models^{\downarrow} \mathsf{TRew}(Q)$.*

Example 6.4.15 *Figure 6.10 shows some tree rewritings of q_1. Note that both the left-most and the right-most queries are in the set Q_a and, as we have shown, are not entailed by $\mathcal{K}_a$. Furthermore, recall the set $\alpha_1'(a) = \{q_1, \{A_0(v_1), A(v_1)\}\}$ from Example 6.2.7. A simple (but tedious) inspection shows that* all *the tree rewritings of q_1 can be incorporated into a $\mathsf{TRew}(q) \cup \{A_0(v_1), A(v_1)\}$-avoiding marked set of knots. Hence $\mathcal{K}_a \not\models \alpha_1'(a)$ and as discussed, this implies $\mathcal{K}_1 \not\models q_1$.*

A central observation is that in the case of $\mathcal{ALCH}$, we can replace the set $\mathsf{TRew}(q)$ in Proposition 6.4.13 with a single CQ. If there is a match for q in some one-way canonical interpretation, then there is single tree-shaped rewriting q' of q that has a directed match in every one-way canonical interpretation where q has a match. In fact, we can obtain this rewriting in a very easy way (and in polynomial time): simply eliminate all *forks* $p(v_1, v_2), p'(v_1', v_2)$ in q by unifying the variables v_1 and v_1'. Observe that, in contrast to the case of $\mathcal{ALCHI}$, the rewriting is independent of some concrete interpretation and a concrete match for q in it. As noted already in Section 6.4.1, we then have $\mathcal{K} \models q'$ iff $\mathcal{K} \models^{\downarrow} q'$, and we can directly employ the algorithm in the previous section.

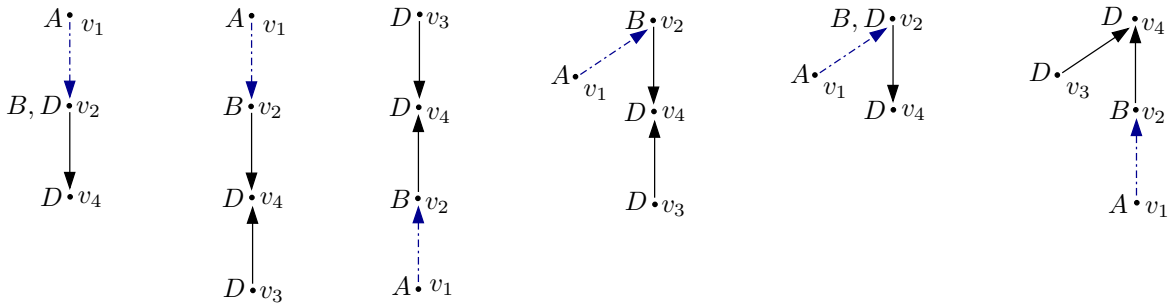

Figure 6.10: Some tree rewritings of q_1

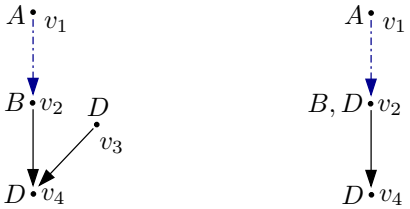

Figure 6.11: The query q_1 and its fork-elimination $\mathsf{FE}(q_1)$

Definition 6.4.16 (Fork elimination [Lut08a]) *For a CQ q, a* fork elimination *of q is a query obtained from q by exhaustively applying the following rule: if the query contains atoms $p(v_1, v_2)$ and $p'(v_1', v_2)$ where $v_1 \neq v_1'$, then replace every occurrence of v_1 with v_1'. By $\mathsf{FE}(q)$ we denote an arbitrary fork elimination of q.*

Example 6.4.17 *The fork elimination of the query q_1 is depicted in Figure 6.11.*

Note that fork eliminations of q coincide up to a renaming of variables. It is easy to see that, for each one-way tree-shaped canonical interpretation $\mathcal{I}$, every pair of variables v_1 and v_1' that are unified in the rewriting has $\pi(v_1) = \pi(v_1')$ in every match π for q in $\mathcal{I}$. That is, the elimination of forks preserves query entailment in one-way tree-shaped canonical interpretations, and hence it preserves query entailment over simple $\mathcal{ALCH}$ KBs. Hence we easily obtain:

Proposition 6.4.18 *If $\mathcal{I}$ is a canonical model of a simple KB $\mathcal{K}$ and q is a CQ in $\mathcal{ALCH}$, then $\mathcal{I} \models q$ iff $\mathcal{I} \models^{\downarrow} \mathsf{FE}(q)$. Hence $\mathcal{K} \models Q$ iff $\mathcal{K} \models^{\downarrow} \bigcup_{q \in Q} \{\mathsf{FE}(q)\}$.*

This concludes the reduction of regular UCQ entailment in $\mathcal{ALCH}$ and $\mathcal{ALCHI}$ to directed entailment of tree-shaped queries.

6.5 Complexity of Query Answering

Putting things together, we have an algorithm for deciding UCQ entailment in $\mathcal{ALCH}$ and $\mathcal{ALCHI}$ that consists of three steps:

1. Construct all the ABox completions θ for $\mathcal{K}$ and all the query annotations α for θ and Q.
2. For each pair (θ, α) of a completion θ and a query annotation α for it, construct for all individuals a the pair $(\mathcal{K}_{\theta(a)}, \mathsf{TreeQ}(a))$ of an induced simple knowledge base $\mathcal{K}_{\theta(a)}$ and a set of tree-shaped queries, where $\mathsf{TreeQ}(a) = \bigcup_{q \in \alpha(a)} \{\mathsf{FE}(q)\}$ if $\mathcal{K}$ and Q are in $\mathcal{ALCH}$, and $\mathsf{TreeQ}(a) = \mathsf{TRew}(\alpha(a))$ otherwise.

3. For each pair (θ, α), use Algorithm 6.9 to test for *all* pairs $(\mathcal{K}_{\theta(a)}, \mathsf{TreeQ}(a))$, whether there is a countermodel for $\mathcal{K}_{\theta(a)}$ and $\mathsf{TreeQ}(a)$.

We show in this section that the algorithm results in worst-case optimal complexity bounds, both for $\mathcal{ALCH}$ and $\mathcal{ALCHI}$, for both data and combined complexity.

6.5.1 Combined Complexity

Recall that, for a given $\mathcal{K}$ and Q, combined complexity measures the cost of deciding $\mathcal{K} \models Q$ in terms of $||\mathcal{K}|| + ||Q||$. We start with the combined complexity for $\mathcal{ALCH}$, which needs a more refined analysis.

Theorem 6.5.1 *Given a KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and a UCQ Q in $\mathcal{ALCH}$, deciding $\mathcal{K} \models Q$ is* ExpTime*-complete.*

Proof. Let $\ell = |\mathsf{N_I}(\mathcal{K})|$, $c = |Cl_{\mathsf{C}}(\mathcal{K})|$, let $t = |\overline{\mathsf{N_R}}(\mathcal{K})|$, and let $u = |Q|$ (when seen as a set of CQs), and recall that ℓ, c and t are all linear in $||\mathcal{K}||$, and u is linear in $||Q||$. Recall also that the number of distinct concept types for $\mathcal{K}$ is bounded by 2^c.

The number of different completions is bounded by $(\ell \cdot 2^c) \cdot (\ell^2 \cdot 2^t)$, hence it is polynomial in ℓ and single exponential in c and t, and thus single exponential in $||\mathcal{K}||$. For each completion, the size of the new CIA $A_0 \sqsubseteq C_{\theta(a)}$ is linear in c, and thus linear in $||\mathcal{K}||$. It was shown in [Lut08a] that, for each CQ q and each $\mathcal{ALCH}$ KB, there is an injective mapping from the subqueries $q|_v^\xi$ to the tree-shaped subqueries $\mathsf{subq}(\mathsf{FE}(q))$ of the fork elimination of q. Since $|\mathsf{subq}(\mathsf{FE}(q))|$ is linearly bounded in $||q||$, then for each q, the number of different $q|_v^\xi$ in an individual annotation $\alpha(a)$ is also linearly bounded in $||q||$. Hence the overall cardinality of $\alpha(a)$ is polynomially bounded in $||Q||$, and the same holds for its size $||\alpha(a)||$. Further, by Proposition 6.4.18, $\mathcal{K}_{\theta(a)} \models \alpha(a)$ iff $\mathcal{K}_{\theta(a)} \models^{\downarrow} \bigcup_{q \in \alpha(a)} \{\mathsf{FE}(q)\}$. For each $\alpha(a)$, $\bigcup_{q \in \alpha(a)} \{\mathsf{FE}(q)\}$ can be computed in time linear in $\alpha(a)$. This means that there are exponentially many pairs (θ, α), and for each of them we have to do a linear number of tests $\mathcal{K}_{\theta(a)} \models^{\downarrow} \bigcup_{q \in \alpha(a)} \{\mathsf{FE}(q)\}$, where $||\mathcal{K}_{\theta(a)}||$ and $|\bigcup_{q \in \alpha(a)} \{\mathsf{FE}(q)\}|$ are both polynomially bounded in $||\mathcal{K}|| + ||Q||$. We then obtain the upper bound by a direct application of Theorem 6.4.11.

The matching lower bound follows already from the well-known ExpTime-hardness of knowledge base satisfiability in $\mathcal{ALCH}$ [Sch91]. □

This ExpTime upper bound for $\mathcal{ALCH}$ was first shown in [OŠE08b] and in [Lut08a] (in the latter case, for the DL $\mathcal{ALCHQ}$ that additionally supports number restrictions). The proof we have provided here differs from both of them, as we discuss in Section 6.6.

A 2ExpTime upper bound for $\mathcal{ALCHI}$ follows from the results in the first part of this thesis—in particular from Theorem 3.4.2—and was already known in the literature (see Section 1.2). Now we show that the algorithm described in this chapter yields the same upper bound.

Theorem 6.5.2 *For a KB $\mathcal{K}$ and a UCQ Q in $\mathcal{ALCHI}$, deciding $\mathcal{K} \models Q$ is feasible in* 2ExpTime.

Proof. As above, there are exponentially many completions, and for each of them, the size of the induced knowledge bases $\mathcal{K}_{\theta(a)}$ is polynomial in $||\mathcal{K}||$. For annotations α, there are $(2\ell)^{|\mathsf{VI}(Q)|}$ many match candidates, where $\ell = |\mathsf{N_I}(\mathcal{K})|$ as above. Thus, using Theorem 6.2.6, we get that checking $\mathcal{K} \not\models Q$ is in 2ExpTime provided that checking $\mathcal{K}_{\theta(a)} \not\models \alpha(a)$ is in 2ExpTime. Using Proposition 6.4.14, $\mathcal{K}_{\theta(a)} \not\models \alpha(a)$ iff $\mathcal{K}_{\theta(a)} \not\models^{\downarrow} \mathsf{TRew}(\alpha(a))$. As for each $a \in \mathsf{N_I}(\mathcal{A})$, $||\alpha(a)||$ is at most single exponential in $||\mathcal{K}||$ and $||Q||$, then $\mathsf{TRew}(\alpha(a))$ is of single exponential size and can

be obtained in time single exponential in $||\mathcal{K}|| + ||Q||$. We obtain the upper bound by applying Theorem 6.4.11. □

The matching lower bound was a relatively long standing open problem, finally closed by Lutz [Lut07]: the above algorithm is worst case optimal, as query answering is 2ExpTime-hard in all extensions of $\mathcal{ALC}$ that support inverse roles.

6.5.2 Data Complexity

For the data complexity, we assume a fixed UCQ Q, TBox $\mathcal{T}$ and RBox $\mathcal{R}$, and we analyze the complexity of deciding $\langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle \models Q$, for a given ABox $\mathcal{A}$ such that only concept names occur in $\mathcal{A}$, and every concept and role name occurring in $\mathcal{A}$ also occurs in $\mathcal{T}$ or $\mathcal{R}$ (see Definition 2.3.3).

By Theorem 6.2.6, to check $\langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle \not\models Q$ we can non-deterministically build a completion θ, non-deterministically compute an annotation α, and then check for each individual a whether $\mathcal{K}_{\theta(a)} \not\models \alpha(a)$. By Proposition 6.4.14, the latter can be done by checking $\mathcal{K}_{\theta(a)} \not\models^{\downarrow} \mathsf{TRew}(\alpha(a))$ using our algorithm for directed entailment. Clearly, a completion can be computed in polynomial in $||\mathcal{A}||$. Since Q is fixed, an annotation for the completion can also be non-deterministically computed in polynomial time in $||\mathcal{A}||$. Indeed, there are only $(2 \cdot |\mathsf{N_I}(\mathcal{A})|)^{|\mathsf{N_V}(Q)|}$ match candidates, and for each of the candidates we pick one subquery which we assign to an individual. Since Q is fixed, for each individual $a \in \mathsf{N_I}(\mathcal{A})$, the resulting $\alpha(a)$ is of size bounded by a constant, and thus also $||\mathsf{TRew}(\alpha(a))||$ is bounded by a constant. If $\mathcal{T}$ and $\mathcal{R}$ are fixed, since every concept and role name occurring in $\mathcal{A}$ also occurs in $\mathcal{T}$ or $\mathcal{R}$, $|Cl_{\mathsf{C}}(\mathcal{K})|$ and $|\overline{\mathsf{N_R}}(\mathcal{K})|$ are also fixed. It then follows from Theorem 6.4.11 that checking $\mathcal{K}_{\theta(a)} \not\models^{\downarrow} \mathsf{TRew}(\alpha(a))$ requires only constant time.

This yields a non-deterministic polynomial time algorithm to check $\langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle \not\models Q$, and thus a coNP upper bound for UCQ entailment. This bound was shown for $\mathcal{ALCHIQ}$ in [OCE08], and was then extended to $\mathcal{SHIQ}$ in [GHLS08] (please see Section 7.5 for discussion). It is worst case optimal, as matching hardness holds already for DLs significantly weaker than $\mathcal{ALC}$ [Sch94a, CDGL$^+$06].

Theorem 6.5.3 *For a KB $\mathcal{K}$ and a UCQ Q in $\mathcal{ALCHI}$, deciding $\mathcal{K} \models Q$ is* coNP*-complete in data complexity.*

6.6 Related Work and Discussion

For a few years, only 2ExpTime upper bounds for query answering in expressive DLs were known, and the best matching lower bound was ExpTime-hardness stemming from knowledge base satisfiability. The first conclusive result concerning the precise complexity of the problem was the 2ExpTime lower bound for $\mathcal{ALCI}$ obtained by Lutz [Lut07]. In this chapter we used knots to show that UCQ answering is feasible in ExpTime for $\mathcal{ALCH}$, and in 2ExpTime for $\mathcal{ALCHI}$.

The ExpTime upper bound for $\mathcal{ALCH}$ is notable, since all previous query answering algorithms for expressive DLs required double exponential time. The technique we have described here, which differs from the ones originally used in [OŠE08b] and in [Lut08b] to show the same result, takes advantage of the more mature machinery that is emerging from the different algorithms proposed until now. This technique in particular was developed in [ELOŠ09a], but only for the restricted setting of CQ answering over simple knowledge bases. The extension to UCQs and arbitrary knowledge bases builds on ideas in [Lut08b, BEL$^+$10, GHLS08].

The upper bound given in [Lut08b] is for deciding the entailment of a Boolean CQ q over a KB in the DL $\mathcal{ALCHQ}$, which extends $\mathcal{ALCH}$ with qualified number restrictions. Similarly to

the algorithm we have presented here, it first uses some 'splits', which are closely related to query annotations, to generate a set of subqueries that have to be avoided in the tree-shaped parts of canonical interpretations. To test the existence of a countermodel for the latter subqueries—instead of using knots as in the algorithm above—Lutz employs the 'rolling-up' technique (see Section 4.3.1) to reduce the problem to knowledge base satisfiability in a DL that extends $\mathcal{ALCH}$ with role conjunctions.

The algorithm in [OŠE08b] answers a CQ q, possible with answer variables, over a KB $\mathcal{K}$ in $\mathcal{ALCH}$, and it uses knots, but in a different way. Roughly, it uses a more elaborate iterative procedure to compute in a bottom up way the combinations of tree-shaped subqueries of q that are entailed in the tree interpretations that start with a knot in a set. The key to showing ExpTime membership is that the size of the different combinations of subqueries that need to be computed is polynomially bounded; we will see that this is closely related to the fact that $\mathsf{FE}(q)$ has only polynomially many tree subqueries. The algorithm was extended to $\mathcal{SH}$ in [OŠE08a, EOŠ09] and will be presented in detail in the next chapter.

The goal of this chapter was not only to show ExpTime-completeness in $\mathcal{ALCH}$, but also to illustrate that, when annotated with suitable query information, knots are a flexible tool for solving the query entailment problem in a rather simple and intuitive way. We considered the DLs $\mathcal{ALCH}$ and $\mathcal{ALCHI}$, but the algorithm can be extended quite easily to other constructors, and allows for elegant refinements to obtain optimal bounds even in the presence of more subtle sources of complexity. Indeed, this knot marking technique has been adapted to show other tight complexity results in DLs with *transitive roles*. In their presence, it is not possible to generate a set of tree-shaped queries of polynomial size even when we are working with one-way models. However, if we disallow role inclusions, a somewhat intricate refinement of the approach can be used to show that in $\mathcal{S}$ (i.e., $\mathcal{ALC}$ with transitive roles), CQ entailment w.r.t. simple KBs is still in ExpTime [ELOŠ09b].

6.6.1 Related Techniques

The *knot* technique, as we have described it, was introduced in the context of non-monotonic Logic Programming with function symbols [ŠE07, EŠ10], but it can be seen as an instance of other reasoning methods that have been used for modal logics and other related fragments of first-order logic.

In particular, knots are a special instance of the *mosaic* technique [Ném86] that is well known in the context of modal logics. The basic idea underlying the technique is that models can be decomposed into a finite collection of small model parts called *mosaics*, and that if a finite set of mosaics is suitably *linked*, its elements can be combined into a model. Mosaics have been used to solve the formula satisfiability problem and to derive tight complexity upper bounds in some modal logics; see [MM07, BdRV01] and references therein. They have also been employed, although not widely, for standard reasoning in DLs [LST05, RKH08b].

Knots and mosaics are closely related to *types*. Roughly, a type is a small mosaic with only one element, and in compensation for the simplicity of the mosaics, more involved global conditions may be required. In fact, the elimination algorithm in Figure 6.9 is a simple variant of the famous *type elimination* algorithm proposed by Pratt for propositional dynamic logic [Pra79]. Type elimination and similar type based algorithms have become a very popular techniques that have been applied to a wide range of logics including, for example, various modal and description logics [PSV06, HM92, LWZ08], the 2-variable fragment of first-order logic [GKV97], and the extension of the latter with counting quantifiers [PH05]. We remark that the algorithm in Section 6.4.1 can also be formulated using *marked types* instead of marked knots, but knots seem to be better suited. Their more comprehensive representation of the local model structure allows for simpler local and global conditions, especially when extending the approach to more

expressive DLs such as those involving transitive roles. With the exception of [EGOŠ08, PH09], we are not aware of papers in which other variations of mosaics and types have been explored for query answering.

Chapter 7

Querying DLs with Transitive Roles and Role Hierarchies

In this chapter, we consider the problem of CQ entailment over knowledge bases in $\mathcal{SH}$, the DL that extends the basic $\mathcal{ALCH}$ with transitivity axioms. We present two main results:

1. We develop a knot-based algorithm for answering CQs over $\mathcal{SH}$ knowledge bases. It works in double exponential time in general, but in single exponential time for CQs with no transitive roles (i.e. for CQs in $\mathcal{ALCH}$), as well as for CQs only few (i.e., constantly many) atoms involving transitive roles. The latter restriction seems not to be severe in practice.
2. We show that CQ entailment in $\mathcal{SH}$ is 2ExpTime-hard. In this way, we not only show that our algorithm is worst-case optimal, but we also identify the combination of transitivity and role inclusions as a new source of complexity, and show that not only inverse roles make query answering harder than standard reasoning,

The algorithm for CQ answering in $\mathcal{SH}$ presented in the first part of this chapter is an extension of a similar algorithm for $\mathcal{ALCH}$ presented in [OŠE08b]. It is worst case optimal (for both $\mathcal{SH}$ and $\mathcal{ALCH}$) and improves the upper bound resulting from all previous algorithms for CQ answering in logics containing $\mathcal{ALCH}$, which either do not have a double exponential upper bound, or need double exponential time already for fragments like $\mathcal{ALCH}$ (see Section 6.6 for more details). Additionally, it has the following features:

- Different from other algorithms, it handles CQs with answer variables in a direct manner, rather than reducing such queries to ground (Boolean) CQs. Unlike we did in Chapter 6, it does not reduce the problem to many tests over trees, but handles the full query over forests directly.
- The algorithm provides a modular *knowledge compilation* of the DL knowledge base, which allows for the reuse of intermediate results. This is because it compiles first a knowledge base into a set of knots, constructs then query answering tables from this set and the input query, and finally collects the query answers from the compiled knowledge and tables using the ABox. The result of the first step may be reused for follow-up queries, i.e., only the query answering tables need to be constructed and the query answers collected. For queries

if	$C_1 \sqsubseteq C_2 \in \mathcal{T}$	then	$\sim C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$C \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$\sim C \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C_1, C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$C_1 \sqcap C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C_1, C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$\exists p.C \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$\forall p.C \in Cl_{\mathsf{C}}(\mathcal{K})$	then	$C \in Cl_{\mathsf{C}}(\mathcal{K})$
if	$\forall p.C \in Cl_{\mathsf{C}}(\mathcal{K})$ and $\mathsf{trans}(p') \in \mathcal{R}$	then	$\forall p'.C \in Cl_{\mathsf{C}}(\mathcal{K})$

Table 7.1: Concept closure $Cl_{\mathsf{C}}(\mathcal{K})$ of an $\mathcal{SH}$ KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$

of small size (bounded by a constant), the table construction is feasible in polynomial time in the size of the knot set, and collecting the query answers is feasible in coNP (viewed as a decision problem); for a fixed ABox, the latter is feasible in polynomial time. This is particularly useful for evaluating many such queries over a rather static knowledge base.

- Similarly, for a fixed query and a DL knowledge base where the TBox and the RBox are fixed but the ABox may change, i.e., in the *data complexity* setting, a non-deterministic version of our algorithm runs in polynomial time, which is also worst-case optimal.

- Finally, the compiled knowledge can be expressed as a disjunctive Datalog program (alternatively, a Datalog program with unstratified negation), which is evaluated over an enhanced ABox. The program can be designed to evaluate also non-ground queries, i.e, with answer variables directly. A Datalog encoding may make the algorithm more amenable for efficient implementation than some of the previous automata- or tableaux-based approaches, given that efficient engines for disjunctive/unstratified Datalog are available.

The rest of this chapter is organized as follows. In the next section we describe the canonical models of a knowledge base, which are forest models that are sufficient for query answering. In the follow-up Section 7.2, we discuss how forest-shaped models of a knowledge base can be represented using knots. In Section 7.3, we present our algorithm for answering CQs using knots. In Section 7.4 we address complexity issues, including both the upper and lower bounds, and describe an encoding into Datalog. In the final Section 7.5, we discuss our results, related work, and possible extensions of the approach.

7.1 Canonical Models for $\mathcal{SH}$

Throughout the following sections, $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ denotes a fixed, given $\mathcal{SH}$ KB. We assume without loss of generality that $\mathcal{A}$ is extensionally reduced (that is, only concept names occur in the concept membership assertions) and that all concepts and roles that occur in $\mathcal{A}$ occur also in $\mathcal{T}$ or $\mathcal{R}$ (see Definition 2.3.3). This assumption—which makes the presentation simpler—is natural in our setting, as we aim at a modular algorithm that handles the extensional and intensional components in separate stages.

As usual, we start by characterizing a suitable canonical model property for $\mathcal{SH}$ KBs, and adapting to $\mathcal{SH}$ the notions of closure, types, and ABox completions.

7.1.1 Syntactic Closure and Types

The definition of the syntactic closure of $\mathcal{K}$ is similar to Definition 6.1; it includes all concepts occurring in $\mathcal{T}$ (and hence also all concepts in $\mathcal{A}$), and it is closed under subconcepts (in NNF).

if	$C_1 \sqsubseteq C_2 \in \mathcal{T}$,	then	$\sim C_1 \sqcup C_2 \in \mathbf{t}$,		
if	$C \in Cl_{\mathsf{C}}(\mathcal{K})$,	then	$C \in \mathbf{t}$	iff	$\sim C \notin \mathbf{t}$,
if	$C_1 \sqcap C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$,	then	$C_1 \sqcap C_2 \in \mathbf{t}$	iff	$\{C_1, C_2\} \subseteq \mathbf{t}$,
if	$C_1 \sqcup C_2 \in Cl_{\mathsf{C}}(\mathcal{K})$,	then	$C_1 \sqcup C_2 \in \mathbf{t}$	iff	$\{C_1, C_2\} \cap \mathbf{t} \neq \emptyset$.

Table 7.2: Concept type $\mathbf{t} \subseteq Cl_{\mathsf{C}}(\mathcal{K})$ for an $\mathcal{SH}$ KB $\mathcal{K}$

Definition 7.1.1 (Concept closure) *The* concept closure $Cl_{\mathsf{C}}(\mathcal{K})$ *of* $\mathcal{K}$ *is the smallest set of* $\mathcal{SH}$ *concepts closed under the rules in Table 7.1.*

Most of the entries in Table 7.1 are self-explanatory, except the last one that is designated to deal with combinations of transitive roles and role hierarchies.

Concept types for $\mathcal{SH}$ are defined as for $\mathcal{ALCHI}$. Role types are also similar, but now they are even simpler because there are no inverses and only the RIAs need to be taken into account.

Definition 7.1.2 (Concept and role types) *A* concept type *of* $\mathcal{K}$ *is a set* $\mathbf{t} \subseteq Cl_{\mathsf{C}}(\mathcal{K})$ *of concepts that satisfies the rules in Table 7.2.*

A role-type *of* $\mathcal{K}$ *is a set* $\rho \subseteq \mathsf{N_R}(\mathcal{K})$ *such that, for each* $P_1 \sqsubseteq P_2 \in \mathcal{T}$, $P_1 \in \mathbf{r}$ *implies* $P_2 \in \mathbf{r}$.

The set of all concept types of $\mathcal{K}$ *is denoted by* $\mathsf{typesC}(\mathcal{K})$, *and the set of all role types by* $\mathsf{typesR}(\mathcal{K})$.

7.1.2 Canonical Models

So far, we have considered canonical models that are tree- or forest-shaped, but they are too restrictive for the case of $\mathcal{SH}$, since the transitivity axioms impose relations between domain elements that are not directly connected but lie on a longer path. A convenient way to characterize models for $\mathcal{SH}$ is to consider models of an $\mathcal{ALCH}$ KB $\mathcal{K}'$, such that there is a one-to-one correspondence between the models of $\mathcal{K}$ and the models of $\mathcal{K}'$. More specifically, we consider interpretations $\mathcal{I}$ that are models of $\mathcal{A}$, $\mathcal{T}$ and the RIAs in $\mathcal{R}$, but not necessarily of the transitivity axioms. By requiring $\mathcal{I}$ to satisfy some additional CIAs, we ensure that it becomes a model of $\mathcal{K}$ when some 'implied arcs' are added.

We use some terminology introduced in [GHLS08], and define *forest bases* as canonical interpretations for $\mathcal{ALCH}$ (i.e., one-way canonical interpretations in the sense of Definition 6.1.3): their domain is a forest with bounded branching and with roots $\mathsf{N_I}(\mathcal{K})$, and each non-root node can only be connected to its children. A canonical model is then defined as the *closure* of a forest base that satisfies the above restrictions, which we call a *model base*.

Definition 7.1.3 (forest base, closure, canonical interpretation) *An interpretation* $\mathcal{I}$ *is called a* forest base (for $\mathcal{K}$) *if:*

1. $\Delta^{\mathcal{I}}$ *is a forest with branching degree bounded by* $|Cl_{\mathsf{C}}(\mathcal{K})|$,
2. $\mathsf{roots}(\Delta^{\mathcal{I}}) = \{a^{\mathcal{I}} \mid a \in \mathsf{N_I}(\mathcal{K})\}$,
3. $\mathcal{I}$ *is connected, that is, for each pair* $\{w, w'\} \subseteq \Delta^{\mathcal{I}}$ *with* $w' \in \mathsf{succ}(w)$ *there is some* $p \in \mathsf{N_R}(\mathcal{K})$ *such that* $(w, w') \in p^{\mathcal{I}}$, *and*
4. $w' \in \mathsf{succ}(w)$ *for every* $w, w' \in \Delta^{\mathcal{I}}$ *such that* $\{w, w'\} \not\subseteq \mathsf{roots}(\Delta^{\mathcal{I}})$ *and* $(w, w') \in p^{\mathcal{I}}$ *for some role name* p.

The closure $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ *of a forest base* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is the interpretation that is identical to* $\mathcal{I}$ *but has, for each* $p \in \mathsf{N_R}$

$$p^{\mathcal{J}} = p^{\mathcal{I}} \cup \bigcup_{p' \sqsubseteq^{\mathcal{R}} p, \mathsf{trans}(p') \in \mathcal{R}} \left((p')^{\mathcal{I}}\right)^{+}$$

We call $\mathcal{I}$ *a* model base for $\mathcal{K}$ *if*

(a) $\mathcal{I} \models \mathcal{T}$ *and* $\mathcal{I} \models \mathcal{A}$,

(b) $\mathcal{I}$ *satisfies each RIA in* $\mathcal{R}$,

(c) $\mathcal{I} \models \{\forall p.C \sqsubseteq \forall p'.(\forall p'.C) \mid \forall p.C \in Cl_{\mathsf{C}}(\mathcal{K})$ *with* $\mathsf{trans}(p') \in \mathcal{R}$ *and* $p' \sqsubseteq^{\mathcal{R}} p\}$.

If $\mathcal{J}$ *is the closure of some model base* $\mathcal{I}$ *for* $\mathcal{K}$*, then we call* $\mathcal{J}$ *a canonical interpretation for* $\mathcal{K}$.

Model bases of $\mathcal{K}$ satisfy all the concept inclusion axioms, ABox assertions and TBox assertions of $\mathcal{K}$ (conditions (a-b)), but they are not necessarily models of $\mathcal{K}$ since the transitivity requirements may be violated. To deal with this, we require (c) which emulates the effect of transitive roles on a model. The closure, which is obtained by closing a model base under transitivity and role inclusions, leads us to a model of $\mathcal{K}$.

For simplicity, in what follows we consider only CQs. Recall that we write $\mathcal{K} \models q(\vec{a})$ if $\vec{a}$ is an answer for q in $\mathcal{K}$, i.e., if there is a match π for q in $\mathcal{I}$ such that $\pi(x_i) = (a_i)^{\mathcal{I}}$ for each $1 \leq i \leq n$ (see Section 2.2.2). We can now state the following important proposition.

Proposition 7.1.4 ([GHLS08]) *The following hold:*

1. *If* $\mathcal{J}$ *is a canonical interpretation for* $\mathcal{K}$*, then* $\mathcal{J} \models \mathcal{K}$*, i.e.,* $\mathcal{J}$ *is a canonical model of* $\mathcal{K}$.
2. *Let* q *be a CQ with* n *answer variables, and let* $\vec{a}$ *be an* n*-ary tuple of individuals. If* $\mathcal{K} \not\models q(\vec{a})$ *then there exists some canonical model* $\mathcal{J}$ *for* $\mathcal{K}$ *such that* $\mathcal{J} \not\models q(\vec{a})$.

Prematches

The proposition above implies that we can safely concentrate on closures of model bases for answering CQs. In fact, we can even consider forest bases instead of their closures when looking for query matches, as we do in this chapter, provided that the notion of match is relaxed accordingly.

We introduce the notion of a *prematch* for a query, which is 'almost a match' on a forest base, and becomes a regular match in the closure. The idea is that to satisfy an atom $p(v, v')$ we may match v and v' to nodes that are not directly related by p but they will be in the closure because there is a path between them where each pair is in the extension of some transitive subrole of p.

Definition 7.1.5 (p-follower, prematches) *Given a forest base* $\mathcal{I}$ *and* $w, w' \in \Delta^{\mathcal{I}}$*, we call* w' *a* p*-follower of* w *(in* $\mathcal{I}$*), if there is a sequence* $w_1, \ldots, w_n$ *in* $\Delta^{\mathcal{I}}$ *and a* $p' \sqsubseteq^{\mathcal{R}} p$ *such that* $w_1 = w$*,* $w_n = w'$*, and*

- $(w_i, w_{i+1}) \in (p')^{\mathcal{I}}$ *for each* $1 \leq i < n$*, and*
- *if* $n > 2$*, then* $\mathsf{trans}(p') \in \mathcal{R}$.

We say a CQ q *has a* pre-match *in* $\mathcal{I}$*, if there is a mapping* $\pi : \mathsf{VI}(q) \to \Delta^{\mathcal{I}}$ *such that:*

(PM1) $C(x) \in q$ *implies* $\pi(x) \in C^{\mathcal{I}}$, *and*

(PM2) $p(x, y) \in q$ *implies* $\pi(y)$ *is a* p*-follower of* $\pi(x)$ *in* $\mathcal{I}$.

It is not hard that to see that a prematch in a forest base becomes a regular match in its closure. Conversely, the existence of a regular match in the closure of some forest base $\mathcal{I}$ implies the existence of a prematch in $\mathcal{I}$. Hence, applying Proposition 7.1.4, we easily obtain:

Proposition 7.1.6 *Let q be a CQ q with answer variables $\vec{x} = x_1, \ldots, x_n$, and let $\vec{a} = a_1, \ldots, a_n$ be an n-ary tuple of individuals. Then the following are equivalent:*

1. $\mathcal{K} \models q(\vec{a})$;
2. *in each model base $\mathcal{I}$ for $\mathcal{K}$, there exists a prematch π for q with $\pi(x_i) = (a_i)^{\mathcal{I}}$ for each $1 \leq i \leq n$.*

By the above proposition, given $\mathcal{K}$, to decide whether a tuple $\vec{a}$ is in the answer of a CQ it suffices to see whether in every model base there is a prematch that gives $\vec{a}$ as an answer. In the rest of this chapter, we present an algorithm for the latter problem.

7.1.3 ABox Completions

Similarly as we did for $\mathcal{ALCHI}$, we define ABox completions that characterize the 'graph part' of canonical models.

Definition 7.1.7 (ABox Completion) *A* completion *for $\mathcal{K}$ is a function θ that maps each $a \in \mathsf{N_I}(\mathcal{K})$ to a concept type $\theta(a) \in \mathsf{typesC}(\mathcal{K})$ and each pair $(a, b) \in \mathsf{N_I}(\mathcal{K}) \times \mathsf{N_I}(\mathcal{K})$ to a role type $\theta(a, b) \in \mathsf{typesR}(\mathcal{K})$ such that:*

1. *for each $C(a) \in \mathcal{A}$, $C \in \theta(a)$,*
2. *for each $p(a, b) \in \mathcal{A}$, $p \in \theta(a, b)$,*
3. *for each $a, b \in \mathsf{N_I}(\mathcal{K})$, if $\forall p.C \in \theta(a)$ and $p \in \theta(a, b)$, then $C \in \theta(b)$,*
4. *for each $a, b \in \mathsf{N_I}(\mathcal{K})$, if $\forall p.C \in \theta(a)$, and $p' \in \theta(a, b)$ for some $p' \sqsubseteq^{\mathcal{R}} p$ with $\mathsf{trans}(p') \in \mathcal{R}$, then $\forall p'.C \in \theta(b)$, and*
5. *if $p \in \theta(a, b)$, $p \in \theta(b, c)$ and $\mathsf{trans}(p) \in \mathcal{R}$, then $p \in \theta(a, c)$.*

The set of all ABox completions for $\mathcal{K}$ is denoted by $\mathsf{comp}(\mathcal{K})$.

7.2 Reasoning in $\mathcal{SH}$ Using Knots

Before presenting our algorithm for query answering, we show how the models of an $\mathcal{SH}$ KB $\mathcal{K}$ can be represented using *knots*. As discussed in Section 6.3.1, knots are small tree-shaped structures that represent patters for subtrees that may occur in the canonical models of $\mathcal{K}$. By imposing suitable *coherence* conditions one can ensure that the knots in a given set can be assembled into interpretations. Every (possibly infinite) forest base for $\mathcal{K}$ can be decomposed into a set of knots and, since only finitely many knots exist for $\mathcal{K}$, this set is always finite.

7.2.1 Representing forest bases for $\mathcal{SH}$ with Knots

Unlike we did in the previous chapter, we do not reduce query answering to tree-shaped interpretations before deploying knots. Instead we adapt the knot technique, so that full forest bases are represented by sets of knots combined with ABox completions. We then define an algorithm answering queries over such a representation.

To represent forest bases, we use knots as defined for $\mathcal{ALCHI}$, in Definition 6.3.1. We recall that a knot for $\mathcal{K}$ is a pair $(\mathbf{t}, \mathbf{S})$, its root is the concept type $\mathbf{t}$, and it has a set of *children* $\mathbf{S}$, $|\mathbf{S}| \leq |Cl_{\mathsf{C}}(\mathcal{K})|$. Each child is a pair $(\mathbf{r}, \mathbf{t}')$ of a role type $\mathbf{r}$ and a concept type $\mathbf{t}'$. We adapt to $\mathcal{SH}$ the *consistency conditions* that ensure that a knot locally complies with the constraints of $\mathcal{K}$, and the global *coherence conditions* that guarantee that consistent knots can be assembled into trees, as follows.

Definition 7.2.1 (Knot consistency, coherency of knot sets, possible successor knot) *A knot* $(\mathbf{t}, \mathbf{S})$ *is* $\mathcal{K}$-consistent *if:*

1. *if* $\exists p.C \in \mathbf{t}$*, then* $p \in \mathbf{r}$ *and* $C \in \mathbf{t}'$ *for some* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$*;*
2. *if* $\forall p.C \in \mathbf{t}$*, then* $C \in \mathbf{t}'$ *for all* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$ *with* $p \in \mathbf{r}$*; and*
3. *if* $\forall p.C \in \mathbf{t}$*, then for each* $p' \sqsubseteq^{\mathcal{R}} p$ *with* $\mathsf{trans}(p') \in \mathcal{R}$ *and each* $(\mathbf{r}, \mathbf{t}') \in S$ *with* $p' \in \mathbf{r}$*, we have* $\forall p'.C \in \mathbf{t}'$*.*

Let $\mathbb{K}$ *be a set of knots. A knot* $(\mathbf{t}, \mathbf{S})$ *is* good in $\mathbb{K}$*, if for each* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$ *there is a* possible successor $(\mathbf{t}', \mathbf{S}') \in \mathbb{K}$*.* $\mathbb{K}$ *is* $\mathcal{K}$*-coherent if each knot it contains is* $\mathcal{K}$*-consistent and good in* $\mathbb{K}$*.*

The coherence conditions are very similar to Definition 6.3.4, except that we do not require the presence of an initial knot containing the distinguished concept A_0. Instead of this, and to represent forest-shaped model bases rather than simple trees, we define additional *compatibility* conditions.

A set of knots $\mathbb{K}$ is *compatible* with a set of types $\mathbf{T}$, if by appending knots in $\mathbb{K}$ we can construct a tree with root $\mathbf{t}$ for each type $\mathbf{t}$ in $\mathbf{T}$. Intuitively, this allows us to build a forest base from a completion and a set of knots that is compatible with the types that occur in it. (We note that if $\mathbb{K}$ is coherent and $\mathbf{T}$-compatibility, and A_0 is in some type of $\mathbf{T}$, then it is coherent in the sense of Definition 6.3.4.)

Definition 7.2.2 (Compatibility) *Let* $\mathbf{T} \subseteq \mathsf{types}\mathsf{C}(\mathcal{K})$*. A* $\mathcal{K}$*-coherent knot set* $\mathbb{K}$ *is* $\mathbf{T}$-compatible*, if for each* $\mathbf{t} \in \mathbf{T}$ *there exists some knot* $(\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}$ *with* $\mathbf{t}_0 = \mathbf{t}$*.*

As $\mathcal{K}$-coherence and $\mathbf{T}$-compatibility are preserved under unions of coherent knot sets, there exists a $\mathbf{T}$-*complete* knot set which contains all knot sets that are $\mathcal{K}$-coherent and $\mathbf{T}$-compatible. From this set we can construct *all* model bases whose roots satisfy types from $\mathbf{T}$.

Definition 7.2.3 (Completeness) *For a knot set* $\mathbb{K}$ *and type* $\mathbf{t}$*, let* $\mathbb{K}|\mathbf{t}$ *denote the smallest subset of* $\mathbb{K}$ *such that*

(a) $(\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}|\mathbf{t}$ *for each* $(\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}$ *with* $\mathbf{t}_0 = \mathbf{t}$*, and*

(b) if $(\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}|\mathbf{t}$*,* $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$ *and* $(\mathbf{t}', \mathbf{S}') \in \mathbb{K}$*, then* $(\mathbf{t}', \mathbf{S}') \in \mathbb{K}|\mathbf{t}$*, i.e.,* $\mathbb{K}|\mathbf{t}$ *is closed under the possible successors in* $\mathbb{K}$*.*

Let $\mathbb{K}$ *be a* $\mathcal{K}$*-coherent set of knots and* $\mathbf{T}$ *a set of types for* $\mathcal{K}$*. We say* $\mathbb{K}$ *is* $\mathbf{T}$-complete *if for each* $\mathbf{t} \in \mathbf{T}$ *and each* $\mathcal{K}$*-coherent set* $\mathbb{K}'$ *we have* $\mathbb{K}'|\mathbf{t} \subseteq \mathbb{K}$*.*

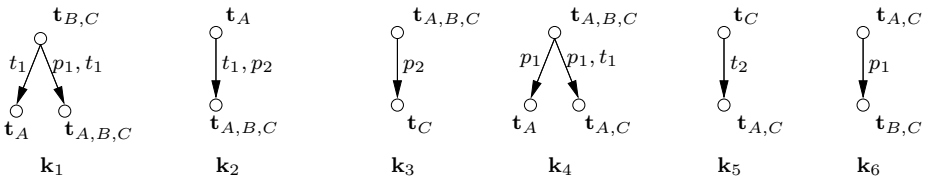

Figure 7.1: Example set of knots

Intuitively, $\mathbb{K}|\mathbf{t}$ is the restriction of $\mathbb{K}$ to knots that have root $\mathbf{t}$, or are reachable from the former via the possible successor relation. A $\mathbf{T}$-complete set that contains $\mathbb{K}|\mathbf{t}$ for all knots in $\mathbf{T}$ is sufficient to build all the trees starting with a type $\mathbf{t} \in \mathbf{T}$.

Example 7.2.4 *A $\mathcal{K}$-coherent set of knots (for some KB $\mathcal{K}$) is depicted in Figure 7.1. Similarly as in Chapter 6, we represent a knot $\mathbf{k} = (\mathbf{t}, \mathbf{S})$ as a tree where the root is labeled $\mathbf{t}$, and that has, for each $(\mathbf{r}, \mathbf{t}') \in S$, an arc labeled $\mathbf{r}$ to a child labeled $\mathbf{t}'$. Five different types occur in these knots. We omit their full description, and simply assume that the set of concept names that occur in each $\mathbf{t}_X$ is exactly $\{X\} \subseteq \{A, B, C\}$. In the arc labels, in contrast, we write the full role types $\{t_1\}$, $\{t_1, p_1\}$, ..., omitting braces. Observe that $\mathbf{k}_2$ is a possible successor of $(\{t_1\}, \mathbf{t}_A)$ in $\mathbf{k}_1$, while $(\{p_1, t_1\}, \mathbf{t}_{A,B,C})$ has the possible successors $\mathbf{k}_3$ and $\mathbf{k}_4$.*

To build model bases for $\mathcal{K}$, we need some completion and some set of knots that is compatible with the concept types that occur in the completion. Given this, we can construct forest bases as follows.

Definition 7.2.5 (Model base induced by a completion and a knot set) *Let θ be an ABox completion for $\mathcal{K}$, and let $\mathbf{T} \subseteq \mathsf{types}\mathsf{C}(\mathcal{K})$ be a set of types such that $\{\theta(a) \mid a \in \mathsf{N_I}(\mathcal{K})\} \subseteq \mathbf{T}$. Furthermore, let $\mathbb{K}$ be a $\mathbf{T}$-compatible knot set. A forest base $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is* induced by θ *and $\mathbb{K}$ if:*

(a) For each $a, b \in \mathsf{N_I}(\mathcal{A})$ and each role p, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in p^{\mathcal{I}}$ iff $p \in \theta(a, b)$.

(b) $\mathcal{I}$ is induced *by a function $\xi : \Delta^{\mathcal{I}} \to \mathbb{K}$ such that, for each element $w \in \Delta^{\mathcal{I}}$, the knot $\xi(w) = (\mathbf{t}, \mathbf{S})$ satisfies:*

- *for each concept name A, $w \in A^{\mathcal{I}}$ iff $A \in \mathbf{t}$, and*
- *there exists a bijection $f : \mathbf{S} \to \mathsf{succ}(w)$ such that for each $s = (\mathbf{r}, \mathbf{t}')$ in $\mathbf{S}$ and each role p, we have $(w, f(s)) \in p^{\mathcal{I}}$ iff $p \in \mathbf{r}$.*

The set of all such $\mathcal{I}$ is denoted by $\mathfrak{F}(\theta, \mathbb{K})$.

An easy consequence of the above definitions is that each forest base in $\mathfrak{F}(\theta, \mathbb{K})$ is a model base for $\mathcal{K}$.

Lemma 7.2.6 *If θ is an ABox completion for $\mathcal{K}$ and $\mathbb{K}$ is $\mathbf{T}$-compatible for some $\mathbf{T} \supseteq \{\theta(a) \mid a \in \mathsf{N_I}(\mathcal{K})\}$, then each $\mathcal{I} \in \mathfrak{F}(\theta, \mathbb{K})$ is a model base for $\mathcal{K}$.*

A $\mathbf{T}$-complete knot set is sufficient to induce a model base $\mathcal{I}$ for $\mathcal{K}$ if the types that occur in the corresponding completion are contained in $\mathbf{T}$, i.e., if $\{C \mid a^{\mathcal{I}} \in C^{\mathcal{I}}\} \in \mathbf{T}$ for each $a \in \mathsf{N_I}(\mathcal{K})$. Hence, if a knot set is complete for all concept types that occur in some ABox completion of $\mathcal{K}$, we can use it to induce all the model bases for $\mathcal{K}$.

Definition 7.2.7 (Model bases induced by a knot set, ABox types) *For a set of types* $\mathbf{T}$ *and a* $\mathbf{T}$*-complete knot set* $\mathbb{K}$*, we denote by* $\mathfrak{F}_{\mathbb{K}}(\mathcal{K})$ *the set of all model bases induced by the knot set* $\mathbb{K}$ *and some ABox completion* θ *of* $\mathcal{K}$ *such that* $\theta(a) \in \mathbf{T}$ *for each* $a \in \mathsf{N_I}(\mathcal{K})$*.*

We denote by $\mathsf{ABoxTypes}(\mathcal{K})$ *the set of all concept types that occur in some ABox completion of* $\mathcal{K}$*, that is*

$$\mathsf{ABoxTypes}(\mathcal{K}) = \bigcup_{\theta \in \mathsf{comp}(\mathcal{K})} \{\theta(a) \mid a \in \mathsf{N_I}(\mathcal{K})\}.$$

If a knot set $\mathbb{K}$ is $\mathsf{ABoxTypes}(\mathcal{K})$-complete, then every model base can be induced from it. Hence, to answer each given CQ, it is enough to look for its prematches in the forest bases in $\mathfrak{F}_{\mathbb{K}}(\mathcal{K})$.

Proposition 7.2.8 *Let* $\mathbb{K}$ *be an* $\mathsf{ABoxTypes}(\mathcal{K})$*-complete knot set, let* q *be a CQ with answer variables* $x_1, \ldots, x_n$*, and let* $\vec{a} = a_1, \ldots, a_n$ *be an* n*-ary tuple of individuals. Then the following are equivalent:*

1. $\mathcal{K} \models q(\vec{a})$*;*
2. *in each model base* $\mathcal{I} \in \mathfrak{F}_{\mathbb{K}}(\mathcal{K})$ *there exists a prematch* π *for* q *with* $\pi(x_i) = {a_i}^{\mathcal{I}}$ *for each* $1 \leq i \leq n$*.*

Proof. The $(1 \rightarrow 2)$ direction follows directly from Proposition 7.1.6 and Lemma 7.2.6.

For the $(2 \rightarrow 1)$ direction, assume $\vec{a} \notin \mathsf{ans}(q, \mathcal{K})$. By Proposition 7.1.6, there exists a model base $\mathcal{I}$ for $\mathcal{K}$ that admits no prematch π for q with $\pi(x_i) = {a_i}^{\mathcal{I}}$ for each $1 \leq i \leq n$. We just need to argue that $\mathcal{I} \in \mathfrak{F}_K(\mathcal{A})$. To this end, we decompose $\mathcal{I}$ into a completion of $\mathcal{A}$ and a set of knots. First, the following θ is a completion for $\mathcal{K}$:

- for each $a \in \mathsf{N_I}(K)$, $\theta(a) = \{C \in \mathit{Cl}_{\mathsf{C}}(\mathcal{K}) \mid a^{\mathcal{I}} \in C^{\mathcal{I}}\}$, and
- for each pair $a, b \in \mathsf{N_I}(K)$, $\theta(a, b) = \{p \in \mathsf{N_R} \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in p^{\mathcal{I}}\}$.

Then we 'decompose' the tree parts of $\mathcal{I}$ into knots. Let ξ be a mapping that assigns to each $w \in \Delta^{\mathcal{I}}$ a knot $\xi(w) = (\mathbf{t}_0, \mathbf{S})$, where

- $\mathbf{t}_0 = \{C \in \mathit{Cl}_{\mathsf{C}}(\mathcal{K}) \mid w \in C^{\mathcal{I}}\}$, and
- $\mathbf{S} = \bigcup_{w \cdot i \in \mathsf{succ}(w)} \{(\mathbf{r}_i, \mathbf{t}'_i)\}$, where $\mathbf{r}_i = \{p \mid (w, w \cdot i) \in p^{\mathcal{I}}\}$ and $\mathbf{t}'_i = \{C \in \mathit{Cl}_{\mathsf{C}}(\mathcal{K}) \mid w \cdot i \in C^{\mathcal{I}}\}$.

By construction, the knot set $\mathbb{K}' = \{\xi(w) \mid w \in \Delta^{\mathcal{I}}\}$ is $\mathcal{K}$-coherent. Furthermore, $\mathbb{K}'$ is $\{\theta(a) \mid a \in \mathsf{N_I}(\mathcal{K})\}$-compatible, so $\mathbb{K}' \subseteq \mathbb{K}$ because $\mathbb{K}$ is $\mathsf{ABoxTypes}(\mathcal{K})$-complete. Finally, since the mapping ξ is as required in Definition 7.2.5, it is easy to verify that $\mathcal{I} \in \mathfrak{F}_K(\mathcal{A})$. □

A simple algorithm for computing a $\mathbf{T}$-complete knot set for a given KB $\mathcal{K}$ and a set $\mathbf{T}$ of types for $\mathcal{K}$ is presented in Figure 7.2. We start by computing the set $\mathbb{K}$ of all knots for $\mathcal{K}$ that have root $\mathbf{t} \in \mathbf{T}$. In a second stage we close $\mathbb{K}$ by adding all $\mathcal{K}$-compatible possible successors knots. Finally, as in the satisfiability algorithm in Figure 6.6, we remove from $\mathbb{K}$ one by one the knots that are not good, because they have a child for which $\mathbb{K}$ does not contain a possible successor. The set $\mathbb{K}$ returned by the algorithm includes $\mathbb{K}|\mathbf{t}$ for each type $\mathbf{t} \in \mathbf{T}$, and it is $\mathcal{K}$-coherent. single exponential in $\mathcal{T}$ and $\mathcal{R}$ and polynomial in $\mathbf{T}$, and does not depend on $\mathcal{A}$. We elaborate on this in Section 7.4.

Algorithm 3: computeKnots

Input: a KB $\mathcal{K}$, a set $\mathbf{T}$ of types for $\mathcal{K}$
Output: a $\mathbf{T}$-complete knot set $\mathbb{K}$
begin
  Compute the set $\mathbb{K}$ of all knots $(\mathbf{t}_0, \mathbf{S})$ for $\mathcal{K}$ that are consistent w.r.t. $\mathcal{K}$ and such that $\mathbf{t}_0 \in \mathbf{T}$;
  repeat
    $\mathbb{K}' := \mathbb{K}$;
    $\mathbb{K} := \mathbb{K}' \cup \{(\mathbf{t}', \mathbf{S}') \mid (\mathbf{t}_0, \mathbf{S}) \in \mathbb{K}'$ and $(\mathbf{r}, \mathbf{t}') \in \mathbf{S}$ and $(\mathbf{t}', \mathbf{S}')$ is $\mathcal{K}$-compatible$\}$;
  until $\mathbb{K}' \neq \mathbb{K}$;
  repeat
    $\mathbb{K}' := \mathbb{K}$;
    $\mathbb{K} := \mathbb{K}' \setminus \{(\mathbf{t}, \mathbf{S}) \in \mathbb{K}' \mid (\mathbf{t}, \mathbf{S})$ is not good in $\mathbb{K}'\}$;
  until $\mathbb{K}' \neq \mathbb{K}$;
  return $\mathbb{K}$
end

Figure 7.2: Building a $\mathbf{T}$-complete knot set for $\mathcal{K}$.

7.3 Query Answering for $\mathcal{SH}$ by Knot Compilation

We now present our algorithm for answering conjunctive queries over $\mathcal{SH}$ knowledge bases. For the rest of this section, we assume a fixed CQ $q = \exists \vec{v}.\varphi(\vec{x}, \vec{v})$, whose answer variables are $\vec{x} = x_1, \ldots, x_n$. We assume that q is connected and extensionally reduced (i.e., only concept and role names occur in q) and that there are only variables in $\mathsf{Vl}(q)$ (see Section 2.2.3). Please keep in mind that in this chapter we handle answer variables explicitely and thus $\vec{x}$ need not be empty.

We consider the query answering problem as described in Definition 2.2.5, i.e., given a tuple of individuals $\vec{a} = a_1, \ldots, a_n$, to decide whether $\mathcal{K} \models q(\vec{a})$. We denote by $\mathsf{ans}(q, \mathcal{K})$ the set of such answers, that is, we want to decide whether $\vec{a} \in \mathsf{ans}(q, \mathcal{K})$ where

$$\mathsf{ans}(q, \mathcal{K}) = \{\vec{a} = a_1, \ldots, a_n \mid \mathcal{K} \models q(\vec{a})\}.$$

Relying on the results above, the algorithm achieves this by establishing for which such tuples $\vec{a} = a_1, \ldots, a_n$ there exists a prematch π for q with $\pi(x_i) = {a_i}^{\mathcal{I}}$ for each $1 \leq i \leq n$, in each model base $\mathcal{I} \in \mathfrak{F}_{\mathbb{K}}(\mathcal{K})$ for some $\mathsf{ABoxTypes}(\mathcal{K})$-complete knot set $\mathbb{K}$.

Our method for query answering relies on knot sets and is presented in three steps:

- We first define a suitable notion of *subqueries* and their matches. The definition of subqueries is motivated by the 'shape' that query prematches can take in forest bases, similarly to the tree-shaped queries we discussed in Chapter 6.

- We then compile an input query q and the 'intensional part' $(\mathcal{T}, \mathcal{R})$ of $\mathcal{K}$ into a *type-query table*, which, informally speaking, tells which subqueries of q can be mapped in any tree generated from knots starting from a particular root type.

- Finally, given an arbitrary ABox $\mathcal{A}$, we can answer q over a $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ by considering partial mappings of q into completions of $\mathcal{A}$ and by looking up the query remainders in the precomputed type-query table.

7.3.1 Subqueries and Rooted Matches

In the first two steps of the algorithm we focus only on the 'tree-parts', deferring the 'A-Box part' to the last stage. That is, we first consider the tree-shaped parts of model bases, which we formally called tree-shaped interpretations, and look for pre-matches for subqueries in them.

A tree-shaped interpretations is defined almost like a forest base, except for the second condition: it has only one root instead of a set of roots corresponding to the interpretations of the ABox individuals.

Definition 7.3.1 (tree-shaped interpretation, subinterpretation) *We can an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$* tree-shaped*, if it satisfies conditions 1, 3, and 4 in Definition 7.1.3 and $\mathsf{roots}(\Delta^{\mathcal{I}})$ contains exactly one element, which we denote by $\mathsf{root}(\mathcal{I})$.*

For a tree interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and a $w \in \Delta^{\mathcal{I}}$, we denote by $\mathcal{I}_{|w}$ the restriction of $\mathcal{I}$ to the domain $(\Delta^{\mathcal{I}})_w$ that contains all descendants of w in $\Delta^{\mathcal{I}}$.

In particular, we obtain a tree-shaped interpretation whenever we take a subtree T_w of $\Delta^{\mathcal{I}}$ for some $w \in \Delta^{\mathcal{I}}$ in a forest base $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and restrict the interpretation accordingly.

Next we introduce a special notion of *subqueries* adorned with some additional information, and a special notion of matches for these adorned subqueries in tree-shaped interpretations. These prematches will be the main ingredient for deciding the existence of full prematches for q in the last stage of the algorithm.

We start by defining *forward-closed* sets of variables, which intuitively are subsets of $\mathsf{Vl}(q)$ such that a prematch for q must necessarily match all the variables in the set inside the same tree-shaped interpretation. More precisely, suppose $\mathcal{I}$ is a tree-shaped interpretation and that q has a prematch π in $\mathcal{I}$. Consider a node $w \in \Delta^{\mathcal{I}}$. Let $V_w \subseteq \mathsf{Vl}(q)$ be the set of variables of q that are mapped to a node in the tree rooted at w. We can make the following observations about the set V_w:

(P1) If $v \in V_w$ and there is an atom $p(v, v')$ in q, then $v' \in V_w$, i.e., v' must be mapped into the subtree of $\mathcal{I}$ rooted at w.

(P2) If there are two atoms $p(v_1, v_2)$ and $p'(v_1', v_2)$ in q, where p and p' are simple w.r.t. $\mathcal{R}$, and $\{v_1, v_2\} \subseteq V_w$, then $v_1' \in V_w$, i.e., v_1' must also be mapped into the same subtree, and in particular at the same element as v_1.

(P3) If $v' \in V_w$ and there is some $p(v, v')$ in q with simple p and such that $v \notin V_w$, then v' must be mapped to w.

(P4) The restriction of the query graph of q to the variables in V_w is acyclic.

(P5) If $v' \in V_w$ and there is $p(v, v')$ in q where p is not simple, and $v \notin V_w$, then $\pi(v') = w$ or $\pi(v')$ is a p-follower of w.

A forward-closed set of variables is a set that induces a maximal connected subquery of $q|_{V_w}$, for some such a V_w. We adorn this subquery with a set of transitive subroles of each p as in (P5). This is reflected in the following definitions:

Definition 7.3.2 (f-subqueries) *A set of variables $V \subseteq \mathsf{Vl}(q)$ is* forward-closed*, if it satisfies the following conditions:*

(a) If $p(v, v') \in \mathsf{Atoms}(q)$ and $v \in V$, then $v' \in V$.

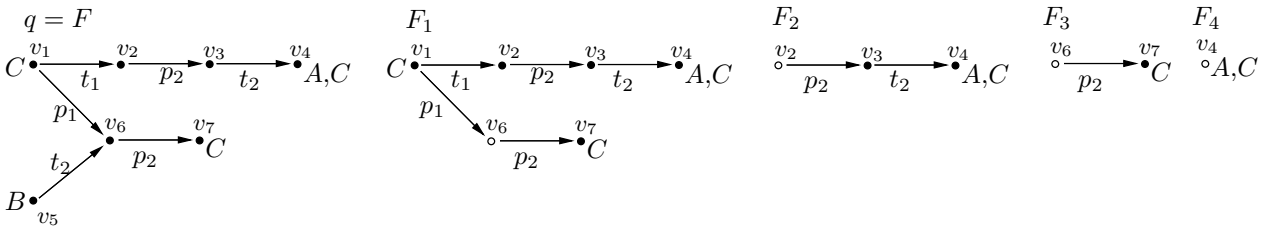

Figure 7.3: An example query and some of its f-subqueries

(b) if $p(v_1, v_2)$ and $p'(v_1', v_2)$ are two atoms in $\mathsf{Atoms}(q)$ where p, p' are simple, and $v_1 \in V$, then we also have $v_1' \in V$.

(c) The restriction of the query graph of q to variables in V is connected and acyclic, i.e., V induces a connected acyclic subquery of q, denoted $q|_V$.

The subquery $q|_V$ induced by V is called the forward-closed subquery *of q w.r.t. V.*

Given $V \subseteq \mathsf{Vl}(q)$ and $v' \in V$, let $\mathsf{back}(V, v') = \{p \mid p(v, v') \in \mathsf{Atoms}(q) \wedge v \notin V\}$. We call $v' \in V$ open *in V if $\mathsf{back}(V, v') \neq \emptyset$. If in addition $\mathsf{back}(V, v')$ does not contain a simple role, then v' is* free *in V.*

*An $\mathcal{R}$-*adornment *for a forward-closed subquery $q|_V$ is a mapping Σ that assigns to every free variable v' in V some set*

$$\Sigma(v') \subseteq \{p' \in \mathsf{N_R}(\mathcal{K}) \mid p' \sqsubseteq^{\mathcal{R}} p,\ p \in \mathsf{back}(V, v'),\ \mathsf{trans}(p') \in \mathcal{R}\}$$

such that for each $p \in \mathsf{back}(V, v')$, there is some $p' \in \Sigma(v')$ with $p' \sqsubseteq^{\mathcal{R}} p$.

A pair (V, Σ) of a forward-closed set of variables $V \subseteq \mathsf{Vl}(q)$ and an $\mathcal{R}$-adornment for $q|_V$ is called a base for the forward-closed subquery of q w.r.t. $\mathcal{R}$ and V. In slight abuse of terminology, we call (V, Σ) an f-subquery of q (w.r.t. $\mathcal{R}$).

The set of f-subqueries of q w.r.t. $\mathcal{R}$ is denoted by $\mathbb{F}_q^{\mathcal{R}}$. Any set $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$ of f-subqueries is called a disjunctive f-subquery *of q.*

Example 7.3.3 *We assume that in our knowledge base $\mathcal{K}$ $\mathsf{trans}(t_1)$ is the only transitivity axiom, and it contains a role inclusion axiom $t_1 \sqsubseteq t_2$. For our examples, we consider the query*

$$q = \exists v_1, \ldots, v_7.\ \ C(v_1) \wedge t_1(v_1, v_2) \wedge p_2(v_2, v_3) \wedge t_2(v_3, v_4) \wedge A(v_4) \wedge C(v_4) \wedge \\ p_1(v_1, v_6) \wedge B(v_5) \wedge t_2(v_5, v_6) \wedge p_2(v_6, v_7) \wedge C(v_7)$$

whose query graph, augmented with node labels $\{A \in \mathsf{N_C} \mid A(v) \in q\}$ and edge labels $\{p \in \mathsf{N_R} \mid p(v, v') \in \mathsf{Atoms}(q)\}$, is depicted in Figure 7.3. $F = (\mathsf{Vl}(q), \emptyset)$ (i.e., the full q) is an f-subquery. In an f-subquery that contains all variables except v_5, the variable v_6 is open (the non-simple p_1 is the only role in $\mathsf{back}(\mathsf{Vl}(q) \setminus \{v_5\}, v_6)$), and therefore $t_1 \in \Sigma(v_6)$ must hold; $F_1 = (\mathsf{Vl}(q) \setminus \{v_5\}, \{(v_6, \{t_1\})\})$ is such an f-subquery. Other f-subqueries with some free variable are $F_2 = (\{v_2, v_3, v_5\}, \{(v_2, \{t_1\})\})$, $F_3 = (\{v_6, v_7\}, \{(v_6, \{t_1\})\})$ and $F_4 = (\{v_4\}, \{(v_4, \{t_1\})\})$. These f-subqueries are also depicted in Figure 7.3; only the (unadorned) query graph is shown, using empty dots for the open variables.

Now we define a notion of prematch for an f-subquery (V, Σ) in a tree-shaped interpretation $\mathcal{I}$. Intuitively, it is a prematch for $q|_V$ in $\mathcal{I}$ that satisfies some additional 'rootedness' conditions: open variables that are not free must be mapped at the root of $\mathcal{I}$, and the match of each free variable v must be reachable from the root via the roles in $\Sigma(v)$.

Definition 7.3.4 (Rooted prematches) *Given* $(V, \Sigma) \in \mathbb{F}_q^{\mathcal{R}}$ *and a tree-shaped interpretation* $\mathcal{I}$*, a* rooted prematch for (V, Σ) in $\mathcal{I}$ *is a mapping* $\pi : V \to \Delta^{\mathcal{I}}$ *such that:*

(RP1) If $v \in V$ *and* $A(v) \in \mathsf{Atoms}(q)$*, then* $\pi(v) \in A^{\mathcal{I}}$*;*

(RP2) If $\{v, v'\} \subseteq V$ *and* $p(v, v') \in \mathsf{Atoms}(q)$*, then* $\pi(v')$ *is a p-follower of* $\pi(v)$ *in* $\mathcal{I}$*;*

(RP3) If v' *is open in* V *but not free, then* $\pi(v') = \mathsf{root}(\mathcal{I})$*;*

(RP4) If v' *is a free variable in* V *and* $p \in \Sigma(v')$*, then* $\pi(v')$ *is a p-follower of* $\mathsf{root}(\mathcal{I})$*.*

We write $\mathcal{I} \models (V, \Sigma)$ *if there exists a rooted prematch* π *for* (V, Σ) *in* $\mathcal{I}$*. Furthermore, we write* $\mathcal{I} \models^d (V, \Sigma)$ *if there is such a rooted prematch* π *with* $\ell(\pi(v')) \leq d$ *for each* $v' \in V$*, where, for each* w*,* $\ell(w) = |w'|$ *for the unique* $w' \in \mathbb{N}^*$ *such that* $w = \mathsf{root}(\mathcal{I}) \cdot w'$*, i.e., the match is within depth* d *in* $\mathcal{I}$*.*

Furthermore, given a disjunctive f-subquery $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$*, we write* $\mathcal{I} \models \mathrm{D}$ *(resp.,* $\mathcal{I} \models^d \mathrm{D}$*) if for some* $(V, \Sigma) \in \mathrm{D}$ *we have* $\mathcal{I} \models (V, \Sigma)$ *(resp.,* $\mathcal{I} \models^d (V, \Sigma)$*).*

7.3.2 Subquery Entailment at Knots and Types

In the following, we provide a method to test existence of rooted matches in certain tree-shaped interpretations that are constructed out of knots in a given $\mathsf{ABoxTypes}(\mathcal{K})$-complete knot set $\mathbb{K}$, starting from a particular knot or type.

We start by formally defining these interpretations. To ease presentation, in what follows we assume a fixed, arbitrary $\mathsf{ABoxTypes}(\mathcal{K})$-complete knot set $\mathbb{K}$, and let $\mathsf{typesC}(\mathbb{K}) = \{\mathbf{t} \mid (\mathbf{t}, \mathbf{S}) \in \mathbb{K}\}$.

Definition 7.3.5 (k-trees and t-trees) *Let* $\mathbf{k} \in \mathbb{K}$ *be a knot. A tree-shaped interpretation* $\mathcal{I}$ *is a called a* $\mathbf{k}$-tree *if if is* induced *by a function* $\xi : \Delta^{\mathcal{I}} \to \mathbb{K}$ *such that* $\xi(\mathsf{root}(\mathcal{I})) = \mathbf{k}$ *and for each element* $w \in \Delta^{\mathcal{I}}$ *the knot* $\xi(w) = (\mathbf{t}, \mathbf{S})$ *satisfies:*

(a) for each concept name A*,* $w \in A^{\mathcal{I}}$ *iff* $A \in \mathbf{t}$*, and*

(b) there exists a bijection $b : \mathbf{S} \to \mathsf{succ}(w)$ *such that for each* $s = (\mathbf{r}, \mathbf{t}')$ *in* $\mathbf{S}$ *and each role* p*, we have* $(w, f(s)) \in p^{\mathcal{I}}$ *iff* $p \in \mathbf{r}$*.*

Similarly, for a type $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$*, a tree-shaped interpretation* $\mathcal{I}$ *is a called a* $\mathbf{t}$-tree*, if there exists an* inducing function $\xi : \Delta^{\mathcal{I}} \to \mathbb{K}$ *such that* $\xi(\mathsf{root}(\mathcal{I})) = (\mathbf{t}, \mathbf{S})$ *for some* $\mathbf{S}$*, and for each* $w \in \Delta^{\mathcal{I}}$ *the knot* $\xi(w)$ *satisfies (a) and (b) above.*

The set of all $\mathbf{k}$*-trees is denoted by* $\mathfrak{T}(\mathbf{k})$*, and the set of all* $\mathbf{t}$*-trees by* $\mathfrak{T}(\mathbf{t})$*.*

Note that for every knot tree $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$, at the root of $\mathcal{I}$ we have a unique bijection b in (b), and thus each node w at depth 1 (i.e., each child of the root) is uniquely identified by some child $s \in \mathbf{S}$ of $\mathbf{k} = (\mathbf{t}, \mathbf{S})$; for convenience, we will refer to w by $b_s^{\mathcal{I}}$. We will also use $\mathcal{I}_s$ to denote $\mathcal{I}_{|b_s^{\mathcal{I}}}$ (the tree interpretation whose domain is the subtree of $\mathcal{I}$ rooted at $b_s^{\mathcal{I}}$).

Definition 7.3.6 (Entailment at knots and types) *Let* $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$*,* $\mathbf{k} \in \mathbb{K}$ *and* $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$*.*

We write
- $\mathbf{k} \models \mathrm{D}$*, if* $\mathcal{I} \models \mathrm{D}$ *for each* $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$*,*
- $\mathbf{k} \models^d \mathrm{D}$*, if* $\mathcal{I} \models^d \mathrm{D}$ *for each* $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$*,*
- $\mathbf{t} \models \mathrm{D}$*, if* $\mathcal{I} \models \mathrm{D}$ *for each* $\mathcal{I} \in \mathfrak{T}(\mathbf{t})$*, and*
- $\mathbf{t} \models^d \mathrm{D}$*, if* $\mathcal{I} \models^d \mathrm{D}$ *for each* $\mathcal{I} \in \mathfrak{T}(\mathbf{t})$*.*

We will use *type-query* and *knot-query* tables to store relevant pairs of types and disjunctive f-subqueries, and pairs of knots and disjunctive f-subqueries, for which the entailment relation above holds.

Definition 7.3.7 (kq-table and tq-table) *A* knot-query table (kq-table) *(for* $\mathbb{K}$ *and* q*) is an arbitrary relation* $\mathsf{KQ} \subseteq \mathbb{K} \times 2^{\mathbb{F}_q^{\mathcal{R}}}$*. We call* KQ (d-)complete *if*

(i) $(k, \mathrm{D}) \in \mathsf{KQ}$ *implies* $\mathbf{k} \models^{(d)} \mathrm{D}$*, and*

(ii) $(k, \mathrm{D}) \in \mathsf{KQ}$ *whenever* $\mathbf{k} \models^{(d)} \mathrm{D}$ *and there is no* $\mathrm{D}' \subset \mathrm{D}$ *with* $\mathbf{k} \models^{(d)} \mathrm{D}'$*.*

Similarly, a type-query table (tq-table) *(for* $\mathbb{K}$ *and* q*) is an arbitrary relation* $\mathsf{TQ} \subseteq \mathsf{typesC}(\mathcal{K}) \times 2^{\mathbb{F}_q^{\mathcal{R}}}$*, and we call* TQ (d-)complete *if*

(i) $(\mathbf{t}, \mathrm{D}) \in \mathsf{TQ}$ *implies* $\mathbf{t} \models^{(d)} \mathrm{D}$*, and*

(ii) $(\mathbf{t}, \mathrm{D}) \in \mathsf{TQ}$ *whenever* $\mathbf{t} \models^{(d)} \mathrm{D}$ *and there is no* $\mathrm{D}' \subset \mathrm{D}$ *with* $\mathbf{t} \models^{(d)} \mathrm{D}'$*.*

In the remainder of this section, we show how to compute d-complete kq-tables and d-complete tq-tables. These tables will allow us to finally obtain a complete tq-table, which we use later to answer queries over the full knowledge base $\mathcal{K}$. The basic strategy is as follows:

(I) we show how to compute a d-complete tq-table TQ^d from a given d-complete kq-table KQ^d, and

(II) we show how to compute a d+1-complete kq-table KQ^{d+1} from a given d-complete tq-table TQ^d.

Provided that we have a 0-complete tq-table TQ^0, we can compute d-complete tq-tables and kq-tables for any $d \in \mathbb{N}$ by iteratively applying the two steps above. It will be easy to see that in this way we can obtain a compete tq-table.

Furthermore, constructing an initial 0-complete tq-table TQ^0 is easy. It contains pairs of a type $\mathbf{t}$ and an f-subquery F with only one variable v, such that $\mathbf{t}$ contains all concepts required to match v. That is, to build TQ^0 we simply take each pair $(\mathbf{t}, F)$ where $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$ and $F = (\{v\}, \Sigma) \in \mathbb{F}_q^{\mathcal{R}}$, such that $v \in \mathsf{VI}(q)$ has no successors in q (i.e., there are no atoms $p(v, v')$ in q), $A \in \mathbf{t}$ for each $A(v)$ in q, and if v is free in V, then Σ assigns to v the maximal set of roles that complies with Definition 7.3.2. This procedure is shown in Figure 7.4.

Example 7.3.8 *The following table* TQ^0 *is an example of a 0-complete tq-table for the query* q *given in Example 7.3.3 and the set of knots given in Example 7.2.4, where as above* $F_4 = (\{v_4\}, \{(v_4, \{T\})\})$ *and* $F_5 = (\{v_7\}, \emptyset)$*. Note that in this case, it is enough to consider singleton disjunctive f-subqueries.*

Type	*disjunctive f-subquery*
$\mathbf{t}_C$	$\{F_5\}$
$\mathbf{t}_{A,C}$	$\{F_5\}$
$\mathbf{t}_{A,C}$	$\{F_4\}$
$\mathbf{t}_{B,C}$	$\{F_5\}$
$\mathbf{t}_{A,B,C}$	$\{F_5\}$
$\mathbf{t}_{A,B,C}$	$\{F_4\}$

The central notion for the computation is that of minimal hitting sets.

Definition 7.3.9 (Minimal hitting sets and k/t-hits) *Let* $\mathbf{k} \in \mathbb{K}$*. Then a set* $h \subseteq \mathbb{F}_q^{\mathcal{R}}$ *is called a* $\mathbf{k}$-hit *of a kq-table* KQ*, if* h *is a* $\subseteq$*-minimal set such that* $h \cap \mathrm{D} \neq \emptyset$ *for each* $(\mathbf{k}', \mathrm{D}) \in \mathsf{KQ}$ *with* $\mathbf{k}' = \mathbf{k}$*.*

Analogously, let $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$*. Then* $h \subseteq \mathbb{F}_q^{\mathcal{R}}$ *is called a* $\mathbf{t}$-hit *of a tq-table* TQ*, if* h *is a* $\subseteq$*-minimal set such that* $h \cap \mathrm{D} \neq \emptyset$ *for each* $(\mathbf{t}', \mathrm{D}) \in \mathsf{TQ}$ *with* $\mathbf{t}' = \mathbf{t}$*.*

Algorithm 4: TQ_Zero

Output: a 0-complete tq-table TQ^0
begin
  $R := \emptyset$;
  forall $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$ **do**
    forall $v \in \mathsf{Vl}(q)$ **do**
      if $\{p \mid p(v, v') \in \mathsf{Atoms}(q)\} = \emptyset$ **and** $\{A \mid A(v) \in \mathsf{Atoms}(q)\} \subseteq \mathbf{t}$ **then**
        if v *is free in* $\{v\}$ **then**
          $\Sigma := \{p' \in \mathsf{N_R}(\mathcal{K}) \mid p' \sqsubseteq^{\mathcal{R}} p,\ p \in \mathsf{back}(\{v\}, v),\ \mathsf{trans}(p') \in \mathcal{R}\}$;
        else
          $\Sigma := \emptyset$;
        $R := R \cup (\{v\}, \{v \mapsto \Sigma\})$;
  return R
end

Figure 7.4: Constructing a 0-complete tq-table (for the knot set $\mathbb{K}$ and the query q).

The following property of minimal hitting sets is important.

Lemma 7.3.10 *The following hold:*

1. *If h is a $\mathbf{k}$-hit of a d-complete kq-table KQ, then there exists some $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$ such that $\mathcal{I} \models^d (V, \Sigma)$ iff $(V, \Sigma) \in h$.*
2. *If h is a $\mathbf{t}$-hit of a d-complete tq-table TQ, then there exists some $\mathcal{I} \in \mathfrak{T}(\mathbf{t})$ such that $\mathcal{I} \models^d (V, \Sigma)$ iff $(V, \Sigma) \in h$.*

Proof. We only consider the case of $\mathbf{k}$-hits (item 1), the proof for $\mathbf{t}$-hits (item 2) is analogous.

Consider a $\mathbf{k}$-hit h as above and consider $\mathrm{D} = \mathbb{F}_q^{\mathcal{R}} \setminus h$. Since $\mathrm{D} \cap h = \emptyset$, we have $(\mathbf{k}, \mathrm{D}) \notin \mathsf{KQ}$. Hence, $\mathbf{k} \not\models^d \mathrm{D}$, i.e., there is some $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$ such that $\mathcal{I} \not\models^d F$ for each $F \in \mathrm{D}$. To prove the claim it suffices to show that $\mathcal{I} \models^d F$ for all $F \in h$.

Consider an arbitrary $F \in h$. As easily seen, by minimality of h there exists some $(\mathbf{k}, \mathrm{D}_F) \in \mathsf{KQ}$ such that $F \in \mathrm{D}_F$ and $|\mathrm{D}_F \cap h| = 1$ (if not, $h \setminus \{F\}$ would be a smaller hitting set). As $\mathrm{D}_F \setminus \{F\} \subseteq \mathrm{D}$, we have that $\mathcal{I} \not\models^d F'$ for each $F' \in \mathrm{D}_F \setminus \{F\}$ As $\mathcal{I} \models^d \mathrm{D}_F$, it follows that $\mathcal{I} \models^d F$. This proves the result. □

Based on this lemma, we can show the following theorem that allows us to deal with step (I) above, viz. computing a d-complete tq-table TQ^d from a given d-complete kq-table KQ^d.

Theorem 7.3.11 *Let $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$, let $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$ be a disjunctive f-subquery, and let KQ be a d-complete kq-table. Then $\mathbf{t} \models^d \mathrm{D}$ iff for each knot $\mathbf{k} \in \mathbb{K}$ with root $\mathbf{t}$ and each $\mathbf{k}$-hit h of KQ, we have $h \cap \mathrm{D} \neq \emptyset$.*

Proof. ($\rightarrow$) Suppose $\mathbf{t} \models^d \mathrm{D}$ but there exists a knot $\mathbf{k} \in \mathbb{K}$ with root $\mathbf{t}$ and a $\mathbf{k}$-hit h of KQ such that $h \cap \mathrm{D} = \emptyset$. By Lemma 7.3.10 above, some $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$ exists such that $\mathcal{I} \not\models^d F$ for each $F \in \mathbb{F}_q^{\mathcal{R}} \setminus h$, hence $\mathcal{I} \not\models^d F$ for each $F \in \mathrm{D}$. This contradicts $\mathbf{t} \models^d \mathrm{D}$.

($\leftarrow$) Suppose $\mathbf{t} \not\models^d \mathrm{D}$ but for each knot $\mathbf{k} \in \mathbb{K}$ with root $\mathbf{t}$ and each $\mathbf{k}$-hit h of KQ, we have $h \cap \mathrm{D} \neq \emptyset$. As $\mathbf{t} \not\models^d \mathrm{D}$, there exists some $\mathcal{I} \in \mathfrak{T}(\mathbf{t})$ such that $\mathcal{I} \not\models^d F$ for each $F \in \mathrm{D}$. Let

Algorithm 5: TQ_from_KQ

Input: a d-complete kq-table KQ
Output: a d-complete tq-table TQ
begin
  $R := \emptyset$;
  forall $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$ **do**
    Compute the set $H = \{h \subseteq \mathbb{F}_q^{\mathcal{R}} \mid k \in \mathbb{K}$ has root $\mathbf{t}$ and h is a k-hit of $\mathsf{KQ}\}$;
    forall $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$ **do**
      if *for each* $h \in H$ *we have* $h \cap \mathrm{D} \neq \emptyset$ **then**
        $R := R \cup \{(\mathbf{t}, \mathrm{D})\}$;
  return R
end

Figure 7.5: From knot-query tables to type-query tables (for the knot set $\mathbb{K}$ and the query q).

$\mathbf{k}$ be the knot at the root of $\mathcal{I}$ and consider the collection $\mathrm{D}'' = \{\mathrm{D}' \setminus \mathrm{D} \mid (\mathbf{k}, \mathrm{D}') \in \mathsf{KQ}\}$. A simple consequence of the d-completeness of KQ is that $\emptyset \notin \mathrm{D}''$. Hence, some minimal hitting set of D'' exist. Take any such minimal hitting set h. Clearly, $h \cap \mathrm{D} = \emptyset$ and h is a $\mathbf{k}$-hit of KQ. Contradiction. □

Using the above Theorem 7.3.11, we can compute a d-complete tq-table TQ^d out of a d-complete kq-table KQ^d. The procedure which exploits the theorem is shown in Figure 7.5.

We now show how to obtain KQ^{d+1} from TQ^d. Intuitively, to make the step from d to $d+1$, we must verify how each knot $(\mathbf{t}, \mathbf{S})$ in $\mathbb{K}$ can 'extend' the mappings that exist at the children in $\mathbf{S}$, or more precisely, the mappings that exist in the $\mathbf{t}_s$-trees of each $(\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}$. This extension relies on the mappings for each $\mathbf{t}_s$-tree captured by the minimal hitting sets of TQ^d, and is formalized in the following notion.

Definition 7.3.12 (assignment for k, D-fulfillment) *Let* $\mathbf{k} = (\mathbf{t}, \mathbf{S}) \in \mathbb{K}$ *and* $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$. *An* (f-subquery) assignment for $\mathbf{k}$ *is a function* $g : \mathbf{S} \to 2^{\mathbb{F}_q^{\mathcal{R}}}$. *We say that* g *is* D-fulfilling, *if there exist some* $F = (V, \Sigma) \in \mathrm{D}$ *and a mapping* $\phi : V \to \{\mathbf{t}\} \cup \mathbf{S}$ *satisfying the following conditions:*

(M1) For each $v \in V$ *with* $\phi(v) = \mathbf{t}$, *we have* $\{A \mid A(v) \in \mathsf{Atoms}(q)\} \subseteq \mathbf{t}$.

(M2) If $p(v, v') \in \mathsf{Atoms}(q)$ *is an atom with* $\phi(v) = \mathbf{t}$, *then* $\phi(v') \in \mathbf{S}$.

(M3) If v' *is open but not free in* V, *then* $\phi(v') = \mathbf{t}$.

(M4) For each $s = (\mathbf{r}_s, \mathbf{t}_s)$ *in* $\mathbf{S}$, *there exist* $F_s^i = (V_s^i, \Sigma_s^i) \in g(s)$, $1 \leq i \leq m$, *such that:*

  (a) $\{v \in V \mid \phi(v) = s\} = \bigcup_{1 \leq i \leq m}(V_s^i)$,

  (b) for every variable v' *that is is free in* V, $v' \in V_s^i$ *implies:*

    (i) $\Sigma(v') \subseteq \mathbf{r}_s$, *and*

    (ii) $\Sigma(v') \subseteq \Sigma_s^i(v')$ *if* v' *is free in* V_s^i.

  (c) for each variable v' *that is open in* V_s^i, *and for each* $p \in \mathsf{back}(V_s^i, v')$, *there is some* $p' \sqsubseteq^{\mathcal{R}} p$ *such that:*

    (i) $p' \in \mathbf{r}_s$, *and*

(ii) $p' \in \Sigma^i_s(v')$ *if* v' *is free in* V^i_s.

Note that if g is D-fulfilling, then every assignment g' that includes g, i.e., with $g'(s) \supseteq g(s)$ for all $s \in \mathbf{S}$, is D′-fulfilling for every D′ $\supseteq$ D.

Roughly, such an D-fulfilling assignment g witnesses the existence of a rooted match for D in an arbitrary $\mathbf{k}$-tree, assuming that each f-subquery F^i_s assigned to a child s has a match π_s in the respective $\mathbf{t}_s$-tree. Note that, by (M4.a), for each $v \in V$ either $\phi(v) = \mathbf{t}$, or $v \in V^i_s$ for some i and some s. Hence, ϕ shows that in every $\mathbf{k}$-tree $\mathcal{I}$ we can potentially find a rooted match π that has $\pi(v)$ for each v with $\phi(v) = \mathbf{t}$, and for every other variable in V, π coincides with the respective π_s. The conditions (M1), (M2) and (M3) ensure the satisfaction of (RP1), (RP2), and (RP3), respectively, while items (b) and (c) in the more elaborate (M4) are needed to ensure (RP4). This will be made more precise in the proof of the next theorem.

Example 7.3.13 *For the knot* $\mathbf{k}_1 = (\mathbf{t}_{B,C}, \{s_1, s_2\})$ *in Figure 7.1, where* $s_1 = (\{T\}, \mathbf{t}_A)$ *and* $s_2 = (\{P, T\}, \mathbf{t}_{A,B,C})\})$, *each assignment* g *with* $g(s_2) = \{F_4\}$ *is* $\{F_6\}$*-fulfilling, where as above* $F_4 = (\{v_4\}, \{(v_4, \{T\})\})$ *and* $F_6 = (\{v_3, v_4\}, \emptyset)$. *This is witnessed by the mapping* $\phi(v_3) = \mathbf{t}_{B,C}$ *and* $\phi(v_4) = s_2$, *which satisfies the conditions M1 to M4 (for M4, consider* $\{(v^1_{s_2}, \Sigma^1_{s_2})\} \subseteq g(s_2)$ *where* $(v^1_{s_2}, \Sigma^1_{s_2}) = F_4$; $y = v_4$ *is not free in* $\{v_3, v_4\}$, *but in* $V^1_{s_2} = \{v_4\}$*). The same assignment is also* $\{F_4\}$*-fulfilling; to see this, simply set* $\phi(v_4) = s_2$.

For a knot $\mathbf{k} = (\mathbf{t}, \mathbf{S})$ from $\mathbb{K}$, the existence of a D-fulfilling assignment g ensures the existence of a rooted prematch for D within depth $d + 1$ in an arbitrary $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$, provided that, for each $s \in \mathbf{S}$, the f-subqueries in $g(s)$ have rooted prematches in the subtree $\mathcal{I}_s$ of $\mathcal{I}$ rooted at s. Conversely, if $\mathcal{I} \models^{d+1}$ D for some $\mathcal{I}$, the assignment g such that, for each $s \in \mathbf{S}$, $g(s)$ is the set of all f-subqueries that are entailed at $\mathcal{I}_s$, is D-fulfilling. More precisely, we have:

Lemma 7.3.14 *Let* $\mathbf{k} = (\mathbf{t}, \mathbf{S})$ *be a knot in* $\mathbb{K}$ *and let* D $\subseteq \mathbb{F}^{\mathcal{R}}_q$. *Furthermore, let* $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$ *and let* g *be an assignment such that* $g(s) = \{F \in \mathbb{F}^{\mathcal{R}}_q \mid \mathcal{I}_s \models^d F\}$ *for all* $s \in \mathbf{S}$. *Then* $\mathcal{I} \models^{d+1}$ D *iff* g *is* D*-fulfilling.*

Proof. First we show ($\leftarrow$). If g is D-fulfilling, by assumption there is some $F_0 = (V_0, \Sigma_0) \in$ D and a mapping ϕ that satisfy M1 to M4 above. In particular, for each $s \in \mathbf{S}$, there exist $F^i_s = (V^i_s, \Sigma^i_s) \in g(s)$, $1 \leq i \leq m$, as in M4 above. For each of these F^i_s, $\mathcal{I}_s \models^d F^i_s$ holds, so there is a rooted prematch π^i_s for F^i_s in $\mathcal{I}_s$.

We construct a rooted prematch for D in $\mathcal{I}$, by combining ϕ and the different π^i_s. The new mapping $\pi : V_0 \to \Delta^{\mathcal{I}}$ is defined as follows:

$$\pi(v) = \begin{cases} \mathsf{root}(\mathcal{I}) & \text{if } \phi(v) = \mathbf{t}_0, \\ b^{\mathcal{I}}_s \cdot \pi^i_s(v) & \text{if } v \in V^i_s \text{ for some } s \text{ and } i \end{cases}$$

(recall that $b^{\mathcal{I}}_s$ is the unique node of $\mathcal{I}$ at depth 1 corresponding to s). Since $\{v \in V \mid \phi(v) = s\} = \bigcup_{1 \leq i \leq m}(V^i_s)$, π is well-defined and total. It only remains to show that π is a rooted prematch for F_0 in $\mathcal{I}$.

1. Consider any $A(v) \in \mathsf{Atoms}(q)$. If $v \in V^i_s$ for some s and i, then $\pi^i_s(v) \in A^{\mathcal{I}}$ because π^i_s is a rooted pre-match, and hence $\pi(v) \in A^{\mathcal{I}}$. Otherwise, $\phi(v) = \mathbf{t}_0$ and then M1 implies $A \in r$ and $\mathsf{root}(\mathcal{I}) \in A^{\mathcal{I}}$. Hence RP1 holds.

2. To show RP2, consider a pair $\{v, v'\} \subseteq V_0$ such that $p(v, v') \in \mathsf{Atoms}(q)$. The following cases are possible:

- $v \in V_s^i$ for some s. Then $v' \in V_s^i$ (due to the closure properties of the set V_s^i). Since π_s^i is a rooted prematch, $\pi_s^i(v')$ is a p-follower of $\pi_s^i(v)$ in $\mathcal{I}_s$, and hence $\pi(v')$ is a p-follower of $\pi(v)$ in $\mathcal{I}$.
- $\phi(v) = \mathbf{t}_0$. Then by M2 we have $\phi(v') = s$ for some $s = (\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}$. Also, by M4, $v' \in V_s^i$ for some i and, as $v \notin V_s^i$, we have $p \in \mathsf{back}(V_s^i, v')$, and by the last item of M4, there is some $p' \in \mathbf{r}_s$ such that $p' \sqsubseteq^{\mathcal{R}} p$ and, additionally, $p' \in \Sigma_s^i(v')$ whenever v' is free in V_s^i. Since π^i is a rooted prematch for F_s^i in $\mathcal{I}_s$, it satisfies RP3 and RP4. This implies that either $\pi_s^i(v')$ is the root of $\mathcal{I}_s$, or v' is free in V_s^i. In the former case, $\pi(v')$ is a p-follower of $\pi(v)$ as desired. In the latter case, $p' \in \Sigma_s^i(v')$ and $\pi_s^i(v')$ is a p'-follower of the root of $\mathcal{I}_s$ by RP4, which also implies that $\pi(v')$ is a p-follower of $\pi(v)$.

3. RP3 follows directly from M3.

4. To show RP4, consider any v' that is free in V_0 and an arbitrary $p \in \Sigma(v')$. There are two cases:

   - $\phi(v') = \mathbf{t}_0$. Then $\pi(v') = \mathsf{root}(\mathcal{I})$ and RP4 holds.
   - $v' \in V_s^i$ for some s and i. Then by the second item of M4, we have $p \in \mathbf{r}_s$ and either (i) v' is open but not free in V_s^i, or (ii) $p \in \Sigma_s^i(v')$. Since π_s^i is a rooted prematch for F_s^i, it satisfies Definition 7.3.4. In case (i), RP3 implies that $\pi_s^i(v')$ is the root of $\mathcal{I}_s$, and hence a p-follower of $\mathsf{root}(\mathcal{I})$ in $\mathcal{I}$ as desired. In case (ii), RP4 implies that $\pi_s^i(v')$ is either the root of $\mathcal{I}_s$ as above, or a p-follower of it. As $p \in \mathbf{r}_s$, again in both cases $\pi_s^i(v')$ is a p-follower of $\mathsf{root}(\mathcal{I})$ in $\mathcal{I}$ as desired.

This shows that π is a rooted prematch for F_0 and hence $\mathcal{I} \models \mathrm{D}$. Furthermore, since for each $v \in V_0$ the length of $\pi(v) = 0$ if $\phi(v) = \mathbf{t}_0$ and $\pi(v) = \pi_s(v) + 1$ otherwise. As the length of $\pi_s(v)$ is bounded by d, we have $\mathcal{I} \models^{d+1} \mathrm{D}$.

Now, to show $(\rightarrow)$, we assume $\mathcal{I} \models^{d+1} \mathrm{D}$, and consider an assignment g with $F \in g(s)$ for each $F \in \mathbb{F}_q^{\mathcal{R}}$ and each $s \in \mathbf{S}$ such that $\mathcal{I}_s \models^d (V, \Sigma)$. To see that g is D-fulfilling, we start by observing that, by assumption, there is an $F = (V, \Sigma) \in \mathrm{D}$ and a rooted prematch π for F in $\mathcal{I}$. For each $s \in \mathbf{S}$, let V_s contain all variables $v \in V$ such that $\pi(v)$ is in the tree $\mathcal{I}_s$. We partition V_s into sets of variables $V_s^1, \dots, V_s^m$ that are connected in q. We define a function Σ_s^i that maps each free $v' \in V_s^i$ to a set of transitive roles as follows:

$$\begin{aligned}\Sigma_s^i(v') \quad = \{p' \in \mathsf{N_R}(\mathcal{K}) \mid \quad & p' \sqsubseteq^{\mathcal{R}} p,\ p \in \mathsf{back}(V_s^i, v'),\ \mathsf{trans}(p') \in \mathcal{R}, \\ & \text{and } \pi(v') \text{ is a } p'\text{-follower of } \mathsf{root}(\mathcal{I}_s)\}\end{aligned}$$

Clearly, each V_s^i is closed under the rules (a.i) and (a.ii) of Definition 7.3.2. Hence, to see that each (V_s^i, Σ_s^i) is an f-subquery, it suffices to observe that, since π is a rooted prematch, $\pi(v')$ is a p-follower of $\mathsf{root}(\mathcal{I})$ for each $p \in \mathsf{back}(V_s^i, v')$, and hence condition (b) also holds.

It is also easy to see that, for each s and each i, $\mathcal{I}_s \models^d (V_s^i, \Sigma_s^i)$ (simply restrict π to the corresponding variables to obtain a rooted prematch in $\mathcal{I}_s$). So, by our assumption about g, $(V_s^i, \Sigma_s^i) \in g(s)$.

Now we can define a mapping $\phi : V \to \{\mathbf{t}_0\} \cup \mathbf{S}$ that witnesses that g is D-fulfilling by setting $\phi(v) = \mathbf{t}_0$ if $\pi(v) = \mathsf{root}(\mathcal{I})$, and $\phi(v) = s$ if $v \in V_s$. It is straightforward to verify that ϕ satisfies M1 to M3 in Definition 7.3.12. For M4, we can use for each $s \in \mathbf{S}$ the $F_s^i = (V_s^i, \Sigma_s^i)$, $1 \leq i \leq m$ defined above, since they are in $g(s)$. Then the first item is trivial; the other two can be verified as follows:

- Consider any $v' \in V_s^i$ that is free in V, and any $p \in \Sigma(v')$. Since π is a rooted prematch and $\pi(v') \neq \mathsf{root}(\mathcal{I})$, by RP4 $\pi(v')$ is a p-follower of $\mathsf{root}(\mathcal{I})$. Hence, if $s = (\mathbf{r}_s, \mathbf{t}_s)$, then $p \in \mathbf{r}_s$ and either $\pi(v') = b_s^{\mathcal{I}}$ or $\pi(v')$ is a p-follower of $b_s^{\mathcal{I}}$. If v' is also free in V_s^i then in both cases $p \in \Sigma_s^i(v')$ by construction of Σ_s^i.

- Consider any $v' \in V_s^i$ that is open in V_s^i, and any $p \in \mathsf{back}(V_s^i, v')$ such that $p(v, v') \in \mathsf{Atoms}(q)$. Since π is a rooted prematch and $x \notin V_s$, either (a) $\pi(v) = \mathsf{root}(\mathcal{I})$ or (b) $x \notin V$. In case (a), by RP2, $\pi(v')$ is a p'-follower of $\mathsf{root}(\mathcal{I})$. In case (b), v' is open in V and, moreover, free in V (as v' being open but not free would imply $\pi(v') = \mathsf{root}(\mathcal{I})$, contradicting $v' \in V_s$). Hence, there is some $p' \sqsubseteq^{\mathcal{R}} p$ with $p' \in \Sigma(v')$ and, by RP4, we also have that $\pi(v')$ is a p'-follower of $\mathsf{root}(\mathcal{I})$. In both cases (a) and (b), it thus follows $p' \in \mathbf{r}_s$, where $s = (\mathbf{r}_s, \mathbf{t}_s)$. It also follows that either $\pi(v') = b_s^{\mathcal{I}}$, or $\pi(v')$ is a p'-follower of $b_s^{\mathcal{I}}$ and p' is transitive. If v' is free in V_s^i, then by construction in both cases $p' \in \Sigma_s^i$.

□

The step from TQ^d to KQ^{d+1} computes the f-subqueries D for which the $\mathbf{t}$-hits of TQ^d are D-fulfilling.

Theorem 7.3.15 *Suppose* TQ *is a d-complete tq-table,* $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$ *is a disjunctive f-subquery, and* $\mathbf{k} = (\mathbf{t}, \mathbf{S})$ *is a knot in* $\mathbb{K}$*. Then* $\mathbf{k} \models^{d+1} \mathrm{D}$ *iff every assignment g that maps each* $(\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}$ *to a* $\mathbf{t}_s$*-hit of* TQ *is* D*-fulfilling.*

Proof. Let G denote the set of all assignments for $\mathbf{k}$ such that, for each $s = (\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}$, $g(s)$ is a $\mathbf{t}_s$-hit of TQ,

($\rightarrow$) Suppose $\mathbf{k} \models^{d+1} \mathrm{D}$. and consider an arbitrary $g \in G$. By Lemma 7.3.10, there is a tree $\mathcal{I}_s \in \mathfrak{T}(\mathbf{t}_s)$ that satisfies exactly the f-subqueries in $g(s)$. Let $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$ be the tree that coincides with all these $\mathcal{I}_s$. As $\mathcal{I} \models^{d+1} \mathrm{D}$, then by Lemma 7.3.14, g is D-fulfilling.

($\leftarrow$). Now assume that each $g \in G$ is D-fulfilling, and consider an arbitrary $\mathcal{I} \in \mathfrak{T}(\mathbf{k})$. For each $s = (\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}$, let $fq(s)$ be the set of all f-subqueries F such that $\mathcal{I}_s \models^d F$, and let g' be the assignment such that $g'(s) = fq(s)$ for all $s \in \mathbf{S}$. Then, $fq(s) \cap \mathrm{D} \neq \emptyset$ must hold for each $(\mathbf{t}, \mathrm{D}) \in \mathsf{TQ}$ such that $\mathbf{t} = \mathbf{t}_s$; hence, there exists some $\mathbf{t}_s$-hit h_s of TQ such that $h_s \subseteq fq(s)$. By assumption, there exists some $g \in G$ such that $g(s) = h_s$ for all $s \in \mathbf{S}$. As g is D-fulfilling and g' includes g, i.e., $g'(s) \supseteq g(s)$ for all s, also g' is D-fulfilling. Thus by Lemma 7.3.14, $\mathcal{I} \models^{d+1} \mathrm{D}$. Hence, $\mathbf{k} \models^{d+1} \mathrm{D}$. □

Example 7.3.16 *Reconsider the knot* $\mathbf{k}_1 = (\mathbf{t}_{B,C}, \{s_1, s_2\})$ *in Figure 7.1, where* $s_1 = (\{t_1\}, \mathbf{t}_A)$ *and* $s_2 = (\{p_1, t_1\}, \mathbf{t}_{A,B,C})\}$*, and the tq-table* TQ^0 *in Example 7.3.8. As for* s_1*, the single* $\mathbf{t}_A$*-hit of* TQ^0 *is* $\emptyset$ *(by minimality, as there is no entry for* $\mathbf{t}_A$ *in* TQ^0*), and for* s_2*, the single* $\mathbf{t}_{A,B,C}$*-hit of* TQ^0 *is* $\{F_4, F_5\}$*. Hence, the set of assignments* G *consists of* g *where* $g(s_1) = \emptyset$ *and* $g(s_2) = \{F_4, F_5\}$*. Since we know from Example 7.3.13 that* g *is* $\{F_4\}$*-fulfilling and* $\{F_6\}$*-fulfilling, it follows that* $\mathbf{k}_1 \models^1 \mathrm{D}$ *for every disjunctive f-subquery* D *that includes either* F_4 *or* F_6 *(or both); in particular,* $\mathbf{k}_1 \models^1 \{F_4\}$ *and* $\mathbf{k}_1 \models^1 \{F_6\}$*, and thus* $(k_1, \{F_4\})$ *and* $(k_1, \{F_6\})$ *are included in* KQ^1*.*

So far, we have only encountered singleton disjunctive f-subqueries in tq- and kq-tables, but not always complete such tables can be derived where only singleton f-subqueries occur. For example, if we continue the computation above, we would eventually obtain a 2-complete tq-table TQ^2 *such that each of its* $\mathbf{t}_{ABC}$*-hits contains either* F_1 *or* F_2*, and one can infer that* $\mathbf{k}_1 \models^3 \{F_1, F_2\}$ *although neither* $\mathbf{k}_1 \models^3 \{F_1\}$ *nor* $\mathbf{k}_1 \models^3 \{F_2\}$ *holds.*

Algorithm 6: KQ_from_TQ

Input: a d-complete tq-table TQ
Output: a $d{+}1$–complete kq-table KQ
begin
  $\mathsf{R} := \emptyset$;
  forall $\mathbf{k} \in \mathbb{K}$ **do**
    $G := \emptyset$;
    Add to G each $g : \mathbf{S} \to 2^{\mathbb{F}_q^{\mathcal{R}}}$ such that for each $(\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}$, $g((\mathbf{r}_s, \mathbf{t}_s))$ is a $\mathbf{t}_s$-hit of TQ;
    forall $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$ **do**
      if *each* $g \in G$ *is* D-*fulfilling* **then**
        $\mathsf{R} := \mathsf{R} \cup \{(k, \mathrm{D})\}$;
  return R
end

Figure 7.6: From type-query tables to knot-query tables (for the knot set $\mathbb{K}$ and the query q).

The algorithm based on Theorem 7.3.15, compute_TQ, is shown in Figure 7.6. Using the algorithms presented so far, we can compute a d-complete tq-table TQ^d for any $d > 0$. Furthermore, as (i) the entailment relation $\mathbf{t} \models \mathrm{D}$ materializes within bounded depth d, and (ii) $\mathbf{t} \models^d \mathrm{D}$ implies $\mathbf{t} \models^{d+1} \mathrm{D}$, the computation reaches a fixed-point after finitely many steps and we can obtain one tq-table TQ that is complete for each $d \in \mathbb{N}$. The complexity analysis of our algorithm in Section 7.4 will actually reveal that the number of steps is double-exponentially bounded in the size of the query and the knowledge.

Proposition 7.3.17 *For each type* $\mathbf{t}$ *such that* $\mathbf{t} \models \mathrm{D}$*, there exists some* $d \in \mathbb{N}$ *such that* $\mathbf{t} \models^d \mathrm{D}$.

Proof. To give a bound d, we construct a tree of interpretations which captures parts of trees in $\mathfrak{T}(\mathbf{t})$ that are relevant for the mappings of D. For an integer $n \geq 0$, let $\mathcal{I}_{\uparrow n}$ be the restriction of a tree-shaped $\mathcal{I}$ up to depth n. We define a tree-shaped graph $T = (V, E)$ as follows:

- the vertex set is $V = \{\mathcal{I}_{\uparrow n} \mid \mathcal{I} \in \mathfrak{T}(\mathbf{t}) \wedge n \geq 0\}$, and
- the child relation is $E = \{\langle \mathcal{I}_{\uparrow n}, \mathcal{I}_{\uparrow n+1} \rangle \mid \mathcal{I} \in \mathfrak{T}(\mathbf{t}) \wedge n \geq 0\}$.

Intuitively, each pair in E represents an expansion of the levels 0, 1,..., $n-1$ of $\mathcal{I}$ by another level using the knots in $\mathbb{K}$. Hence each path in T corresponds to an interpretation in $\mathfrak{T}(\mathbf{t})$. Observe that T is finitely branching. Consider now the set P of all nodes $\mathcal{I}_{\uparrow n} \in V$ such that $\mathcal{I}_{\uparrow n} \models \mathrm{D}$ and $\mathcal{I}_{\uparrow n-1} \not\models \mathrm{D}$ (the latter in case where $n > 0$). As $\mathbf{t} \models \mathrm{D}$, by construction of T each path in it contains some node from P. Let T' result from T by removing all successors of nodes in P. Since T' does not have infinite branches and is finitely branching, by König's Lemma T' is finite. Hence for each $\mathcal{I} \in \mathfrak{T}(\mathbf{t})$ the match for D occurs within finite depth d, where d is the length of the longest branch in T'. □

Corollary 7.3.18 *For* $\mathbb{K}$ *and* q*, there exists* $d \in \mathbb{N}$ *such that for every type* $\mathbf{t} \in \mathsf{typesC}(\mathbb{K})$ *and every* $\mathrm{D} \subseteq \mathbb{F}_q^{\mathcal{R}}$*,* $\mathbf{t} \models \mathrm{D}$ *iff* $\mathbf{t} \models^d \mathrm{D}$*. Furthermore, if a tq-table* TQ *is complete, then it is* d*-complete for some finite* $d \in \mathbb{N}$*.*

Algorithm 7: compute_TQ

Input: a knot set $\mathbb{K}$ and a CQ q
Output: a complete tq-table TQ
begin
  $\mathsf{TQ}^0 := \mathsf{TQ\_Zero}$;
  $d = 0$;
  repeat
    $\mathsf{KQ}^{d+1} := \mathsf{KQ\_from\_TQ}(\mathsf{TQ}^d)$;
    $\mathsf{TQ}^{d+1} := \mathsf{TQ\_from\_KQ}(\mathsf{KQ}^{d+1})$;
    $d := d + 1$;
  until $\mathsf{TQ}^{d-1} \neq \mathsf{TQ}^d$;
  return TQ^d
end

Figure 7.7: Computing a complete type-query table (for the knot set $\mathbb{K}$ and the query q).

7.3.3 Query Entailment over full KBs

We now show how to use the knot set $\mathbb{K}$ and a complete tq-table TQ for $\mathbb{K}$ and the query q to answer q over $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. The basic idea is to 'compile' $\mathcal{A}$, $\mathbb{K}$ and TQ into a set $\mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})$ of ABoxes $\mathcal{A}'$ that we call *expansions* of $\mathcal{A}$, in such a way that

(∗) a tuple $\vec{a}$ is an answer for q in $\mathcal{K}$ iff it is an answer for q in $\langle \mathcal{A}', \emptyset, \emptyset \rangle$ for *every* expansion $\mathcal{A}'$ of $\mathcal{A}$.

We can then answer q by evaluating it over the set of expansions $\mathcal{A}'$, which are small and rather simple structures.

Notation 7.3.19 (Query answers over an ABox) *For readability, in what follows, we use* $\mathsf{ans}(q, \mathcal{A})$ *to denote the set* $\mathsf{ans}(q, \langle \mathcal{A}, \emptyset, \emptyset \rangle)$*, that is*

$$\mathsf{ans}(q, \mathcal{A}) = \{\vec{a} = a_1, \ldots, a_n \mid \mathcal{I} \models q(\vec{a}) \text{ for each model } \mathcal{I} \text{ of } \mathcal{A}\}.$$

It is easy to see that deciding $\mathsf{ans}(q, \mathcal{A})$ *amounts to deciding whether there is a mapping* $\psi : \mathsf{VI}(q) \to \mathsf{N_I}(\mathcal{A})$ *such that*

- $\psi(x_i) = a_i$ *for each answer variable* x_i,
- $\{A \mid A(v) \in \mathsf{Atoms}(q)\} \subseteq \{A \mid A(\psi(v)) \in \mathcal{A}\}$ *for each* $A(v) \in \mathsf{Atoms}(q)$, *and*
- $\{p \mid p(v, v') \in \mathsf{Atoms}(q)\} \subseteq \{p \mid p(\psi(v), \psi(v')) \in \mathcal{A}\}$ *for each* $p(v, v') \in \mathsf{Atoms}(q)$.

That is, query answering over ABoxes is just like answering q over a regular relational database, and can be efficiently done using existing technology. We will see in the next section that one can even go further: evaluating answering q over *all* the expansions $\mathcal{A}'$ can be reduced to querying one single Datalog program.

Let us see how to obtain a set of expansions that satisfy (∗). Intuitively, each expansion $\mathcal{A}'$ is obtained by adding to $\mathcal{A}$: (1) an ABox completion θ, (2) a knot $\mathbf{k}_a$ from $\mathbb{K}$ for each individual a, and (3) an f-subquery assignment $g_{\mathbf{k}}(a)$ for each knot $\mathbf{k}_a$. The idea is to have a one-to-one correspondence between the query matches in models of $\mathcal{A}'$, and the query prematches in all the forest bases of $\mathcal{K}$ that 'coincide' with $\mathcal{A}'$, that is, in each forest base $\mathcal{I} \in \mathfrak{F}(\theta, \mathbb{K})$ such that (i) the subtree of depth at most one rooted at each individual a coincides with $\mathbf{k}_a$, and (ii) for each level one node $b_s^{\mathcal{I}}$ (where s is a child of $\mathbf{k}_a$), the subtree $\mathcal{I}_s$ (with root $b_s^{\mathcal{I}}$) has a rooted match

exactly for the subqueries assigned by $g_{\mathbf{k}}(a)$. We will show that $(*)$ is guaranteed if we generate an expansion for each non-conflicting combination of (i)–(iii) above.

To generate each expansion $\mathcal{A}'$, we proceed in three steps:

(1) First we extend $\mathcal{A}$ with the completion θ. In this step, we ensure that the concepts and roles satisfied by the individuals coincide with θ, by adding a concept membership assertion $A(a)$ for each concept A in $\theta(a)$, and a role membership assertion $p(a,b)$ for each role p in $\theta(a,b)$, for all the individuals a, b. Then each forest base $\mathcal{I}$ that is a model of the extended ABox must coincide with θ in its initial part, that is, it must have $\{C \in Cl_{\mathsf{C}}(\mathcal{K}) \mid a^{\mathcal{I}} \in C^{\mathcal{I}}\} = \theta(a)$ and $\{p \in \mathsf{N_R}(\mathcal{K}) \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in C^{\mathcal{I}}\} = \theta(a,b)$ for all individuals a, b.

(2) Then, for each individual a, we 'freeze' the knot $\mathbf{k}_a$ and add it to $\mathcal{A}$ as a set of assertions, using a fresh variable a_s for each child s of $\mathbf{k}_a$. As a result we obtain an ABox $\mathcal{A}_a$ such that a forest base $\mathcal{I}$ is a model of $\mathcal{A}_a$ iff $\mathcal{I} \in \mathfrak{F}(\theta, \mathbb{K})$, and for each a, the knot $\xi(a)$ in Definition 7.2.5 coincides with $\mathbf{k}_a$.

(3) Finally, for each child $s = (\mathbf{t}_s, \mathbf{r}_s)$ of $\mathbf{k}_a$, and each f-subquery $F = (V, \Sigma) \in g_{\mathbf{k}}(a)(s)$, we also 'freeze' F and state it as a set of ABox assertions, using a fresh variable $a_{s,F,v}$ for each $v \in V$. If $g_{\mathbf{k}}(a)$ is such that $\mathbf{t}_s \models F$ for each $F \in g_{\mathbf{k}}(a)$, then we know that F has a rooted prematch in every subtree $\mathcal{I}_{|a_s}$ of a forest base $\mathcal{I}$. So we enforce in $\mathcal{A}'$ the existence of a match $\pi_{s,F}$ for F rooted at a_s, exactly for the F in $g_{\mathbf{k}}(a)$. This will ensure $(*)$.

To this end we add, for all atoms $A(v)$ and $p(v, v')$ in $\mathsf{Atoms}(q|_V)$, assertions

- $A(a_s)$ for each atom $A(v)$ where if v is open and not free in V,
- $A(a_{s,F,v})$ otherwise,
- $p(a_s, a_{s,F,v'})$ if v is open and not free in V, and v' closed, and
- $p(a_{s,F,v}, a_{s,F,v'})$ otherwise.

Furthermore, we add an assertion $p(a_s, a_{s,F,v'})$ for each $(v,p) \in \Sigma$.

Formally, ABox expansions are defined as follows.

Definition 7.3.20 (ABox expansions) *Let TQ be a complete tq-table, let θ be a completion for $\mathcal{K}$, and let $k : \mathsf{N_I}(\mathcal{A}) \to \mathbb{K}$ be a function that assigns a knot $k(a) = (\mathbf{t}_a, \mathbf{S}_a)$ with $\mathbf{t}_a = \theta(a)$ to each individual a. We define $\mathbf{S}(k(a)) = \{s \in \mathbf{S}_a \mid k(a) = (\mathbf{t}_a, \mathbf{S}_a)\}$. Furthermore, let G_k be a function that maps each individual a to an assignment $G_k(a)$ for $k(a)$ such that $G_k(a)(s)$ is a $\mathbf{t}$-hit of TQ for each $s = (\mathbf{r}, \mathbf{t}) \in \mathbf{S}(k(a))$.*

Then the following ABox $\mathsf{abox}(\mathcal{A}, \theta, k, G_k)$ is called a $(\mathbb{K}, \mathsf{TQ})$-expansion of $\mathcal{A}$*:*

1) For the completion θ,

$$\mathsf{abox}(\theta) \;=\; \textstyle\bigcup_{a \in \mathsf{N_I}(\mathcal{A})} \{A(a) \mid A \in \theta(a) \cap \mathsf{N_C}(\mathcal{K})\} \cup \bigcup_{a,b \in \mathsf{N_I}(\mathcal{A})} \{p(a,b) \mid p \in \theta(a,b)\}.$$

2) Let $k(a) = (\mathbf{t}_a, \mathbf{S}_a)$, and for each $s \in \mathbf{S}_a$, let a_s be a fresh individual. Then

$$\begin{aligned} \mathsf{abox}(k(a)) \;=\; & \{A(a) \mid A \in \mathbf{t}_a \cap \mathsf{N_C}(\mathcal{K})\} \cup \\ & \textstyle\bigcup_{s=(\mathbf{r}_s,\mathbf{t}_s) \in \mathbf{S}_a} \{p(a, a_s) \mid p \in \mathbf{r}_s\} \cup \{A(a_s) \mid A \in \mathbf{t}_s \cap \mathsf{N_C}(\mathcal{K})\}. \end{aligned}$$

3) For each $s \in \mathbf{S}(k(a))$, $F = (V, \sigma) \in G_k(a)(s)$, and $v \in V$, let $a_{s,F,v}$ be a fresh individual in $\mathsf{N_I}$. Then

$$\begin{aligned}\mathsf{abox}(s) \;=\; \textstyle\bigcup_{F\in G_k(a)(s)} & \{A(a_s) \mid A(v)\in \mathsf{Atoms}(q), v\in \mathit{onf}(V)\}\cup\\ & \{A(a_{s,F,v}) \mid A(v)\in \mathsf{Atoms}(q), v\in V\setminus \mathit{onf}(V)\}\cup\\ & \{p(a_s,a_{s,F,v}) \mid p(v,v')\in \mathsf{Atoms}(q), v\in \mathit{onf}(V), v'\in V\setminus \mathit{onf}(V)\}\cup\\ & \{p(a_{s,F,v},a_{s,F,v}) \mid p(v,v')\in \mathsf{Atoms}(q), \{v,v'\}\subseteq V\setminus \mathit{onf}(V)\}\cup\\ & \{p(a_s,a_{s,F,v}) \mid (v,p)\in\Sigma\},\end{aligned}$$

where $\mathit{onf}(V)\subseteq o(V)$ *denotes the variables that are* open and not free *in* V.

4) for each $a\in \mathsf{N_I}(\mathcal{A})$,

$$\mathsf{abox}(a,k(a),G_k(a)) \;=\; \mathsf{abox}(k(a))\cup \textstyle\bigcup_{s\in\mathbf{S}(a)}\mathsf{abox}(s)$$

5) Let $\mathcal{A}' = \mathcal{A}\cup \mathsf{abox}(\theta)\cup\bigcup_{a\in\mathsf{N_I}(\mathcal{A})}\mathsf{abox}(a,k(a),G_k(a))$ *be the union of all the constructed ABoxes.*

Then $\mathsf{abox}(\mathcal{A},\theta,k,G)$ *is obtained by closing* $\mathcal{A}'$ *under the following rules:*

- *if* $\mathsf{trans}(p)\in\mathcal{R}$, $p(a,b)\in\mathcal{A}'$ *and* $p(b,c)\in\mathcal{A}'$, *then* $p(a,c)\in\mathcal{A}'$, *and*
- *if* $p'\sqsubseteq^{\mathcal{R}} p$ *and* $p'(a,b)\in\mathcal{A}'$ *then* $p(a,b)\in\mathcal{A}'$.

We denote with $\mathsf{exp}(\mathcal{A},\mathbb{K},\mathsf{TQ})$ *the set of all* $(\mathbb{K},\mathsf{TQ})$*-expansions of* $\mathcal{A}$.

Now we can formally state the main result of this section, which shows that we can reduce CQ answering over $\mathcal{K}$ to CQ answering over expanded ABoxes (i.e., relational databases).

Theorem 7.3.21 (Main result) *Let* TQ *be a complete tq-table for* $\mathbb{K}$ *and* q. *Then*

$$\mathsf{ans}(q,\mathcal{K}) = \bigcap_{\mathcal{A}'\in\mathsf{exp}(\mathcal{A},\mathbb{K},\mathsf{TQ})}\mathsf{ans}(q,\mathcal{A}').$$

Proof. ($\subseteq$) To show that $\mathsf{ans}(q,\mathcal{K})\subseteq\bigcap_{\mathcal{A}'\in\mathsf{exp}(\mathcal{A},\mathbb{K},\mathsf{TQ})}\mathsf{ans}(q,\mathcal{A}')$, suppose $\vec{a}=a_1,\ldots,a_n\in\mathsf{ans}(q,\mathcal{K})$, and consider an arbitrary $\mathcal{A}'=\mathsf{abox}(\mathcal{A},\theta,k,G_k)\in\mathsf{exp}(\mathcal{A},\mathbb{K},\mathsf{TQ})$. Let $\mathcal{I}$ be a forest base such that the closure of $\mathcal{I}$ is a model of $\mathcal{A}'$.

For each individual $a\in\mathsf{N_I}(\mathcal{A})$ and each $a_s=(\mathbf{r}_s,\mathbf{t}_s)\in\mathbf{S}(k(a))$, $G_k(a)(a_s)$ is a $\mathbf{t}_s$-hit of TQ. Moreover, TQ is complete. Hence, by Lemma 7.3.10 and Corollary 7.3.18, there exists some $\mathcal{I}_{a_s}\in\mathfrak{T}(\mathbf{t}_s)$ such that $\{F\mid\mathcal{I}_{a_s}\models F\}=G_k(a)(s)$. Let $\xi_{a_s}:\Delta^{\mathcal{I}}\to\mathbb{K}$ be a function inducing $\mathcal{I}_{a_s}$.

We consider a forest base model $\mathcal{I}'\in\mathfrak{F}(\theta,\mathbb{K})$ such that:

(1) The domain of $\mathcal{I}'$ is

$$\Delta^{\mathcal{I}'}=\{1,\ldots,|\mathsf{N_I}(\mathcal{A})|\}\cup\bigcup_{a\in\mathsf{N_I}(\mathcal{A}),a_s\in\mathbf{S}(k(a))}\{\tau(a)\cdot\tau_a(a_s)\cdot w\mid\mathsf{root}(\mathcal{I}_{a_s})\cdot w\in\Delta^{\mathcal{I}_{a_s}}\}$$

where $\tau:\mathsf{N_I}(\mathcal{A})\to\{1,\ldots,|\mathsf{N_I}(\mathcal{A})|\}$ and $\tau_a:\mathbf{S}(k(a))\to\{1,\ldots,|\mathbf{S}(k(a))|\}$ for each $a\in\mathsf{N_I}(\mathcal{A})$, are arbitrary bijections.

That is, $\Delta^{\mathcal{I}'}$ simply extends the union of all $\Delta^{\mathcal{I}_{a_s}}$ with the roots $\{1,\ldots,|\mathsf{N_I}(\mathcal{A})|\}$ that are the interpretation of the individuals, renaming nodes accordingly to ensure that $\Delta^{\mathcal{I}'}$ is a forest.

(2) $\mathcal{I}'$ is induced by a function ξ such that:

- for each individual $a\in\mathsf{N_I}(\mathcal{A})$, $\xi(a^{\mathcal{I}'})=k(a)$, and

- for every other node $\tau(a) \cdot \tau_a(a_s) \cdot w \in \Delta^{\mathcal{I}'}$, $\xi(\tau(a) \cdot \tau_a(a_s) \cdot w) = \xi_{a_s}(\mathsf{root}(\mathcal{I}_{a_s}) \cdot w)$.

That is, each individual a instantiates the knot $k(a)$, and the tree rooted at child $\tau_a(a_s)$ of each $\tau(a)$ is an isomorphic copy of the interpretation $\mathcal{I}_{a_s}$.

As $\mathcal{I}' \in \mathfrak{F}(\theta, \mathbb{K})$, by Lemma 7.2.6 $\mathcal{I}' \models \mathcal{K}$, and as $\vec{a} = a_1, \ldots, a_n \in \mathsf{ans}(q, \mathcal{K})$ by assumption, there is a prematch π' for q in $\mathcal{I}'$ such that $\pi'(x_i) = a_i$ for each for each $0 \leq i \leq n$.

We use this prematch π' to show that a match π for q in $\mathcal{I}$ exists. First we define, for each $a \in \mathsf{N_I}(\mathcal{A})$ and each $a_s \in \mathbf{S}(k(a))$, a set $\mathbb{F}_q^{\mathcal{R}}(a_s)$ of f-queries $F = (V, \Sigma_V)$ obtained as follows.

- V is the set of variables of some maximal connected subquery of $q|_{V(a_s)}$, where $V(a_s) = \{v \in \mathsf{V_I}(q) \mid \pi'(v) = \tau(a) \cdot \tau_a(a_s) \cdot w \text{ for some } w \in \mathbb{N}^*\}$.
- $\Sigma_V = \{(v, p) \in V \times \mathsf{N_R}(\mathcal{K}) \mid v \text{ is open in } V(a_s),\ p \in \mathsf{back}(V, v),\ p' \sqsubseteq^{\mathcal{R}} p,\ \mathsf{trans}(p') \in \mathcal{R} \text{ and } \pi'(v) \text{ is a } p'\text{-follower of } \tau(a) \cdot \tau_a(a_s)\}$.

For each variable $v \in \mathsf{V_I}(q)$, either $\pi'(v) = a^{\mathcal{I}'}$ for some $a \in \mathsf{N_I}(\mathcal{A})$, or $v \in V$ for one of these F. Hence we can define a binding π for q as follows:

$$\pi(v) = \begin{cases} a^{\mathcal{I}} & \text{if } \pi'(v) = \tau(a) \text{ for some } a \in \mathsf{N_I}(\mathcal{A}), \\ {a_s}^{\mathcal{I}} & \text{if } \pi'(v) = \tau_a(a_s) \text{ for some } a \in \mathsf{N_I}(\mathcal{A}) \text{ and some } a_s \in \mathbf{S}(k(a)), \\ {a_{s,F,v}}^{\mathcal{I}} & \text{if } \pi'(v) = \tau_a(a_s) \cdot w \text{ for some } w \in \mathbb{N}^*, w \neq \varepsilon \text{ and,} \\ & \quad v \in V \text{ for some } F = (V, \Sigma) \in \mathbb{F}_q^{\mathcal{R}}(a_s). \end{cases}$$

For each such F, the restriction of π' to the variables in V is a rooted match for F. Hence $\mathcal{I}'_s \models F$, which implies $F \in G_k(a)(s)$. Hence it is not hard to verify that $\pi(v')$ is a p-follower of $\pi(v)$ for each $p(v, v') \in \mathsf{Atoms}(q)$. Since $\mathcal{A}'$ is closed under transitivity and role inclusions, we get $(\pi(v), \pi(v')) \in p^{\mathcal{I}}$. As $\pi(v) \in A^{\mathcal{I}}$ for each $A(v) \in \mathsf{Atoms}(q)$, π is a match for q in $\mathcal{I}$. Further, π does not modify the matches to individuals from π, and thus preserves query answers. That is, since each answer variable x_i has $\pi'(x_i) = {a_i}^{\mathcal{I}'}$ for some individual a_i, then $\pi(x_i) = {a_i}^{\mathcal{I}}$. This shows that $\mathcal{I} \models q(\vec{a})$ and, as $\mathcal{A}'$ and $\mathcal{I}$ were selected arbitrarily, it follows that $\vec{a} \in \mathsf{ans}(q, \mathcal{A}')$ for every expansion $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})$.

($\supseteq$) To show the other direction, assume that $\vec{a} \in \mathsf{ans}(q, \mathcal{A}')$ for each $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})$. Let $\mathcal{I} \in \mathfrak{F}_{\mathbb{K}}(\mathcal{K})$. By the $\mathsf{ABoxTypes}(\mathcal{K})$-completeness of $\mathbb{K}$, we have $\mathcal{I} \in \mathfrak{F}(\theta, \mathbb{K})$, i.e., $\mathcal{I}$ is induced by θ and $\mathbb{K}$, for some ABox completion θ for $\mathcal{K}$, and there is some function ξ that induces $\mathcal{I}$. For each individual $a \in \mathsf{N_I}(\mathcal{A})$, let $k(a) = \xi(a^{\mathcal{I}}) = (\mathbf{t}_a, \mathbf{S}_a)$. Let $\mathbf{S}(k(a)) = \{s \in \mathbf{S}_a \mid k(a) = (\mathbf{t}_a, \mathbf{S}_a)\}$ for each a, and let G'_k be a function that maps each individual a to an assignment $G'_k(a)$ for $k(a)$ such that, for each $s = (\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}(k(a))$, $G'_k(a)(s)$ is the set of all f-subqueries that have a rooted prematch in $\mathcal{I}_s$. As $\mathbf{t}_s \in \mathfrak{T}(\mathbf{t}_s)$, by completeness of TQ there exists some $\mathbf{t}_s$-hit h of TQ such that $h \subseteq G'_k(a)(s)$. Thus we can define a function G_k that also maps each individual a to an assignment $G'_k(a)$ for $k(a)$, such that for each $s = (\mathbf{r}_s, \mathbf{t}_s) \in \mathbf{S}(k(a))$, $G_k(a)(s) \subseteq G'_k(a)(s)$ and $G_k(a)(s)$ is a $\mathbf{t}_s$-hit of TQ. Now consider an expansion $\mathcal{A}' = \mathsf{abox}(\mathcal{A}, \theta, k, G_k) \in \mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})$, and a model $\mathcal{I}'$ of $\mathcal{A}'$. As $\vec{a} \in \mathsf{ans}(q, \mathcal{A}')$, there exists a prematch π' for q in $\mathcal{I}'$ such that $\pi'(x_i) = {a_i}^{\mathcal{I}'}$ for each $0 \leq i \leq n$. For each $v \in \mathsf{V_I}(q)$ such that $\pi'(v) \neq a^{\mathcal{I}'}$ for every $a \in \mathsf{N_I}(\mathcal{A})$, there is some $F = (V, \Sigma) \in \mathbb{F}_q^{\mathcal{R}}$ such that $v \in V$ and $\mathcal{I}' \models F$. By construction of $\mathcal{A}'$, for each such F we have $F \in G_k(a)(s)$ for some $a \in \mathsf{N_I}(\mathcal{A})$ and some $s \in \mathbf{S}(k(a))$. As $G_k(a)(s) \subseteq G'_k(a)(s)$, we have $\mathcal{I}_s \models F$, that is, there is a rooted match π_s for F in $\mathcal{I}_s$. By taking the union of all these $\pi(s)$ and setting $\pi(v) = a^{\mathcal{I}}$ whenever $\pi'(v) = a^{\mathcal{I}'}$ for some individual a, we obtain a prematch for q in $\mathcal{I}$. As the prematch π preserves the bindings of answer variables to individuals, we have $\pi(x_i) = {a_i}^{\mathcal{I}}$ for each $0 \leq i \leq n$ and $\mathcal{I} \models q(\vec{a})$ as desired. □

Algorithm 8: computeAnswers

Input: KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, CQ q
Output: $\mathsf{ans}(q, \mathcal{K})$
begin
  Compute a $\mathsf{ABoxTypes}(\mathcal{K})$-complete knot set $\mathbb{K}$ for $\mathcal{T}$;
  Compute a complete tq-table TQ for q from $\mathbb{K}$;
  Compute the set $\mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})$ of $(\mathbb{K}, \mathsf{TQ})$-expansions of the ABox $\mathcal{A}$;
  Let $\mathsf{Ans} := \bigcap_{\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})} \mathsf{ans}(q, \mathcal{A}')$;
  return Ans
end

Figure 7.8: Computing the answers to the query q over the KB $\mathcal{K}$.

7.4 Computational Complexity

In this Section we give some complexity results. First we analyze the complexity of the query answering algorithm to obtain a 2ExpTime upper bound. Then in Section 7.4.2 we show that this is tight, as query entailment in $\mathcal{SH}$ is 2ExpTime-hard. Finally, in Section 7.4.3 we identify some syntactic restrictions on queries which ensure that the algorithm runs in single exponential time, obtaining a tight ExpTime bound for a large class of instances that includes CQ answering in $\mathcal{ALCH}$.

7.4.1 Upper Bound

We analyze now the complexity of our algorithm for CQ answering over $\mathcal{SH}$ knowledge bases. Recall that the method consists of three main steps: (1) computing a $\mathsf{ABoxTypes}(\mathcal{K})$-complete knot set for an input KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, (2) computing a complete tq-table TQ for the computed knot set $\mathbb{K}$ and an input CQ q, and (3) collecting answers to q by traversing TQ-expansions of $\mathcal{A}$. To show that the method answers queries in double exponential time, we first establish the following result.

Theorem 7.4.1 *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ be an $\mathcal{SH}$ KB, let q be a conjunctive query, and let $\mathbb{F}_q^{\mathcal{R}}$ be the set of f-subqueries of q. Then, given a tuple $\vec{a}$ of individuals, deciding whether $\vec{a} \in \mathsf{ans}(q, \mathcal{K})$ is feasible in time single exponential in $||\mathcal{K}|| + ||q|| + |\mathbb{F}_q^{\mathcal{R}}|$.*

Proof. We analyze the three steps of the procedure. To this end, we let $c = |Cl_{\mathsf{C}}(\mathcal{K})|$ and $t = |\mathsf{N_R}(\mathcal{K})|$, and start with the following observation:

($*$) For any set of types $\mathbf{T}$, a $\mathbf{T}$-complete knot set $\mathbb{K}$ for $\mathcal{K}$ can be obtained from $\mathbf{T}$ and $\mathbb{K}$ in time single exponential in $c + t$ via the algorithm computeKnots in Figure 7.2.

Indeed, the number of distinct types $\mathbf{t}$ for $\mathcal{K}$ is bounded by 2^c, while the number of distinct knots for $\mathcal{K}$ is bounded by $m = 2^c \cdot (2^t \cdot 2^c)^c = 2^{c+tc+c^2}$, and thus single exponential in $c + t$. Hence constructing the initial knot set in the first step of the algorithm is feasible in time single exponential in $c+t$. In the subsequent "fill-up" stage that closes the set under successor knots, the procedure may add only single exponentially many knots, while in the final "clean-up" stage each knot is removed at most once and never introduced again, and the whole procedure terminates in a number of steps that is at most single exponential in $c + t$.

Since both c and t are linear in $||\mathcal{K}||$, this shows that in step 1, an $\mathsf{ABoxTypes}(\mathcal{K})$-complete knot set $\mathbb{K}$ can be obtained in time single exponential in $||\mathcal{K}||$.

For the step 2, the Algorithm compute_TQ in Figure 7.7 computes a complete type-query table TQ for a set of knots $\mathbb{K}$ and q, in time polynomial in $|\mathbb{K}| + 2^{|\mathbb{F}_q^{\mathcal{R}}|}$, and hence single exponential in $m = ||\mathcal{K}|| + ||q|| + |\mathbb{F}_q^{\mathcal{R}}|$. This follows from the next observations:

i) At each iteration, by construction, the algorithm computes a d+1-complete tq-table TQ^{d+1} (via a d+1-complete kq-table KQ^{d+1}) from a d-complete tq-table TQ^{d}. Furthermore, the computed tables are 'complete' in the sense that *all* entailed disjunctive f-subqueries queries are included in the tables.[1] More precisely, for each d, we have $(\mathbf{t}, \mathrm{D}) \in \mathsf{TQ}^{d}$ iff $\mathbf{t} \models_d \mathrm{D}$. Since $\mathbf{t} \models_d \mathrm{D}$ implies $\mathbf{t} \models_{d+1} \mathrm{D}$, we get that the computation is monotonic, i.e., each computed TQ^{d+1} includes TQ^{d}.

ii) The largest possible tq-table for $\mathcal{K}$ and q is $2^{c} \cdot 2^{|\mathbb{F}_q^{\mathcal{R}}|}$, which is clearly of size single exponential in m. Hence, and given the monotonicity, the algorithm terminates within a number of steps that is single exponential in m.

iii) Each iteration, which consists of a call to KQ_from_TQ and then a call to TQ_from_KQ, takes time at most single exponential in m. For the call to KQ_from_TQ, all pairs $\mathbf{k}, \mathrm{D}$ of knots $\mathbf{k}$ and disjunctive f-subqueries D are traversed. The number of such pairs is single exponential in m. Furthermore, for each pair $(\mathbf{k}, \mathrm{D})$, whether it is included in the resulting table is decided by checking the conditions prescribed in Theorem 7.3.11, and this takes time at most single exponential in m. The call to TQ_from_KQ is analogous: there are at most single exponentially (in m) many pairs $(\mathbf{t}, \mathrm{D})$ of types and disjunctive f-subqueries, and to test the conditions in Theorem 7.3.15 for each of them is also feasible in single exponential time.

For the final step 3, which is based on Theorem 7.3.21, we note that the number of $(\mathbb{K}, \mathsf{TQ})$-expansions of $\mathcal{A}$ is the number of ways of choosing a completion θ for $\mathcal{K}$, choosing for each individual a of $\mathcal{A}$ a knot $k(a)$, and then choosing a set of f-subqueries for each leaf of the resulting forest, yielding an assignment $g_{k(a)}$; this is again bounded by a single exponential in m. Finally, checking whether $\vec{a} \in \mathsf{ans}(q, \mathcal{A}')$ for a given tuple $\vec{a}$ and an expansion $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})$ is also feasible in time single exponential in m. □

We can now easily infer the upper bound for the query answering problem in $\mathcal{SH}$. Indeed, for a given CQ q, the size of the set $\mathbb{F}_q^{\mathcal{R}}$ of f-forward subqueries of q is bounded by $2^{|\mathsf{VI}(q)|}$, which is single exponential in $||q||$. Therefore, by the above theorem, the procedure can be run in time double exponential in the size of the input KB $\mathcal{K}$ and the query q.

Proposition 7.4.2 *Given a KB $\mathcal{K}$ and a CQ q in $\mathcal{SH}$, and a tuple of individuals $\vec{a}$, deciding whether $\vec{a} \in \mathsf{ans}(q, \mathcal{K})$ is in* 2ExpTime.

We finally note that *enumerating* all the answer tuples $\vec{a}$ in $\mathsf{ans}(\mathcal{K}, q)$ for an $\mathcal{SH}$ KB $\mathcal{K}$ and a query q is also feasible in time double exponential in $||\mathcal{K}|| + ||q||$. This is because the number of candidate answer tuples $\vec{a}$ is only single exponential in $||\mathcal{K}||$.

We dedicate the next section to proving that our algorithm is worst-case optimal.

7.4.2 Lower Bound

In the previous section we have shown that our algorithm for query answering in $\mathcal{SH}$ runs in double exponential time. This bound is not new; it follows from the automata algorithm in

[1] In fact, only the subset minimal subqueries would need to be stored. However, the worst case complexity remains unchanged.

the first part of this thesis (Theorem 3.4.2), and from results in the literature, e.g., [CEO07, GHLS07, GHS07, OŠE08a]. What remained an upper question is whether the bound is tight, as the only 2ExpTime lower bound for query answering in DL was the one for $\mathcal{ALCI}$ [Lut07], which is not contained in $\mathcal{SH}$, and the best matching lower bound for $\mathcal{SH}$ was the ExpTime bound stemming from KB satisfiability.

Now we close the gap, and show that the above mentioned bounds are optimal: query entailment is 2ExpTime-hard in all extensions of $\mathcal{ALC}$ that support role inclusions and transitivity axioms. This identifies the combination of role inclusions and transitivity axioms as the second source of complexity that, like inverse roles, makes query answering provably harder than KB satisfiability checking.

We prove the following theorem:

Theorem 7.4.3 *CQ entailment in $\mathcal{SH}$ is* 2ExpTime*-complete.*

The 2-ExpTime-hardness result relies on a reduction from the word problem for *Alternating Turing machines* (ATMs) with exponential work space. We briefly recall here the definition of ATMs, but refer the reader to [CKS81] for background and details.

An ATM is given by a tuple $\mathcal{M} = (Q, \Sigma, q_0, \delta)$, where

- $Q = Q_\exists \uplus Q_\forall \uplus \{q_{\mathsf{acc}}\} \uplus \{q_{\mathsf{rej}}\}$, the set of *states*, consists of *existential states* in $Q_\exists$, *universal states* in $Q_\forall$, an *accepting state* q_{acc}, and a *rejecting state* q_{rej};
- Σ is the *alphabet* that additionally contains the *blank symbol* $\textvisiblespace$;
- $q_0 \in Q_\exists \cup Q_\forall$ is the *starting* state; and
- $\delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{+1, -1\}$ is the *transition relation*; for later use, we define $\delta(q, \sigma) = \{(q', \sigma', M) \mid (q, \sigma, q', \sigma', M) \in \delta\}$.

A *configuration* of $\mathcal{M}$ is a word wqw' with $w, w' \in \Sigma^*$ and $q \in Q$, whose intended meaning is that the one-side infinite tape contains the string ww' with only blanks behind it, that the machine is in state q, and that the head is on the symbol just after w. The *successor configurations* of a configuration wqw' are defined in terms of δ as usual; without loss of generality, we assume that $\mathcal{M}$ is well-behaved and never attempts to move left if the head is on the left-most position. Configurations of the form wqw' with $q \in \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$ are called *halting configurations* and have no successor configurations.

A *computation* of an ATM $\mathcal{M}$ on a word w is a sequence of configurations $K_0, K_1, \ldots$ such that $K_0 = q_0 w$ (the *initial configuration*) and K_{i+1} is a successor configuration of K_i, for all $i \geq 0$. For our concerns, we may assume that all computations are finite (on any input), and define acceptance only for this case.

A configuration wqw' is *accepting*, if either (a) $q = q_{\mathsf{acc}}$, or (b) $q \in Q_\exists$ and at least one of its successor configurations is accepting, or (c) $q \in Q_\forall$ and all of its successor configurations are accepting. The ATM $\mathcal{M}$ *accepts* the input w, if the *initial configuration* is accepting. The *word problem of* $\mathcal{M}$ is, given $\mathcal{M}$ and w, to decide whether $\mathcal{M}$ accepts w.

Note that an ATM with only existential states can be viewed as a standard non-deterministic Turing machine, which accepts a word iff there exists a sequence of successive configurations that starts in the *initial configuration*, with initial state q_0 and the input word w on the tape, and ends in an accepting state q_{acc}. For ATMs, these sequences become *trees* of configurations, where branching is caused by universal states (there is a successive configuration for each transition in $\delta(q, a)$ with $q \in Q_\forall$). Such a tree is a *computation tree*, and it is *accepting* if q_{acc} is reached on all paths.

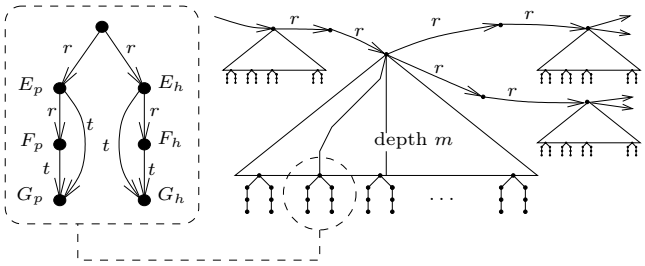

Figure 7.9: The structure of models.

It is known that 2ExpTime coincides with the class AExpSpace of all problems solvable by ATMs with exponentially bounded space. We use the following lemma.

Lemma 7.4.4 ([CKS81]) *There is an ATM $\mathcal{M}$ for which the word problem is* 2ExpTime*-hard and such that $\mathcal{M}$ works in exponential space, i.e., all configurations $w'qw''$ in computations on w fulfill $|w'w''| \leq 2^{|w|}$.*

By the above lemma, to show 2ExpTime-hardness of CQ entailment in $\mathcal{SH}$, it suffices to reduce the word problem for an exponentially space bounded ATM $\mathcal{M} = (Q, \Sigma, q_0, \delta)$ and an input word w. Lets fix such an $\mathcal{M}$ and w.

For each input w to $\mathcal{M}$, we define a KB $\mathcal{K}_{\mathcal{M},w}$ and a query $q_{\mathcal{M},w}$ such that $\mathcal{M}$ accepts w iff $\mathcal{K}_{\mathcal{M},w} \not\models q_{\mathcal{M},w}$. In fact, each canonical model $\mathcal{I}$ of $\mathcal{K}_{\mathcal{M},w}$ with $\mathcal{I} \not\models q_{\mathcal{M},w}$ will represent an accepting computation of $\mathcal{M}$ on w. Roughly, the model contains binary trees of depth $m := |w|$ (length of w) that represent configurations using their 2^m leaves to store the tape contents, which we call *configuration trees*. The roots of the configuration trees are in turn connected into simulate a *computation tree* as described above. This is illustrated in Figure 7.9; the initial configuration tree is existential and thus has a single successor configuration tree. Its (magnified) successor is universal and has two successor configuration trees.

To enforce this structure, we need some technical tricks. In particular, each configuration tree will represent *two* configurations: the *current configuration* K_h and the *previous* configuration K_p. We use $\mathcal{K}_{\mathcal{M},w}$ to ensure locally at each configuration tree that K_h is indeed a successor configuration of K_p. The query $q_{\mathcal{M},w}$ is then used to globally guarantee that the K_p value of each configuration tree is identical to the K_h value of the predecessor in the computation tree. We will call a computation tree *proper*, if it satisfies the latter condition.

We now give a precise definition of how configuration trees and computation trees are represented as a model. A single, non-transitive role r is used for the edges of computation trees and of configuration trees. Observe that, as shown in Figure 7.9, we use *two* r-edges between two consecutive configuration trees. We also use a transitive role t, to be explained later. The alphabet symbols Σ of $\mathcal{M}$ and the states Q are used as concept names. We also use the concept names from $\mathbf{B} := \{B_1, \ldots, B_m\}$ to encode addresses of tape cells in binary. For a node n of a forest model $\mathcal{I}$ and $i < 2^m$, we write $\mathsf{adr}^{\mathcal{I}}(n) = i$ if the truth values of $B_1^{\mathcal{I}}, \ldots, B_m^{\mathcal{I}}$ at n encode the number i. A tape cell with address i and content $a \in \Sigma$ is represented by a node n with $\mathsf{adr}^{\mathcal{I}}(n) = i$ that satisfies the concept name a. If the head is currently on the cell and $\mathcal{M}$'s state is q, then n also satisfies q; otherwise, n satisfies the concept name nil.

To later on ensure properness using the query, we use additional nodes and concept names. The latter are E_h, E_p, F_h, F_p, G_h, and G_p, used as markers; and the concept names from $\mathbf{Z} := \{Z_{a,q} \mid a \in \Sigma,\ q \in Q \cup \{nil\}\}$. The additional nodes are attached to the leaves of configuration trees, as indicated on the left-hand side of Figure 7.9 and detailed in the subsequent

definition. Intuitively, nodes labeled E_h store the current configuration and nodes labeled E_p the previous.

Definition 7.4.5 (i-cell) *Let $\mathcal{I}$ be an interpretation and $i < 2^m$. We call $n \in \Delta^{\mathcal{I}}$ an* i-cell *if the following hold:*

(a) n has r-successors n_p and n_h with $\mathsf{adr}^{\mathcal{I}}(n_p) = \mathsf{adr}^{\mathcal{I}}(n_h) = i$ that respectively satisfy E_p and E_h, and both satisfy exactly one $a \in \Sigma$ and exactly one $q \in Q \cup \{nil\}$.

(b) n_p (resp., n_h) has an r-successor n'_p (resp., n'_h) satisfying F_p (resp., F_h) and such that $\mathsf{adr}^{\mathcal{I}}(n'_p)$ (resp., $\mathsf{adr}^{\mathcal{I}}(n'_h)$) is the bit-wise complement of i. Furthermore, for all $a \in \Sigma$ and $q \in Q \cup \{nil\}$, we have:

(i) n_h satisfies $Z_{a,q}$ iff n_h does not satisfy both a and q;

(ii) n'_p satisfies $Z_{a,q}$ iff n_p does not satisfy both a and q;

(iii) n'_h and n_p satisfy $Z_{a,q}$;

(c) n'_p (resp., n'_h) has a t-successor n''_p satisfying G_p (resp., n''_h satisfying G_h) such that n''_p (resp., n''_h) is also a t-successor of n_p (resp., n_h).

We simply speak of a cell if i is unimportant. Note that the ability of $\mathcal{SH}$ to express (c) in Definition 7.4.5 via the axioms $r \sqsubseteq t$ and $\mathsf{trans}(t)$ is crucial for the reduction. The same condition can be expressed via a so-called *left identity* $r \circ t \sqsubseteq t$.

We now define (q, a, i)-*configuration nodes*, which are the roots of configuration trees. Intuitively, a configuration tree whose root is a (q, a, i) node represents a transition where the automaton writes the symbol a and moves to state q and position i. We also define interpretations that encode *computation trees* of $\mathcal{M}$. In what follows, we say a node n' is an r^m-successor of a node n, if n' is reachable from n by traveling m r-edges.

Definition 7.4.6 ((q, a, i)-configuration node, Computation tree) *Let $\mathcal{I}$ be an interpretation. We call $n \in \Delta^{\mathcal{I}}$ a* (q, a, i)-configuration node *if (1) it has an r^m-successor that is a j-cell (called* j-cell of n*), for each $j < 2^m$ and (2) the E_h-node of the i-cell of n satisfies q and a, and all other j-cells have nil in their E_h-nodes.*
We call $\mathcal{I}$ a computation tree *for w if $\mathcal{I}$ is tree-shaped and*

(I) the root ϵ of $\mathcal{I}$ has an r-successor n that is a $(q_0, a, 0)$-configuration node whose i-cells describe the initial configuration for input w;

(II) for each (q, a, i)-configuration node n, if $q \in Q_\exists$ (resp., $q \in Q_\forall$), then for some (resp., for each) tuple $(q', a', M) \in \delta(q, a)$ there exists an r^2-successor node n' that is an (q', a', i')-configuration node with $i' = i + M$, where $M \in \{-1, +1\}$ is the executed move. Furthermore, the E_h node of a i-cell of n' satisfies a', and, for all remaining j-cells c of n' with $j \neq i$, if the E_p node of c satisfies $a \in \Sigma$, then the E_h node of c also satisfies a (i.e., a i-cell has the new symbol written, while for the remaining cells, the E_h nodes in the resulting configuration tree carry over the symbols from their respective E_p nodes).

We call $\mathcal{I}$ accepting*, if $q = q_{\mathsf{acc}}$ in each (q, a, i)-configuration for which there is no successor configuration. Furthermore, $\mathcal{I}$ is* proper*, if for each pair of successive configuration nodes n, n' as in Definition 7.4.6.II and each $i < 2^m$, the i-cell of n has the same (q, a)-label in its E_h-node as the i-cell of n' in its E_p-node.*

It is not hard to see that there is a correspondence between accepting proper computation trees for w and accepting computations of $\mathcal{M}$ on w. The properness condition ensures that the *previous* configuration encoded in the E_p nodes of a configuration tree coincides with the *current* configuration encoded in the E_h nodes of the previous configuration tree. Then, due to the condition (II) in the above Definition 7.4.6, we get that each pair of successive configuration nodes encodes a correct transition of $\mathcal{M}$. On the other hand, given an accepting run of $\mathcal{M}$ on w, we can define an accepting computation tree.

Proposition 7.4.7 *$\mathcal{M}$ accepts w iff there exists an accepting proper computation tree for w.*

In the next section, we define an $\mathcal{SH}$ knowledge base capturing (proper and improper) computation trees, and in the subsequent section, we define a query for testing properness.

Building Computation Trees

Proposition 7.4.8 *Given w, we can build in polynomial time a KB $\mathcal{K}_{\mathcal{M},w}$ whose forest models are exactly the accepting computation trees for w.*

In the following, by constructing $\mathcal{K}_{\mathcal{M},w}$, we provide a proof of the above proposition. We define

$$\mathcal{K}_{\mathcal{M},w} = \langle \{I(a)\}, \mathcal{T}_w, \mathcal{R}_w \rangle$$

where a is an individual, I is a concept name (that identifies the initial node), and the TBox $\mathcal{T}_w$ and the RBox $\mathcal{R}_w$ contain the axioms described below.

Enforcing Configuration Nodes

Recall that configuration nodes are roots of binary trees of depth m whose leaves are i-cells corresponding to tape cells of $\mathcal{M}$. We next provide axioms enforcing conditions (1) and (2) in the definition of configurations nodes (see Definition 7.4.6). More precisely, nodes satisfying a special concept name R are forced to be configuration nodes. For technical reasons, the $m{+}1$ levels of a tree rooted at a configuration node are identified with concept names $L_0, \ldots, L_m$. For two concepts C and D, we use $C \to D$ as a shorthand for the concept $\neg C \sqcup D$. We introduce the following axioms, which generate a tree whose leaves cover the address range $0, \ldots, 2^m - 1$:

$$\begin{array}{rcll}
R & \sqsubseteq & L_0 & \\
L_i & \sqsubseteq & \exists r.(L_{i+1} \sqcap B_{i+1}) \sqcap \exists r.(L_{i+1} \sqcap \neg B_{i+1}) & \text{for all } 0 \leq i < m \\
L_i \sqcap B_j & \sqsubseteq & \forall r.(L_{i+1} \to B_j) & \text{for all } 0 < j \leq i < m \\
L_i \sqcap \neg B_j & \sqsubseteq & \forall r.(L_{i+1} \to \neg B_j) & \text{for all } 0 < j \leq i < m
\end{array}$$

Recall that the leaves of configuration trees must be i-cells, and hence the properties (a)-(c) in Definition 7.4.5 must be enforced. To enforce (a), we use the symbols from Σ, the states from Q and *nil* as concept names. We label such nodes with exactly one concept from Σ (the content of a cell c), and with exactly one concept from $Q^+ := Q \cup \{nil\}$; intuitively, the label $q \in Q$ means that the head of $\mathcal{M}$ is on the tape cell c and that $\mathcal{M}$ is in state q, while the label *nil* means that the head is not at position j. The above is realized as follows:

$$L_m \sqsubseteq \exists r.(E_p \sqcap E) \sqcap \exists r.(E_h \sqcap E)$$

$$E \sqsubseteq \bigsqcup_{a \in \Sigma} a \sqcap \bigsqcap_{a \neq a' \in \Sigma} \neg(a \sqcap a')$$

$$E \sqsubseteq \bigsqcup_{q \in Q^+} q \sqcap \bigsqcap_{q \neq q' \in Q^+} \neg(q \sqcap q').$$

To enforce the structures as prescribed in the remaining properties (b)-(c), we use the following axioms:

1. The existence of the required nodes is enforced via the following axioms:

$$\begin{array}{rcl} E_p & \sqsubseteq & \exists r.(F_p \sqcap \exists t.G_p) \\ E_h & \sqsubseteq & \exists r.(F_h \sqcap \exists t.G_h) \end{array}$$

2. The address for E_p and E_h nodes, and its bitwise complement in F_p and F_h nodes is obtained by adding for each $1 \leq i \leq m$ the following axioms:

$$\begin{array}{rcl} L_m \sqcap B_i & \sqsubseteq & \forall r.B_i \\ L_m \sqcap \neg B_i & \sqsubseteq & \forall r.\neg B_i \\ E \sqcap B_i & \sqsubseteq & \forall r.\neg B_i \\ E \sqcap \neg B_i & \sqsubseteq & \forall r.B_i \end{array}$$

3. The conditions (b.i)-(b.iii) are enforced by the following axioms: for all $a \in \Sigma, q \in Q^+$,

$$\begin{array}{rcl} E_h & \sqsubseteq & (a \sqcap q) \leftrightarrow \neg Z_{a,q} \\ E_p & \sqsubseteq & (a \sqcap q) \rightarrow \forall r.(\neg Z_{a,q} \sqcap \bigsqcap_{(a,q)\neq(a',q')} Z_{a',q'}) \\ E_h & \sqsubseteq & \forall r.Z_{a,q} \\ E_p & \sqsubseteq & Z_{a,q} \end{array}$$

4. Finally, to enforce (c), we add $r \sqsubseteq t$ and $\mathsf{trans}(t)$.

It remains to ensure that each node n that satisfies R also satisfies that for exactly one address $i < 2^m$, the i-cell of n satisfies some $q \in Q$ and all j-cells, $j \neq i$, satisfy *nil* (cf. (2) in Definition 7.4.6). To achieve this, we use a concept name H (for the head position) and make sure that it occurs in the label of an L_m node iff its address is i, and that only an E_h successor of such an L_m node contains labels from Q.

$$\begin{array}{rcll} L_0 & \sqsubseteq & H & \\ (L_i \sqcap H) & \sqsubseteq & (\forall r.((L_{i+1} \sqcap B_i) \rightarrow H) \sqcap \forall r.((L_{i+1} \sqcap \neg B_i) \rightarrow \neg H)) & \\ & & \sqcup\ (\forall r.((L_{i+1} \sqcap \neg B_i) \rightarrow H) \sqcap \forall r.((L_{i+1} \sqcap B_i) \rightarrow \neg H)) & \text{for all } 0 \leq i < m \\ (L_i \sqcap \neg H) & \sqsubseteq & (\forall r.(L_{i+1} \rightarrow \neg H) & \text{for all } 1 \leq i < m \\ L_m \sqcap H & \sqsubseteq & \forall r.(E_h \rightarrow \bigsqcup_{q\in Q} q) & \\ L_m \sqcap \neg H & \sqsubseteq & \forall r.(E_h \rightarrow \mathit{nil}) & \end{array}$$

We remark here that for configurations represented by E_p nodes we omit here adding similar axioms. Indeed, the query $q_{\mathcal{M},w}$ that we construct will, as a byproduct, also check whether a state $q \in Q$ is stored at exactly one address for E_p nodes.

Enforcing Computation Trees

To generate computation trees, we add axioms ensuring that tree-shaped models of $\mathcal{K}_{\mathcal{M},w}$ satisfy conditions (I) and (II) in Definition 7.4.6. In the following, we use $\forall r^i.C$ to denote the i-fold nesting $\forall r.\cdots\forall r.C$. In particular, $\forall r^0.C$ is C.

The initial configuration as described in (I) is ensured as follows. Let $w = a_0 \cdots a_n$ be the initial word. We will additionally keep track of the position of the R/W head of $\mathcal{M}$. To this

end, we use concept names $Q'_1, \ldots, Q'_m$ and $Q_1, \ldots, Q_m$ for the previous position and the current position resulting due to a transition. We add the following:

$$\begin{array}{lcll} I & \sqsubseteq & \exists r.R & \\ I & \sqsubseteq & \forall r^{m+1}.(\mathsf{pos} = i \rightarrow \forall r.(E_h \rightarrow a_i)) & \text{for all } i < n \\ I & \sqsubseteq & \forall r^{m+1}.(\mathsf{pos} = 0 \rightarrow \forall r.(E_h \rightarrow q_0)) & \\ I & \sqsubseteq & \forall r^{m+1}.(\mathsf{pos} \geq n \rightarrow \forall r.(E_h \rightarrow \text{␣})) & \\ I & \sqsubseteq & \forall r.\neg Q_i & \text{for all } 1 \leq i \leq m \end{array}$$

where $(\mathsf{pos} = i)$ and $(\mathsf{pos} \geq n)$ are the obvious (Boolean) concepts expressing that the value of the address $B_1, \ldots, B_m$ equals i and is at least n, respectively (recall that ␣ is the blank symbol).

We turn to the condition (II) in Definition 7.4.6. In detail, to represent that a configuration node n' is a successor of a configuration node n upon taking the transition $(q', a', M) \in \delta(q, a)$, we label n' with the concept name $T_{q',a',M}$ and we connect n to n' via two consecutive r arcs. Furthermore, if q is existential, we enforce that some n' exists with suitable label $T_{q',a',M}$, and if q is universal, we enforce that for each $(q', a', M) \in \delta(q, a)$ some n' exists with label $T_{q',a',M}$; we exploit that the state q and the symbol a are stored in an E_h-node of n, for one unique address. We also ensure that the address of R/W head is copied to the follow-up configuration nodes.

$$\begin{array}{rcll} R \sqcap \exists r^{m+1}.(E_h \sqcap q \sqcap a) & \sqsubseteq & \bigsqcup_{(q',a',M)\in\delta(q,a)} \exists r^2.(R \sqcap T_{q',a',M}) & \text{for all } q \in Q_\exists, a \in \Sigma, \\ R \sqcap \exists r^{m+1}.(E_h \sqcap q \sqcap a) & \sqsubseteq & \bigsqcap_{(q',a',M)\in\delta(q,a)} \exists r^2.(R \sqcap T_{q',a',M}) & \text{for all } q \in Q_\forall, a \in \Sigma. \\ Q_i & \sqsubseteq & \forall r^2 Q'_i & \text{for all } 0 < i < m \\ \neg Q_i & \sqsubseteq & \forall r^2 \neg Q'_i & \text{for all } 0 < i < m \end{array}$$

Next we provide axioms that define the position of the R/W head resulting by transition. It is obtained by applying addition or subtraction to the address encoded by Q'_i concepts. We use $INV_1, \ldots, INV_m$ to decide on the bits that need to be inverted:

$$\begin{array}{rcll} T_{q,a,+1} & \sqsubseteq & PLUS & \text{for all} q \in Q, a \in \Sigma, \\ T_{q,a,-1} & \sqsubseteq & MINUS & \text{for all} q \in Q, a \in \Sigma, \\ R \sqcap PLUS & \sqsubseteq & INV_m & \\ R \sqcap Q'_i \sqcap INV_i \sqcap PLUS & \sqsubseteq & INV_{i-1} & \text{for all } 1 < i \leq m \\ R \sqcap PLUS \sqcap (\neg Q'_i \sqcup \neg INV_i) & \sqsubseteq & \neg INV_{i-1} & \text{for all } 1 < i \leq m \\ R \sqcap MINUS & \sqsubseteq & INV_m & \\ R \sqcap \neg Q'_i \sqcap INV_i \sqcap MINUS & \sqsubseteq & INV_{i-1} & \text{for all } 1 < i \leq m \\ R \sqcap MINUS \sqcap (Q'_i \sqcup \neg INV_i) & \sqsubseteq & \neg INV_{i-1} & \text{for all } 1 < i \leq m \\ Q'_i \sqcap INV_i & \sqsubseteq & \neg Q_i & \text{for all } 1 \leq i \leq m \\ Q'_i \sqcap \neg INV_i & \sqsubseteq & Q_i & \text{for all } 1 \leq i \leq m \\ \neg Q'_i \sqcap INV_i & \sqsubseteq & Q_i & \text{for all } 1 \leq i \leq m \\ \neg Q'_i \sqcap \neg INV_i & \sqsubseteq & \neg Q_i & \text{for all } 1 \leq i \leq m \end{array}$$

We also propagate the two addresses to the leaves by adding, for each $0 < j \leq m$, the following axioms:

$$\begin{array}{rcll} L_i \sqcap Q_j & \sqsubseteq & \forall r.(L_{i+1} \to Q_j) & \text{for all } 0 \leq i < m \\ L_i \sqcap \neg Q_j & \sqsubseteq & \forall r.(L_{i+1} \to \neg Q_j) & \text{for all } 0 \leq i < m \\ L_i \sqcap Q'_j & \sqsubseteq & \forall r.(L_{i+1} \to Q'_j) & \text{for all } 0 \leq i < m \\ L_i \sqcap \neg Q'_j & \sqsubseteq & \forall r.(L_{i+1} \to \neg Q'_j) & \text{for all } 0 \leq i < m. \end{array}$$

To enforce the second part of condition (II), we make sure that for a configuration node n satisfying $T_{q',a',M}$, the symbol in the previous position of the R/W head is changed to a', while the symbols in other positions are transferred from E_p nodes to E_h nodes. The first part is done by adding, for all $q' \in Q$, $a' \in \Sigma$, $M \in \{+1, -1\}$ the axioms:

$$T_{q',a',M} \sqsubseteq \forall r^m.(L_m \to T_{q',a',M}),$$

$$L_m \sqcap T_{q',a',M} \sqcap \vec{Q}' = \vec{B} \sqsubseteq \forall r.(E_h \to a'),$$

$$L_m \sqcap T_{q',a',M} \sqcap \vec{Q} = \vec{B} \sqsubseteq \forall r.(E_h \to q'),$$

where $\vec{Q} = \vec{B}$ stands for $\bigsqcap_{0<i\leq m}((Q_i \sqcap B_i) \sqcup (\neg Q_i \sqcap \neg B_i))$ and $\vec{Q}' = \vec{B}$ for $\bigsqcap_{0<i\leq m}((Q'_i \sqcap B_i) \sqcup (\neg Q'_i \sqcap \neg B_i))$.

All remaining tape cells do not change:

$$L_m \sqcap \exists r.(E_p \sqcap a \sqcap nil) \sqsubseteq \forall r.(E_h \to a) \quad \text{for all } a \in \Sigma.$$

This concludes the definition of $\mathcal{T}_w$ and $\mathcal{R}_w$, and hence of the KB $\mathcal{K}_{\mathcal{M},w}$. By construction, all forest-shaped models of $\mathcal{K}_{\mathcal{M},w}$ satisfy the conditions in Definition 7.4.6, and hence are computation trees.

Testing Properness of Computation Trees

As already mentioned, we use the query $q_{\mathcal{M},w}$ to test whether the tree is proper. More precisely, $q_{\mathcal{M},w}$ should have a match in a computation tree iff that tree is *not* proper. We start with a characterization of (im)properness in terms of the auxiliary concept names from above. In the following, we say that two cells n and n' are *A-conspicuous*, where A is a concept name, if

(†) A is true at the E_h-node of n and the E_p-node of n', or

(‡) A is true at the F_h-node of n and the F_p-node of n'.

Proposition 7.4.9 *A computation tree $\mathcal{I}$ is not proper iff (⋆) there exist cells n and n' in successive configurations of $\mathcal{I}$ K such that n and n' are A-conspicuous for all $A \in \mathbf{B} \cup \mathbf{Z}$.*

Proof. The proposition holds due to the way auxiliary labels are defined. First note that if n, n' are cells of two successive configurations in $\mathcal{I}$, then the conditions imposed on $\mathsf{adr}^{\mathcal{I}}(\cdot)$ in Definition 7.4.5 imply that $\mathsf{adr}^{\mathcal{I}}(n) = \mathsf{adr}^{\mathcal{I}}(n')$ iff for all $A \in \mathbf{B}$, n and n' are A-conspicuous; this is because bit-wise complement is used for the addresses of F_p- and F_h-nodes.
($\Rightarrow$) Suppose that $\mathcal{I}$ is not proper. Then there exist two i-cells n and n' of two successive configurations of $\mathcal{I}$ such that the E_h-node of n and the E_p-node of n' satisfy different pairs (q, a) and (q', a'). As $\mathsf{adr}^{\mathcal{I}}(n) = \mathsf{adr}^{\mathcal{I}}(n')$, n and n' are A-conspicuous for all $A \in \mathbf{B}$. By (b.iii) of Definition 7.4.5, $Z_{q,a}$ is true at the F_h-node of n; by (b.ii) and since $(q, a) \neq (q', a')$, $Z_{q,a}$ is also true at the F_p-node of n' (recall that $Z_{q',a'}$ is false for at most one pair q', a').

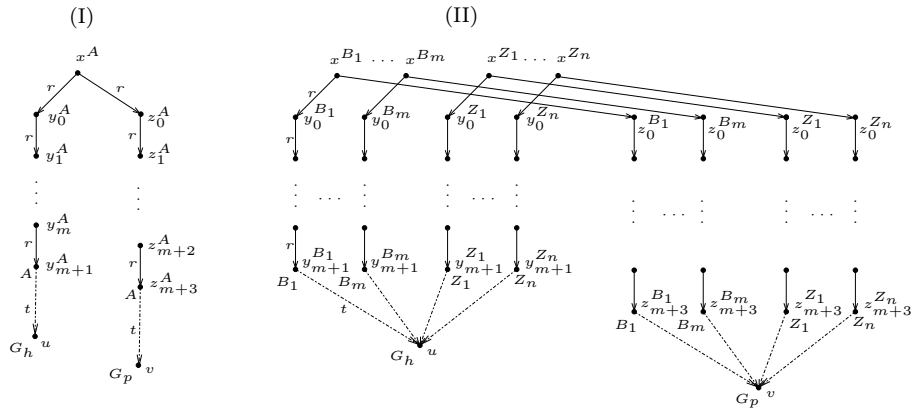

Figure 7.10: The basic query $q(A, u, v)$ and the final query $q_{\mathcal{M},w}$.

We can argue similarly that $z_{a',q'}$ is true at the E_h-node of n and the E_p-node of n'. For $(q'', a'') \notin \{(q, a), (q', a')\}$, $Z_{a'',q''}$ holds at the E_h-, E_p-, F_h-, and F_p-nodes of both n and n'. In summary, n and n' are A-conspicuous for all $A \in \mathbf{Z}$. Hence, $(\star)$ is true.
$(\Leftarrow)$ To show this, we prove the contrapositive. Suppose that $\mathcal{I}$ is proper and let n and n' be any cells of two successive configurations in $\mathcal{I}$. If n and n' are not A-conspicuous for some $A \in \mathbf{B}$ then $(\star)$ is false; otherwise, $\mathsf{adr}^{\mathcal{I}}(n) = \mathsf{adr}^{\mathcal{I}}(n')$ holds, and as $\mathcal{I}$ is proper, the E_h-node of n and the E_p-node of n' satisfy the same $q \in Q$ and $a \in \Sigma$. By (b.i) of Definition 7.4.5, $Z_{a,q}$ is false at the E_h-node of n; by (b.ii), $Z_{a,q}$ is false at the F_p-node of n'. Hence, n and n' are not $Z_{a,q}$-conspicuous, which means that also in this case $(\star)$ is false. □

It thus remains to find a query $q_{\mathcal{M},w}$ that has a match iff $(\star)$ is satisfied. The structure of $q_{\mathcal{M},w}$ is displayed in Figure 7.10(II).

We obtain $q_{\mathcal{M},w}$ by taking, for each $A \in \mathbf{B} \cup \mathbf{Z}$, a copy of the basic query $q(A, u, v)$ in Figure 7.10(I) such that the different copies share only the variables u and v, and then taking the union. Intuitively, $q(A, u, v)$ deals with A-conspicuousness, and the shared variables u, v ensure that the different component queries speak about the same cells n, n'. In more detail, let n, n' be cells of two successive configurations that are A-conspicuous for all $A \in \mathbf{B} \cup \mathbf{Z}$. We can find a match for $q_{\mathcal{M},w}$ as follows: start with matching u on the G_h-node of n and v on the G_p-node of n'. Now take an $A \in \mathbf{B} \cup \mathbf{Z}$. If (†) applies, then match y^A_{m+1} on the E_h-node of n and z^A_{m+1} on the E_p-node of n'; if (‡) applies, then match y^A_{m+1} on the F_h-node of n and z^A_{m+1} on the F_p-node of n'. The matches of all other variables are now uniquely determined by the (non-transitive) role edges in the query. In particular, the lengths of the role chains in the query ensure that x^A will be matched to the root of the configuration node in which n occurs in case (‡) and to the predecessor of this root node in case (†). Observe that the paths labeled with z-variables are exactly two steps longer than those labeled with y-variables, and thus the query only relates n and n' if they belong to successor configurations.

In summary, we can show that

Proposition 7.4.10 *A computation tree $\mathcal{I}$ is proper iff $\mathcal{I} \not\models q_{\mathcal{M},w}$.*

Together with Propositions 7.4.6 and 7.4.8, this yields the desired reduction, establishing the lower bound required for Theorem 7.4.3.

7.4.3 Improving the Upper Bound

We discuss here some syntactic restrictions to obtain classes of CQs for which query answering is feasible in single exponential time. To this end, it is sufficient to ensure that a query can be decomposed into only polynomially many f-subqueries; the complexity drop follows then from Theorem 7.4.1.

We assume an arbitrary $\mathcal{SH}$ KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, and define next some notions to measure the structural complexity of a query (w.r.t. $\mathcal{K}$). In this section, we call a role p *simple w.r.t.* $\mathcal{K}$ if it is simple w.r.t. $\mathcal{R}$.

Definition 7.4.11 (fork degree, non-trivial forks) *For any query q, we define*

$$\mathsf{R}^q_+(v) = \{v_n \mid p_1(v, v_1) \in q, p_2(v_1, v_2) \in q, \ldots, p_n(v_{n-1}, v_n) \in q \wedge n \geq 1\},$$

i.e., $\mathsf{R}^q_+(v)$ denotes the set of variables reachable from v in the query graph of q in one or more steps. Furthermore, let $\mathsf{R}^q_*(v) = \{v\} \cup \mathsf{R}^q_+(v)$ and, for any set V of variables, let $\mathsf{R}^q_+(V) = \bigcup_{v \in V} \mathsf{R}^q_+(v)$ and $\mathsf{R}^q_*(V) = \bigcup_{v \in V} \mathsf{R}^q_*(v)$.

A set $V \subseteq \mathsf{VI}(q)$ is called a fork set *(of q) if the following are true:*

(a) for each $v \neq v' \in V$, it holds that $v' \notin \mathsf{R}^q_+(v)$ and $v \notin \mathsf{R}^q_+(v')$, i.e., no variable in V reaches another variable in V; and

(b) the set $\mathsf{R}^q_*(V)$, i.e., the closure of V under reachable variables, induces a subquery of q whose graph is connected and acyclic.

Note that (b) implies that no variable $v \in V$ reaches a cycle in q, i.e., there is no $v \in \mathsf{R}^q_*(V)$ such that $v' \in \mathsf{R}^q_+(v')$.

Then the fork degree of q, *denoted $\mathsf{fd}(q)$, is defined as the size of the largest fork set of q.*

The number of non-trivial forks *in a query q is the number of variables $v \in \mathsf{VI}(q)$ satisfying the following:*

(a) there exist two atoms $p(v_1, v) \in q$ and $p'(v'_1, v) \in q$ such that $v_1 \neq v'_1$ and p' is not simple w.r.t. $\mathcal{K}$,

(b) v does not reach a cycle in q, i.e., there exists no $v' \in \mathsf{R}^q_*(v)$ such that $v' \in \mathsf{R}^q_+(v')$.

Example 7.4.12 *For the query q in Example 7.3.3, $V = \{v_1, v_5\}$ is the only fork set of q which contains more than one variable; any other such candidate fork set violates either condition (a) or condition (b). Hence, $\mathsf{fd}(q) = 2$.*

Note that for fork sets V of size larger than one, each variable $v \in V$ has a common successor with some other variable $v' \in V$ (in the previous example, v_1 has a common successor with v_5). Intuitively, the fork degree of q tells us how many "incomparable" variables we can pick so that they induce a connected acyclic subquery of q. Given this, we can formulate a syntactic condition ensuring lower complexity of query answering.

Theorem 7.4.13 *If $\mathcal{Q}$ is a class of CQs such that for any $q \in \mathcal{Q}$:*

(a) the number of non-trivial forks in q is bounded by some constant c,

(b) $\mathsf{fd}(q)$ is bounded by c, and

(c) for each pair $v, v' \in \mathsf{VI}(q)$, $|\{p' \mid p' \sqsubseteq^{\mathcal{R}} p, p(v, v') \in q, \text{ and } \mathsf{trans}(p) \in \mathcal{R}\}| \leq c$,

then the set $\mathbb{F}_q^{\mathcal{R}}$ of f-subqueries of q is polynomial in $||q||$. Hence, answering a query $q \in \mathcal{Q}$ over the KB $\mathcal{K}$ is feasible in single exponential time in $||q|| + ||\mathcal{K}||$.

Proof. Consider an arbitrary f-subquery (V, Σ) of $q \in \mathcal{Q}$. By definition, V induces a connected acyclic subquery of q. Let M_V be the set of all $v \in V$ that have no predecessor in V, i.e., for which there exists no $p(v', v) \in q$ with $v' \in V$. Clearly, M_V is a fork set of q (see Definition 7.4.11), and hence by (b) $|M_V| \leq c$. We get that the number of different M_V over all possible V is bounded by $|\mathsf{Vl}(q)|^c$, and is polynomial in $||q||$. It is not hard to see that given two f-subqueries (V_1, Σ_1) and (V_2, Σ_2) of q with $M_{V_1} = M_{V_2}$ we also have $V_1 = V_2$. Hence, the number of distinct V that can be chosen is bounded by $|\mathsf{Vl}(q)|^c$, and is polynomial in $||q||$.

Now we consider the number of possibilities to choose Σ. Observe that Σ is defined only for the free variables of V. The number of variables $v' \in V$ that are in free in V is bounded by $2{\cdot}c$ because of the bounded number of non-trivial forks in q (condition (a)). For each such v', we have $|\{p' \sqsubseteq^{\mathcal{R}} p,\ p \in \mathsf{back}(V, v'),\ \mathsf{trans}(p') \in \mathcal{R}\}| \leq c$ by condition (c). Hence, the total possible choices for Σ are bounded by $2{\cdot}c \cdot 2^c$, which is polynomial in $||q||$.

The second part of the claim follows then from Theorem 7.4.1. □

When computing the fork degree of a query, we do not ignore trivial forks $p(v_1, v_2), p'(v_1', v_2)$ where p, p' are simple, and treat v_1 and v_1' as 'incomparable' variables that may induce different fork sets. This is not really necessary, as such forks do not increase the number of distinct f-subqueries: (a.ii) of Definition 7.3.2 enforces that either both v_1 and v_1' or neither v_1 nor v_1' belong to a f-subquery. Hence we can safely eliminate these trivial forks, and look at the fork degree of the resulting query. We can achieve this using the same kind of *fork elimination* we used for $\mathcal{ALCH}$ queries in Chapter 6. The next definition is very close to Definition 6.4.16, but now we make a distinction between simple and non-simple roles.

Definition 7.4.14 (Fork elimination) *For a CQ q, a* fork elimination *of q is a query obtained from q by exhaustively applying the following rule: if the query contains atoms $p(v_1, v_2)$ and $p'(v_1', v_2)$ where $v_1 \neq v_1'$ and p, p' are simple w.r.t. $\mathcal{K}$, then replace every occurrence of v_1 with v_1'. By $\mathsf{FE}(q)$ we denote an arbitrary fork elimination of q.*

Note that fork eliminations of q coincide up to a renaming of variables. We can now state the slightly relaxed conditions which diminish the impact of trivial forks to the fork degree.

Theorem 7.4.15 *Let $\mathcal{Q}$ be a class of CQs such that for any $q \in \mathcal{Q}$:*

(a) the number of non-trivial forks in $\mathsf{FE}(q)$ is bounded by some constant c,

(b) $\mathsf{fd}(\mathsf{FE}(q))$ is bounded by c, and

(c) for each pair $v, v' \in \mathsf{Vl}(q)$, $|\{\, p' \mid p' \sqsubseteq^{\mathcal{R}} p, p(v, v') \in q,\ \text{and } \mathsf{trans}(p) \in \mathcal{R}\}| \leq c$,

Then deciding $\vec{a} \in \mathsf{ans}(q, \mathcal{K})$ for a given $q \in \mathcal{Q}$ and tuple $\vec{a}$, is feasible in single exponential time in $||q|| + ||\mathcal{K}||$.

Proof. By Theorem 7.4.1, it suffices to show that the number of f-subqueries of q is polynomial in $||q||$. As argued in the proof of Theorem 7.4.13, the conditions (a) and (c) ensure that the number of ways to choose Σ for an f-subquery (V, Σ) of q is polynomial in $||q||$.

The number of choices for V is also polynomial in $||q||$. To see this, for any conjunctive query q' define $\mathcal{V}_{q'} = \{V \mid \langle V, \Sigma\rangle \text{ is an f-subquery of } q'\}$, and observe that (i) $|\mathcal{V}_{\mathsf{FE}(q)}|$ is polynomial in $||q||$, and (ii) $|\mathcal{V}_q| = |\mathcal{V}_{\mathsf{FE}(q)}|$. The former follows from (b) and Theorem 7.4.13 (as $|\mathsf{FE}(q)| \leq ||q||$). For the latter, note that a rewrite step in fork elimination preserves the number of variable sets

satisfying (a.ii) of Definition 7.3.2. More precisely, if q'' is obtained from q' by the rewrite rule in Definition 7.4.14, then $|\mathcal{V}_{q'}| = |\mathcal{V}_{q''}|$ (as easily seen by establishing a bijection from $\mathcal{V}_{q'}$ to $\mathcal{V}_{q''}$). □

Based on the above theorem we can obtain further query classes of lower computational complexity. In particular, the conditions (a-c) of Theorem 7.4.15 are satisfied for the class of queries that allow for simple roles only. Indeed, given such a query q, (a) and (c) are trivially satisfied. For (b), observe that for any variable v of $\mathsf{FE}(q)$ there are two possibilities. The variable v occurs in a cycle in the query graph of $\mathsf{FE}(q)$, and hence v is not included in any fork set and does not contribute to the fork degree. Alternatively, v and its successors induce a subquery of $\mathsf{FE}(q)$ whose graph is a tree. In this case, $\{x\}$ is the single fork set where v may occur.

Importantly, the above can be generalized to the case where only a bounded number of atoms $p(x, y)$, where p is non-simple, occur in a query. To show this, we use a more refined query complexity measure.

Definition 7.4.16 (counting transitive arcs) *For any query q, let $t(q)$ denote the number of all pairs of variables $v, v' \in \mathsf{Vl}(q)$ such that:*

(1) q contains some atom $p(v, v')$ where p is not simple w.r.t. $\mathcal{K}$,

(2) q contains no atom $p'(v, v')$ where p' is simple w.r.t. $\mathcal{K}$,

(3) v' does not reach a cycle in the query graph of q, i.e., no $v'' \in \mathsf{R}^q_*(v')$ exists such that $v'' \in \mathsf{R}^q_+(v'')$, and

(4) some variable $u \in \mathsf{R}^q_*(v')$ has more than one predecessor in q, i.e., $|\{u' \mid R(u', u) \in q\}| > 1$.

Note that (3) eliminates pairs of variables that do not matter for the fork degree due to cyclicity (see (b) in Definition 7.4.11). Condition (4) refines this, by further eliminating cases where $\mathsf{R}^q_*(v')$ induces a query subgraph of q that is tree-shaped and disconnected to the remainder of the query graph of q. We remark that $t(q)$ can be easily computed.

Proposition 7.4.17 *For each CQ q it holds that $\mathsf{fd}(\mathsf{FE}(q)) \leq t(\mathsf{FE}(q)) + 1 \leq t(q) + 1$.*

Proof. It is easy to verify that $t(\mathsf{FE}(q)) \leq t(q)$: a fork elimination step preserves cycles in $\mathsf{R}^q_*(v)$ for every variable v (but might introduce new ones). Furthermore, it can not increase the number of different predecessors of v; hence, items (3) and (4) of Definition 7.4.16 hold for $\mathsf{FE}(q)$ if they hold for q. The same holds for (1) and (2).

Let q' result from $\mathsf{FE}(q)$ by removing all atoms $p(v, v')$ where v reaches a cycle in the query graph of $\mathsf{FE}(q)$, i.e., some $v'' \in \mathsf{R}^{\mathsf{FE}(q)}_*(v)$ exists such that $v'' \in \mathsf{R}^{\mathsf{FE}(q)}_+(v'')$. Note that $\mathsf{fd}(q') = \mathsf{fd}(\mathsf{FE}(q))$. Without loss of generality, we assume that q' contains a single unary atom $A(v)$ for each variable $v \in \mathsf{Vl}(q')$.

Since q' is acyclic, we can construct q' along a topological sort $v_1 < v_2 < \cdots < v_n$ of its variables, i.e., $p(v_j, v_i) \in q'$ implies $j < i$, starting from v_1 with $A(v_1)$ and adding variable v_i with $A(v_i)$ and all atoms $p(v_j, v_i)$ for $i > 1$.

We show now by induction on $n \geq 1$ that

$$\mathsf{fd}(q') \leq t(q') + 1. \tag{7.1}$$

Base case. Here $q' = \{A(v_1)\}$ and $\mathsf{fd}(q') = 1$; thus (7.1) holds for q'.

Induction Step. Suppose we join a variable v_n with $A(v_n)$ and atoms $p_1(v_{n_1}, v_n), \ldots, p_n(v_{n_{k_n}}, v_n)$ to q' of the assumed form, which yields a query q'' of similar form, and let $A = \{v_{n_1}, \ldots, v_{n_{k_n}}\}$. We consider two cases.

Case 1. Suppose first that $|A| \leq 1$, i.e., v_n is connected to at most one variable in q'. Then, clearly $t(q'') = t(q')$ (condition 4 is violated for the pair v_{n_1}, v_n) and $\mathsf{fd}(q'') = \mathsf{fd}(q')$, which means that (7.1) holds for q''.

Case 2. Suppose that $|A| = m > 1$, i.e., v_n is connected to multiple distinct variables $v_{n_1}, \ldots, v_{n_m}$ in q''. In this case,

$$t(q') + (m-1) \leq t(q'') \tag{7.2}$$

holds, as each pair v, v' in $\mathsf{Vl}(q')$ that satisfies the conditions 1-4 for $t(q')$ satisfies them for $t(q'')$, and at least $m-1$ pairs $v_{n,i}, v_n$ satisfy them for $t(q'')$, given that fork elimination is not applicable to any $p(v_{n_i}, v_n)$, $p'(v_{n_j}, v_n)$.

Let $V \subseteq \mathsf{Vl}(q'')$ be a fork set for q'' such that $|V| = \mathsf{fd}(q'')$. Let $\mathcal{V} = \{V_1, \ldots, V_k\}$ be the set of all maximal $V_i \subseteq V$ (w.r.t. $\subseteq$) that are fork sets for q'. Then for $V_i \neq V_j \in \mathcal{V}$, the sets $\mathsf{R}_*^{q'}(V_i)$ and $\mathsf{R}_*^{q'}(V_j)$ are disjoint and $\bigcup \mathcal{V} = V$. Furthermore, $k \leq m$ must hold: as $V_i \neq V_j \in \mathcal{V}$ must be connected in q'' via v_n, we have $\mathsf{R}_*^{q''}(V_i) \cap \mathsf{R}_*^{q''}(V_j) = \{v_n\}$. On the other hand, $\mathsf{R}_*^{q''}(V_i) \cap A \neq \emptyset$ and $\mathsf{R}_*^{q''}(V_j) \cap A \neq \emptyset$. Hence, at most m different V_i exist.

Now consider for the query $q_i' \subseteq q'$ that contains all atoms from q_i on the variables $\mathsf{R}_*^{q'}(V_i)$. Then V_i is a fork set for q_i'; hence by the induction hypothesis for q_i',

$$|V_i| \leq \mathsf{fd}(q_i') \leq t(q_i') + 1.$$

As each pair $v_i, v_j \in \mathsf{Vl}(q_i')$ satisfies conditions 1-4 for $t(q')$ if it satisfies them for $t(q_i')$, and since the V_i are pairwise disjoint and the queries q_i' are pairwise disconnected, we conclude

$$|V| = \sum_{i=1}^{k} |V_i| \leq \sum_{i=1}^{k} (t(q_i') + 1) \leq t(q') + k \leq t(q') + m.$$

Thus using (7.2),

$$\mathsf{fd}(q'') = |V| \leq t(q') + m \leq t(q'') + 1,$$

and the induction statement (7.1) holds for q''; this completes the proof. □

We also observe that for every query q, the number of non-trivial forks in $\mathsf{FE}(q)$ is $\leq t(q)$. Indeed, due to the rewrite rule, for each variable v of $\mathsf{FE}(q)$ there exists at most one variable v_1 such that $\{p \mid p(v_1, v) \in q\}$ contains a simple role. In other words, for each other variable $v_1 \neq v_1'$, all roles in $\{p \mid p(v_1', v) \in q\}$ must be non-simple. This means that if v is a variable counted in as a non-trivial fork (i.e., satisfies the conditions in Definition 7.4.11), then for v there exists at least one v' such that the pair v', v is counted in $t(q)$ (i.e., v', v satisfy the conditions in Definition 7.4.16).

Hence, and given Proposition 7.4.17, we can reshape Theorem 7.4.15 as follows.

Theorem 7.4.18 *If $\mathcal{Q}$ is a class of CQs and c is a constant such that for any $q \in \mathcal{Q}$:*

(a) $t(q) \leq c$, and

(b) for each pair $v, v' \in \mathsf{Vl}(q)$, $|\{\, p' \mid p' \sqsubseteq^{\mathcal{R}} p, p(v, v') \in q, \text{ and } \mathsf{trans}(p) \in \mathcal{R}\}| \leq c$,

then deciding $\vec{d} \in \mathsf{ans}(q, \mathcal{K})$ for a given $q \in \mathcal{Q}$ and tuple $\vec{d}$, is feasible in time single exponential in $||q|| + ||\mathcal{K}||$.

Of course this includes, in particular, the case where all roles are simple.

Corollary 7.4.19 *(Full) query answering in $\mathcal{ALCH}$ is feasible in single exponential time in the size of the input.*

From the well-known ExpTime-hardness of satisfiability testing in $\mathcal{ALC}$ [Sch91], it follows that these bounds are tight: the presented query answering procedure is worst-case optimal for $\mathcal{ALCH}$ and for all CQs that comply to Theorem 7.4.18.

Data Complexity

A straightforward adaptation of our query answering procedure to a non-deterministic version yields an algorithm that is worst-case optimal in data complexity. In particular, we show the coNP upper bound. To see it, we assume a fixed CQ q, a TBox $\mathcal{T}$ and an RBox $\mathcal{R}$, and analyze the complexity of deciding $\vec{a} \in \mathsf{ans}(q, \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle)$ for a given ABox $\mathcal{A}$ and a tuple $\vec{a}$ of individuals, where $\mathcal{A}$ is extensionally reduced, and $\mathsf{N_C}(\mathcal{A}) \cup \mathsf{N_R}(\mathcal{A}) \subseteq \mathsf{N_C}(\mathcal{T}) \cup \mathsf{N_R}(\mathcal{T}) \cup \mathsf{N_R}(\mathcal{R})$, i.e., the concept assertions in $\mathcal{A}$ are all of the form $A(a)$ for some concept name A, and all concepts and roles that occur in $\mathcal{A}$ occur also in $\mathcal{T}$ or $\mathcal{R}$. Then we let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.

Recall that for any set $\mathbf{T}$ of types for $\mathcal{K}$ we can obtain a $\mathbf{T}$-complete knot set $\mathbb{K}$ in time single exponential in $|Cl_{\mathsf{C}}(\mathcal{K})| + |\mathsf{N_R}(\mathcal{K})|$ (see $(*)$ in the proof of Theorem 7.4.1). The slight difference from the algorithm in Figure 8, is that we want to achieve independence from any specific ABox. We thus set $\mathbf{T}$ to the set $\mathsf{typesC}(\mathcal{K})$ of all possible types for $\mathcal{K}$, and compute a $\mathsf{typesC}(\mathcal{K})$-complete knot set $\mathbb{K}$. Note that $\mathbb{K}$ depends only on $\mathcal{T}$ and $\mathcal{R}$, and is $\mathsf{ABoxTypes}(\mathcal{K}')$-complete for any KB $\mathcal{K}' = \langle \mathcal{A}', \mathcal{T}, \mathcal{R} \rangle$. Then we can compute a complete type-query TQ for $\mathbb{K}$ and q; we have shown that the time required for this is polynomial in $|\mathbb{K}| + 2^{|\mathbb{F}_q^{\mathcal{R}}|}$. As $\mathcal{T}$, $\mathcal{R}$ and q are fixed, constructing $\mathbb{K}$ and TQ is feasible in constant time.

Given $\mathbb{K}$ and TQ, the complexity result follows from the fact that deciding $\vec{a} \notin \mathsf{ans}(q, \mathcal{K})$ is feasible in non-deterministic polynomial time in $||\mathcal{A}||$. Indeed, this can be done in a guess-and-check manner as follows:

- Build non-deterministically an expansion $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathbb{K}, \mathsf{TQ})$. More precisely, guess an ABox completion θ and, for each individual $a \in \mathsf{N_I}(\mathcal{A})$, add $\mathsf{abox}(a, k(a), g_{k(a)})$ according to a non-deterministic choice of some knot $k(a) = (\mathbf{t}, \mathbf{S})$ from $\mathbb{K}$ with root type $\mathbf{t} = \theta(a)$, and some f-query assignment $g_{k(a)}$ for $k(a)$ as required in in Definition 7.3.20 (i.e., that assigns TQ-hits to all children). Observe that since $\mathcal{T}$, $\mathcal{R}$ and q are fixed, such an expansion $\mathcal{A}'$ can be non-deterministically computed in polynomial time in $||\mathcal{A}||$ (the maximum number of possible concept and role types are fixed and, since the number of individuals is linear in $||\mathcal{A}||$, checking the conditions of Definition 7.1.7 is simple; also, for each a, $\mathsf{abox}(a, k(a), g_{k(a)})$ has size bounded by a constant, and only constantly many different $\mathsf{abox}(a, k(a), g_{k(a)})$ exist).

- Verify $\vec{a} \notin \mathsf{ans}(q, \mathcal{A}')$. There are $|\mathsf{N_I}(\mathcal{A}')|^{|\mathsf{VI}(q)|}$ different candidate query mappings π for q in $\mathcal{A}'$. As $|\mathsf{VI}(q)|$ is fixed and $|\mathsf{N_I}(\mathcal{A}')|$ is linear in $||\mathcal{A}||$, the number of such candidates is polynomial in $||\mathcal{A}||$. Testing whether π witnesses $\vec{a} \in \mathsf{ans}(q, \langle \mathcal{A}', \emptyset, \emptyset \rangle)$ is also polynomial in $||\mathcal{A}||$. Hence, the verification step is feasible in polynomial time in $||\mathcal{A}||$.

This yields a tight coNP upper bound for the data complexity of the problem (the upper bound was shown in [GHLS08], and the lower bound is due to [Sch94a]; please see Section 7.5 for a detailed discussion).

We get an analogous coNP result for answering varying CQs of small size (bounded by a constant) over a knowledge base $\mathcal{K}$ with a fixed TBox and RBox, if its compilation into a suitable

T-complete knot $\mathbb{K}$ set is available; this may be due to off-line pre-compilation, or to caching $\mathbb{K}$ after the first query. In this setting, the tq-table TQ is constructible in polynomial time (only constantly many subqueries exist), and deciding whether $\vec{a} \notin \mathsf{ans}(q, \mathcal{K})$ is still feasible in NP. In fact, if also the ABox is fixed, the last step is feasible in polynomial time (only constantly many expansions $\mathcal{A}'$ of $\mathcal{A}$ and constantly many candidate mappings of $\mathsf{VI}(q)$ into each $\mathcal{A}'$ exist, which can be easily traversed).

7.4.4 Encoding into Datalog

The guess-and-check procedure above can easily be simulated in disjunctive Datalog [DEGV01]. A disjunctive Datalog *program* P consists of *rules* of the form

$$A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_n \qquad m + n > 0 \tag{7.3}$$

where each A_i and each B_j is a function-free first-order atom. We assume that the rules are *safe*, which means that each variable occurring in A_i also occurs in some B_j. The semantics of such a program P is given by the minimal (w.r.t. $\subseteq$) sets of ground (variable-free) atoms that are closed under the rules of P (called minimal models or answer sets). A ground atom A is a *cautious consequence* of P, if A occurs in all answer sets of P.

Viewing individuals as constants and using disjunctive rules, it is not hard to generate the expansions $\mathcal{A}'$ of an input ABox $\mathcal{A}$ as answer sets of a program $P(\mathcal{A})$. Using Theorem 7.3.21, answering a query $q = \exists \vec{v} \varphi(\vec{x}, \vec{v})$ amounts to answering the Datalog query $q(\vec{x}) \leftarrow \varphi(\vec{x}, \vec{v})$ over $P(\mathcal{A})$, and the latter is in turn equivalent to collecting all tuples $\vec{a}$ such that $q(\vec{a})$ is a cautious consequence of $P(\mathcal{A})$.

The program $P(\mathcal{A})$ can be generated from the precomputed knot set $\mathbb{K}$ and the tq-table TQ. The rules in $P(\mathcal{A})$ must: 1. generate a completion of $\mathcal{A}$, and then 2. generate an expansion, associating to each individual a an expanded ABox $\mathsf{abox}(a, \mathbf{k}, g)$ for some non-deterministically chosen knot $\mathbf{k}$ and assignment g. The first task is straightforward:

(i) assertions $A(a)$ and $p(a, b)$ in $\mathcal{A}$ are simply stated as facts in $P(\mathcal{A})$,

(ii) the membership of individuals in atomic concepts and roles can be "guessed" using disjunctive facts of the form $A(a) \vee \overline{A}(a) \leftarrow$ and $p(a, b) \vee \overline{p}(a, b) \leftarrow$,

(iii) the closure of the concept types under the conditions in Table 7.1, as well as the closure of role types under the $\sqsubseteq^{\mathcal{R}}$ relation and the closure of roles under transitivity can easily be enforced using rules with (bounded number of) variables; consistency of types is ensured using simple constraints $\leftarrow A(X), \overline{A}(X)$ and $\leftarrow p(X, Y), \overline{p}(X, Y)$.

For generation of the expansions (step 2), we use a constant $\mathbf{k}$ for each knot in $\mathbb{K}$ and a constant g for each $\mathbf{t}$-hit of TQ. Then we can use a ground atom $abox(a, \mathbf{k}, g)$ for each individual $a \in \mathsf{N_I}(\mathcal{A})$, knot $\mathbf{k}$ and assignment g, and trigger the expansion of the abox as follows:

(i) a disjunctive rule of the form $abox(a, \mathbf{k}_1, g_1) \vee \cdots \vee abox(a, \mathbf{k}_n, g_n) \leftarrow$ triggers the non-deterministic choice of a particular k and g for each a, and

(ii) rules $H \leftarrow abox(a, \mathbf{k}, g)$ for all assertions $H \in \mathsf{abox}(a, \mathbf{k}, g)$ ensure the correct expansion of $\mathcal{A}$.

The program $P(\mathcal{A})$ is constructible in polynomial time in $||\mathcal{A}||$, and since cautious inference from disjunctive Datalog programs with bounded number of variables is coNP-complete, this reduction is also worst-case optimal in data complexity.

We finally note that instead of disjunction, also (unstratified) negation may be used for the encoding. Thus, a range of reasoning engines for disjunctive/unstratified Datalog (e.g., DLV [ELM+97], smodels [NS97], clasp [GKNS07]) can be used for implementation.

7.5 Discussion and Conclusion

In this chapter, we have presented two main contributions:

- We have described a novel algorithm for CQ answering over knowledge bases in $\mathcal{SH}$ which has some attractive features. Most notably, an initial query compilation step allows us to employ efficient Datalog engines to reason over different ABoxes. The algorithm runs in 2ExpTime in general but only in ExpTime for $\mathcal{ALCH}$. It is worst-case optimal both in combined and data complexity.
- We have identified transitive roles and role inclusions as a source of complexity that makes query answering 2ExpTime-hard in all expressive DLs containing $\mathcal{SH}$.

7.5.1 Comparing the Knot-Based Approaches

The algorithm for $\mathcal{ALCH}$ and $\mathcal{SH}$ described in this chapter is based on knots, and so is the algorithm for $\mathcal{ALCH}$ and $\mathcal{ALCHI}$ that we discussed in Chapter 6. They illustrate two different ways to apply knots for query answering, which are dual in some sense.

For a given query q, the algorithm in Chapter 6 marks knots with sets of subqueries of q for which a match must be avoided, while the algorithm we described in this chapter computes, for each knot, an entailed disjunction of subqueries for which a match certainly exists. The approach in Chapter 6 is more compact and conceptually simpler, and it resembles a non-deterministic top-down construction of a canonical model. The second algorithm, on the other hand, can be seen as more constructive way of computing query answers in a bottom-up way, which allows to reduce the level of non-determinism by using more 'local' information at each computation step. The second approach is well suited to directly handle CQs with answer variables by encoding into disjunctive Datalog; for the first approach, this is not apparent.

The two approaches also have common features that can be seen as inherent to the knot based approach. The computation of the knot-based representation of models is independent of the queries, and provides a worst-case optimal method for standard reasoning. Query answering is then realized by associating to the knots query information, and optimal complexity bounds can be achieved for different logics provided that the *relevant subqueries* of the query are adequately characterized.

7.5.2 Data Complexity

The knot-based algorithms we have presented in chapters 6 and 7 are both optimal w.r.t. data complexity. The bounds are not new, however.

The coNP lower bound has been known for many years already, and for a DL that is significantly weaker than the basic $\mathcal{ALC}$ [Sch94b]. Naturally, research on data complexity has then focused mostly on the so-called *light-weight DLs*, which do not contain the full $\mathcal{ALC}$, and significant progress has been done in understanding the boundaries of tractability and extending as far as possible the languages for which query answering can be done efficiently, in polynomial time, as for $\mathcal{EL}$ [KRH07, Ros07, KL07], or even in logarithmic space, as it is the case in the DL-Lite families [CGL$^+$07].

The few attempts that have aimed at establishing tight upper bounds for expressive DLs have so far lead to the same coNP bound. This was established for standard reasoning in $\mathcal{ALCHIQ}$ in [Mot06], for query answering in $\mathcal{ALCHIQ}$, $\mathcal{ALCHOQ}$ and $\mathcal{ALCHIO}$ in [OCE08], for query answering in $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ in [GLHS08, Gli07], and for query answering in the guarded two-variable fragment of first-order logic with counting quantifiers in [PH09]. These results indicate that, unlike the combined complexity, the data complexity of query answering is not higher than the one of standard reasoning, at least for a wide range of Description Logics.

7.5.3 Datalog Encoding and Knowledge Compilation

The algorithm we have described in this chapter handles CQs with distinguished (output) variables directly. Although allowing answer variables does not make a difference for theoretical complexity considerations, their impact in practice can be significant. Indeed, when one is interested in enumerating query answers, one makes a customary grounding step in which an input query is reduced to possibly exponentially many Boolean queries, one for each possible output tuple $\vec{c}$, and for each tuple an independent run of the algorithm is required. This grounding step can be expensive in practice and generate a lot of redundant calls to the procedure.

Our approach also provides a promising alternative to previous algorithms when we think in terms of data complexity. Indeed, to our knowledge, previous algorithms for query answering in expressive DLs which are optimal w.r.t. data complexity, usually do an initial guessing step related to the ABox, and then execute a complex procedure that is exponential in the size of the intensional axioms and the query in the general case. The procedure terminates in polynomial time if both components are fixed, but may result in large polynomials in many cases. Furthermore, the full computation is repeated from scratch for each input ABox. This applies, for example, to the algorithms in [OCE08, GHLS08, Gli07, Lut08b], and even to the algorithm in Chapter 6. The algorithm described in this chapter, on the other hand, seems to be more amenable to optimizing the reasoning procedures w.r.t. the data, as the guessing related to the data is deferred to the last step, and it is independent of the complex reasoning steps over the intensional axioms and query, which can be spared in later calls. Indeed, the TBox and RBox can be compiled into a knot set and then, together with the query, into a complete tq-table. Both steps are completely independent of the data. Hence, in the context of relatively fixed intensional information and a pre-defined query to be evaluated over dynamic data, doing these two computationally expensive steps off-line could lead to a significant speed-up in practice. Finally, for each input ABox, queries with answer variables can be evaluated directly over the Datalog encoding of the compiled knowledge, exploiting existing efficient engines.

7.5.4 Related Work

We have compared our algorithm to other knot-based algorithms earlier in this section, and discussed other approaches that are related to knots in Section 6.6.1. Of the other existing algorithms for query answering in expressive DLs, the resolution-based method by Hustadt et al. [HMS05] is perhaps most closely related to ours. Similar as in our approach, their method first "compiles" the knowledge base and the query into a special form, and then exploits the possibility to answer the query by means of a disjunctive Datalog program. However, this is done on different grounds: the knot technique is model-theoretic in nature, while Hustadt et al.'s method is proof-theoretic, cleverly exploiting resolution and superposition machinery. Furthermore, the knot technique handles transitive roles in the query, which are not allowed in [HMS05], and this algorithm is worst-case optimal for data complexity. In contrast, their technique yields optimal bounds for data complexity of standard reasoning, but only a non-optimal 2ExpTime upper bound for query answering.

Chapter 8

Summary and Conclusions

In this thesis we have studied the problem of query answering in expressive Description Logics, with special emphasis on identifying suitable *reasoning techniques*, and deriving precise *complexity results*.

Regarding the first aspect, we have explored two kinds of techniques: based on automata on infinite forests, and based on knots. They both proved to be adequate to achieve our goal. We used them to develop query answering algorithms for a wide range of description logics, and we analyzed their complexity to obtain upper complexity bounds. By identifying a new source of complexity, we were also able to prove that most of the given upper bounds are tight.

We hope that these results will contribute to a better understanding of the feasibility of accessing data repositories governed by expressive knowledge representation languages, and of the challenges that must be faced. The complexity results we have obtained—ranging from ExpTime-completeness to membership in 3ExpTime—show that queries over expressive Description Logics are a powerful tool which can be used to solve complex problems in a concise way. A clear understanding of its power and of the sources of its complexity enables us to use this tool in a more adequate way. While many researchers and developers agree that having flexible mechanisms to query knowledge bases is desirable, there is not so much common ground when it comes to finding the right formalisms and choosing the best tools to realize query answering services. There are still many challenges to be faced before reaching the maturity of, for example, standard reasoning in Description Logics, where developers can choose from a broad selection of formalisms and rely on off-the-shelf engines to provide reasoning services. We believe that understanding the power and the limitations of query languages over knowledge bases is an important challenge, and hope that our contribution will be a step in the right direction.

8.1 Discussion

In the rest of this chapter, we discuss in more detail some of the implications of our results, and the challenges that remain for future work. To conclude this thesis, in Section 8.2 we do a detailed recount of the complexity bounds we have derived and compile them together with other results from the literature, to draw a comprehensive picture of the current landscape of computational complexity of query answering in expressive Description Logics.

8.1.1 Automata Theoretic Techniques for Query Answering

The automata-based algorithm for query answering developed in Chapters 3 and 4 and the derived complexity results show that automata on infinite objects are a powerful and flexible tool in this setting. The algorithm is very general: many popular DLs and many important query languages can be handled by our algorithm as special cases. Furthermore, the resulting upper bound is not higher than the upper bounds for expressive logics that had been obtained before, even in significantly restricted settings.

The generality of the technique can also be seen as a limitations. Intuitively, in an automata algorithm, the complexity stemming from all the different sources is compiled together into the states of the automaton. The different factors that may contribute to the complexity of the problem, such as the size of the extensional and intensional components, the different constructors allowed in the logic, the size of the query and the different possible shapes of query mappings, contribute to the size of the state set. The emptiness algorithms allow us to measure the complexity in terms of the number of states, but it is often hard to distinguish how the contribution of the different elements is reflected in the final complexity bounds. If in addition the construction uses different automata operations, as in Chapter 4, then it is even harder to account for the independent contributions of different factors to the final bound, and all the structure of the original problem is lost. As a result, the automata algorithm does not allow us to single out the contribution of the extensional component in a way that yields a tight upper bound for the data complexity. Deriving more fine grained bounds for syntactically restricted cases, for example, was relatively easy with the knot-based approach, while with automata it does not seem feasible without much more intricate book-keeping and possibly more involved automata constructions [KV01, BVW94].

An interesting direction for further research may be to apply these techniques to more expressive query languages, such as recursive Datalog queries under suitable restrictions. Another interesting direction is query answering over DLs equipped with some form of closed world assumption [MHRS06, Hus94, SFdB09], which may be of interest in the database setting. The techniques may be applicable to reason in hybrid formalisms that combine DLs and rules (see [ADG$^+$05] and its references), or to provide some query answering services in systems based on other logics of interest in knowledge representation, such as guarded fragments of first-order logic with predicates of arbitrary arities, possibly extended with fixed-point operators [GW99, Grä98].

8.1.2 The knot approach to query answering

Chapters 6 and 7 described two different but related knot-based techniques for query answering in DLs. From these algorithms it seems possible to abstract a general *knot approach* to query answering. Focusing on the 'tree-part' and disregarding ABox completions, the approach can be roughly described as follows. To answer a query q over a KB $\mathcal{K}$ using knots, we need:

- A suitable characterization of (the tree parts of) a canonical model for KBs in terms of *knot sets*. That is, a notion of a coherent knot set such that the structures that can be built from it coincide with the canonical models of $\mathcal{K}$.
- An adequate definition of a set of *relevant subqueries* of q, and a suitable notion of *matches* for such subqueries, such that (i) the existence of a match for a subquery can be decided *locally* at each knot $\mathbf{k}$, and depends only on $\mathbf{k}$ and on the existence of located matches for some *smaller subqueries*, and (ii) the existence of a match for q can be characterized in terms of matches for the subqueries.

With this machinery, query entailment reduces to deciding the existence of some relations between knots and sets of subqueries. For example, in Chapter 6, we had to decide the existence of

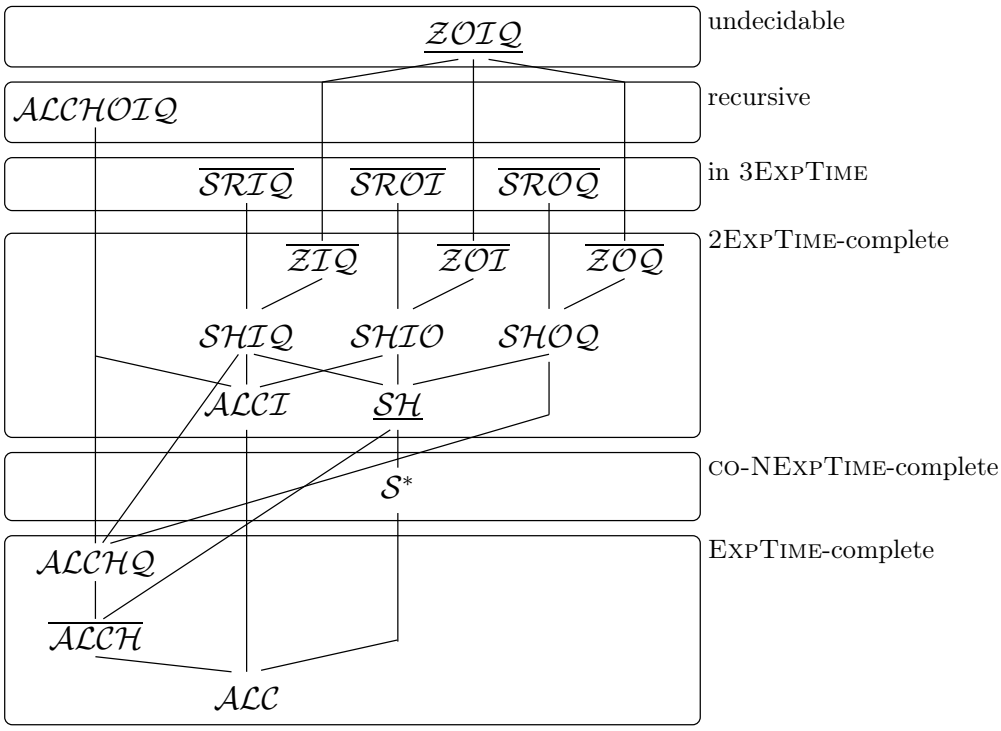

Figure 8.1: The complexity of query answering in expressive DLs. For the overlined/underlined logics, the upper/lower bound was shown in this thesis.

a set of marked knots, and in Chapter 7 of a knot-query table. The complexity of the algorithm is determined by the number of possible associations of knots and sets of subqueries. Once this is bounded, an optimal procedure for deciding the existence of the desired set is not hard to obtain. Hence, the combined complexity of query answering in a DL fundamentally depends on the size of the set of all subqueries that needs to be considered in the knot associations.

From the results we have discussed, we can conclude that for some expressive DLs that enjoy forest-shaped canonical models, the complexity of query answering is determined by the following factor. Once a match for some query variable in an interpretation has been fixed, in how many ways can the other variables be organized into subqueries? If this number is polynomially bounded, then query answering is not harder than satisfiability. In particular, this is the case when the query has only polynomially many tree rewritings.

This contrasts with the behavior of DLs where disjunction is disallowed. For example, in Horn-$\mathcal{SHIQ}$ there are exponentially many subqueries to be considered, but it is not necessary to store sets of them at the knots. Instead, we can pose them one by one over one single *universal model* that is good for answering all queries. Hence query answering is feasible in ExpTime and is not harder than knowledge base satisfiability [EGOŠ08].

8.1.3 Transitive roles and the complexity of query answering

The impact of transitivity in the complexity of query answering was a longstanding open problem. From a practical point of view it was acknowledged that it made the design of algorithms harder. Most of the early query answering algorithms did not allow for unrestricted transitive roles in the query [LR98a, Tes01, HMS04, OCE08], and algorithms that allowed them were much more intricate than algorithms that did not [GHLS07, CEO07]. Transitive roles are desirable from an application perspective: they often play an important role in ontologies, as they are

used to represent fundamental relations such as 'part of' [Sat00]. However, the development of ExpTime algorithms for query answering in extensions of $\mathcal{ALC}$ seemed to require that transitive roles were disallowed or severely restricted [Lut08b, OŠE08b], and a proof of hardness for some class higher than ExpTime was not apparent.

As a first tangible contribution towards settling the problem, we have shown in Chapter 7 that in the presence of transitive roles and role inclusions, query answering in any extension of $\mathcal{ALC}$ becomes 2ExpTime hard. If the number of transitive atoms is suitably bounded (see Theorem 7.4.18), then the problem is still solvable in ExpTime. The hardness proof, however, requires the presence of a transitive role and a role inclusion, and does not provide conclusive evidence of the effect of transitive roles alone.

As it turns out, transitive roles alone are a source of complexity, but a rather subtle one. It is not easy to adapt the knot marking algorithm described in Chapter 6 without causing an exponential blow-up, because queries with transitive roles can have exponentially many tree rewritings even on one-way canonical models. In fact, annotations can be of exponential size, and a similar reduction to simple ABoxes would not give sets of queries of polynomial size. Over simple KBs that have only one individual in the ABox one can use a sophisticated adaptation of the knot marking algorithm, which uses a weaker form of tree-rewritings called *pseudo-tree queries*, to show that query answering in $\mathcal{S}$ is feasible in exponential time [ELOŠ09b]. This upper bound extends to ABoxes whose relational structure is a tree. However, using an ABox that is not tree-shaped (but DAG-shaped), one can show that CQ entailment in $\mathcal{S}$ is co-NExp-Time-hard [ELOŠ09b]. That the shape of the ABox has such an impact on the complexity is surprising, and there is no evidence so far of a similar behavior in other well-known DLs. A matching co-NExpTime upper bound was most recently established for the case where there is only one role available, and it seems plausible that the bound may extend to the general case [BEL$^+$10].

8.2 The Complexity of Query Answering in Expressive DLs

Significant progress has been made in the last years towards understanding the complexity landscape for query answering in expressive DLs. In this thesis we have settled some of the open questions, although others remain. The main complexity-related contributions of this thesis are the following:

(1) We have shown that knowledge base satisfiability in any fragment of $\mathcal{ZOIQ}$ enjoying the canonical model property, and in particular in $\mathcal{ZIQ}$, $\mathcal{ZOQ}$ and $\mathcal{ZOI}$, is decidable in single exponential time (Theorem 3.4.2).

(2) We have shown a 2ExpTime upper bound for entailment of positive two-way regular path queries in $\mathcal{ZIQ}$ (Theorem 4.2.2), and containment of conjunctive queries in positive two-way regular path queries w.r.t. $\mathcal{ZIQ}$ knowledge bases (Theorem 4.2.3). We also proved a 2ExpTime upper bound for entailment and containment of positive two-way regular path queries in $\mathcal{ZOQ}$ and $\mathcal{ZOI}$ (Theorems 4.2.2 and 4.2.3).

(3) We have obtained a 2ExpTime upper bound for deciding knowledge base satisfiability in any fragment of $\mathcal{SROIQ}$ whose rewriting into $\mathcal{ZOIQ}$ enjoys the canonical model property, and in particular for $\mathcal{SRIQ}$, $\mathcal{SROQ}$, and $\mathcal{SROI}$ (Theorem 5.2.2). If, in addition, the input knowledge base is sparse then satisfiability can be decided in single exponential time.

(4) The query entailment and containment results were also extended to the DLs of the $\mathcal{SR}$ family. Namely, entailment of positive two-way regular path queries in $\mathcal{SRIQ}$ and containment of conjunctive queries in positive two-way regular path queries w.r.t. $\mathcal{SRIQ}$ knowledge

bases are decidable in 3ExpTime, while entailment and containment of positive two-way regular path queries in $\mathcal{SROQ}$ and $\mathcal{SROI}$ are decidable in 3ExpTime (Theorems 5.3.3 and 5.3.4). If the knowledge base is sparse or only simple roles occur in the query, then we obtain a 2ExpTime upper bound.

(5) We have shown a tight ExpTime upper bound for entailment of unions of conjunctive queries in $\mathcal{ALCH}$ (Theorem 6.5.1), and for entailment of conjunctive queries in $\mathcal{SH}$ whenever the occurrences of non-simple roles in the query satisfy certain syntactic restrictions (Theorem 7.4.18).

(6) We have shown that conjunctive query entailment in $\mathcal{SH}$ is 2ExpTime-hard (Theorem 7.4.3).

The results listed in items (1) to (4) were published in [CEO09b]. A previous version of the algorithm that only covered $\mathcal{ALCQIb}_{reg}$ had been published in [CEO07], and it was extended to $\mathcal{ZIQ}$ and $\mathcal{SRIQ}$ in [Ort08]. The results listed in item (2) generalize previous 2ExpTime upper bounds for entailment of union of conjunctive queries in $\mathcal{SHIQ}$ [GHLS08], entailment of positive two-way regular path queries in $\mathcal{ALCQIb}_{reg}$ [CEO07], containment of conjunctive queries in $\mathcal{ALCQI}_{reg}$ [CDGL08], and entailment of unions of conjunctive queries in $\mathcal{SHOQ}$ [GHLS08]. They improve significantly the 3NExpTime upper bounds for entailment of positive queries (without regular expressions) in the weaker $\mathcal{ALCHOQ}$ and $\mathcal{ALCHOQ}$ from [OCE08]. The given bound does not hold for $\mathcal{ZOIQ}$ in general, where positive two-way regular path query entailment is in fact undecidable [ORŠ10]. The only decidability result so far for an expressive DL combining $\mathcal{I}$, $\mathcal{O}$ and $\mathcal{Q}$ was shown for $\mathcal{ALCHOIQ}$ in [GR09], but no elementary upper bound on the complexity can be obtained from the algorithm described there. The results listed in item (3) improve over the 2NExpTime upper bound that the 3 sublogics inherit from $\mathcal{SROIQ}$ [Kaz08], while the results mentioned in (4) are, to our knowledge, the first bounds for query entailment in these DLs.

The first result mentioned in item (5) was shown here using an extension of a technique published in [ELOŠ09a]. ExpTime upper bounds for conjunctive query entailment were shown for $\mathcal{ALCHQ}$ in [Lut08a] and for $\mathcal{ALCH}$ in [OŠE08b]. The latter was extended to restricted families of queries in $\mathcal{SH}$ in [OŠE08a, EOŠ09], where the second result mentioned in item (5) was published.

Finally, the result mentioned in item (6) was published in [ELOŠ09b], where we also showed that conjunctive query entailment in the DL $\mathcal{S}$ is co-NExpTime-hard in general, but in ExpTime if the ABox is tree-shaped. A tight co-NExpTime upper bound was most recently established for queries with only one role [BEL$^+$10]. We believe it will extend to the general case, and showing this is part of our ongoing research. Item (6), together with the 2ExpTime-hardness for $\mathcal{ALCI}$ shown in [Lut07], implies that the results in item (2) are tight.

These results are summarized in Figure 8.1. The overlined logics indicate the upper bounds shown in this thesis, and the underlined $\mathcal{SH}$ indicates that the lower bound was also shown in the thesis. All bounds hold for CQ entailment, but the undecidability of $\mathcal{ZOIQ}$ requires that regular role expressions are allowed in the query. The 2ExpTime and 3ExpTime upper bounds are also for P2RPQs. In the case of $\mathcal{S}$, the upper bound has only been shown for the restricted case in which the query contains only one role.

Bibliography

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[ACKZ09] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The *DL-Lite* family and relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009.

[ACPS96] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 137–148, 1996.

[ADG+05] Grigoris Antoniou, Carlos Viegas Damasio, Benjamin Grosof, Ian Horrocks, Michael Kifer, Jan Maluszynski, and Peter F. Patel-Schneider. Combining rules and ontologies. A survey. Technical Report Deliverable I3-D3, REWERSE Project, February 2005. Available at `http://rewerse.net/deliverables/m12/i3-d3.pdf`.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[AV99] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *J. of Computer and System Sciences*, 58(3):428–452, 1999.

[Baa91] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, 1991.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Sc.* Cambridge University Press, Cambridge, 2001.

[BEL+10] Meghyn Bienvenu, Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Šimkus. Query answering in the description logic $\mathcal{S}$. In *Proceedings of the 23rd International Workshop on Description Logics (DL2010)*, CEUR-WS, 2010.

[BHLW03] Franz Baader, Jan Hladik, Carsten Lutz, and Frank Wolter. From tableaux to automata for description logics. *Fundamenta Informaticae*, 57:1–33, 2003.

[BLM08] Franz Baader, Carsten Lutz, and Boris Motik, editors. *Proceedings of the 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[BLMV08] Piero Bonatti, Carsten Lutz, Aniello Murano, and Moshe Y. Vardi. The complexity of enriched μ-calculi. *Logical Methods in Computer Science*, 4(3:11):1–27, 2008.

[BLR03] Alexander Borgida, Maurizio Lenzerini, and Riccardo Rosati. Description logics for data bases. In Baader et al. [BCM$^+$03], chapter 16, pages 462–484.

[BP04] Piero A. Bonatti and Adriano Peron. On the undecidability of logics with converse, nominals, recursion and counting. *Artificial Intelligence*, 158(1):75–96, 2004.

[Bun97] Peter Buneman. Semistructured data. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 117–121, 1997.

[BVW94] Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. In *Proc. of the 6th Int. Conf. on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 1994.

[CDG03] Diego Calvanese and Giuseppe De Giacomo. Expressive description logics. In Baader et al. [BCM$^+$03], chapter 5, pages 178–218.

[CDGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[CDGL02] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2ATAs make DLs easy. In *Proc. of the 15th Int. Workshop on Description Logic (DL 2002)*, pages 107–118. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-53/`, 2002.

[CDGL$^+$05a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proc. of the 18th Int. Workshop on Description Logic (DL 2005)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-147/`, 2005.

[CDGL$^+$05b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[CDGL$^+$06] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, 2006.

[CDGL+07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[CDGL08] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment and answering under description logics constraints. *ACM Trans. on Computational Logic*, 9(3):22.1–22.31, 2008.

[CDGLV99] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting regular expressions in semi-structured data. In *Proc. of ICDT'99 Workshop on Query Processing for Semi-Structured Data and Non-Standard Data Formats*, 1999.

[CDGLV00] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 176–185, 2000.

[CDGLV02] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *J. of Computer and System Sciences*, 64(3):443–465, 2002.

[CDGLV03] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

[CDGV03] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Decidable containment of recursive queries. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, volume 2572 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2003.

[CDGV05] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Decidable containment of recursive queries. *Theoretical Computer Science*, 336(1):33–56, 2005.

[CEO07] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics: An automata-theoretic approach. In *Proceedings of the Twentysecond AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 391–396, 2007.

[CEO09a] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. Technical Report INFSYS RR-1843-09-04, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, December 2009.

[CEO09b] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Regular path queries in expressive description logics with nominals. In C. Boutilier, editor, *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pages 714–720, 2009.

[CGHM+08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008.

[CGL+07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *l-lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77)*, pages 77–90, 1977.

[DdN05] Stéphane Demri and Hans de Nivelle. Deciding regular grammar logics with converse through first-order logic. *J. of Logic, Language and Information*, 14(3):289–329, 2005.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[DFN97] Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors. *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97, Dagstuhl Castle, Germany, July 28-31, 1997, Proceedings*, volume 1265 of *Lecture Notes in Computer Science*. Springer, 1997.

[DG95] Giuseppe De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

[DGL94] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI'94)*, pages 205–212, 1994.

[DT01] Alin Deutsch and Val Tannen. Optimization properties for classes of conjunctive regular path queries. In Giorgio Ghelli and Gösta Grahne, editors, *Proc. of the 8th Int. Workshop on Database Programming Languages (DBPL 2001)*, volume 2397 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2001.

[EGOŠ08] Thomas Eiter, Georg Gottlob, Magdalena Ortiz, and Mantas Šimkus. Query answering in the description logic horn-$\mathcal{SHIQ}$. In *Proceedings of the Eleventh European Workshop on Logics in Artificial Intelligence (JELIA 2008)*, pages 166–179, Berlin, Heidelberg, 2008. Springer-Verlag.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. of the 32nd Annual Symp. on the Foundations of Computer Science (FOCS'91)*, pages 368–377, 1991.

[ELM+97] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A deductive system for non-monotonic reasoning. In Dix et al. [DFN97], pages 364–375.

[ELOŠ09a] Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Šimkus. Query answering in description logics: The knots approach. In Hiroakira Ono, Makoto Kanazawa, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009*, volume 5514 of *Lecture Notes in Computer Science*, pages 26–36. Springer, 2009.

[ELOŠ09b] Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Šimkus. Query answering in description logics with transitive roles. In C. Boutilier, editor, *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pages 759–764, 2009.

[EOŠ09] Thomas Eiter, Magdalena Ortiz, and Mantas Šimkus. Conjunctive query answering in the description logic sh using knots. Technical Report INFSYS RR-1843-09-03 (available at `http://www.kr.tuwien.ac.at/research/reports/`), 2009.

[EŠ10] Thomas Eiter and Mantas Šimkus. Fdnc: Decidable nonmonotonic disjunctive logic programs with function symbols. *ACM Trans. Comput. Log.*, 11(2), 2010.

[FG08] Dieter Fox and Carla P. Gomes, editors. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. AAAI Press, 2008.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

[FLS98] Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 139–148, 1998.

[GHLS07] Birte Glimm, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.

[GHLS08] Birte Glimm, Ian Horrocks, Carsten Lutz, and Uli Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research*, 31:151–198, 2008.

[GHS06] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for description logics with transitive roles. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Proc. of the 19th Int. Workshop on Description Logic (DL 2006)*, volume 189, Windermere, Lake District, United Kingdom, May 2006. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`.

[GHS07] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Conjunctive query entailment for $\mathcal{SHOQ}$. In *Proc. of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/Vol-250/`, pages 65–75, 2007.

[GHS08] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of conjunctive queries in shoq. In *Proceedings of the 11th International Conference on the Principles of Knowledge Representation and Reasoning (KR-08)*, pages 252–262. AAAI Press/The MIT Press, 2008.

[GKNS07] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *Clasp* : A conflict-driven answer set solver. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Proc. LPNMR-07*, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2007.

[GKV97] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

[GLHS08] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Answering conjunctive queries in the $\mathcal{SHIQ}$ description logic. *Journal of Artificial Intelligence Research*, 31:150–197, 2008.

[Gli07] Birte Glimm. *Querying Description Logic Knowledge Bases.* PhD thesis, The University of Manchester, Manchester, United Kingdom, 2007.

[GR09] Birte Glimm and Sebastian Rudolph. Conjunctive query entailment: Decidable in spite of O, I, and Q. In *Proceedings of the 2009 Description Logic Workshop (DL 2009)*, 2009.

[Grä98] Erich Grädel. Guarded fragments of first-order logic: A perspective for new description logics? In *Proc. of the 11th Int. Workshop on Description Logic (DL'98).* CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-11/`, 1998.

[GT03] Gösta Grahne and Alex Thomo. Query containment and rewriting using views for regular path queries under constraints. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2003)*, pages 111–122, 2003.

[GU92] Ashish Gupta and Jeffrey D. Ullman. Generalizing conjunctive query containment for view maintenance and integrity constraint verification (abstract). In *Workshop on Deductive Databases (In conjunction with JICSLP)*, page 195, Washington D.C. (USA), 1992.

[GW99] Erich Grädel and Igor Walukiewicz. Guarded fixed point logic. In *Proc. of the 14th IEEE Symp. on Logic in Computer Science (LICS'99)*, pages 45–54. IEEE Computer Society Press, 1999.

[HKS05] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The irresistible $\mathcal{SRIQ}$. In *Proc. of the 1st Int. Workshop on OWL: Experiences and Directions (OWLED 2005)*, 2005.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic.* The MIT Press, 2000.

[Hla04] Jan Hladik. A tableau system for the description logic shio. In Ulrike Sattler, editor, *IJCAR Doctoral Programme*, volume 106 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2004.

[HM92] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artif. Intell.*, 54(3):319–379, 1992.

[HMS04] Ulrich Hustadt, Boris Motik, and Ulrike Sattler. A decomposition rule for decision procedures by resolution-based calculi. In *Proc. of the 11th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2004)*, pages 21–35, 2004.

[HMS05] Ulrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of reasoning in very expressive description logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005.

[HS03] Ian Horrocks and Ulrike Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*. Morgan-Kaufmann Publishers, 2003.

[HS04] Ian Horrocks and Ulrike Sattler. Decidability of shiq with complex role inclusion axioms. *Artif. Intell.*, 160(1):79–104, 2004.

[HS05] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, 2005.

[HST00] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *J. of the Interest Group in Pure and Applied Logic*, 8(3):239–264, 2000.

[HT00] Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic ABoxes. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.

[Hus94] Ullrich Hustadt. Do we need the closed world assumption in knowledge representation? In Franz Baader, Martin Buchheit, Manfred A. Jeusfeld, and Werner Nutt, editors, *KRDB*, volume 1 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1994.

[Kaz08] Yevgeny Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In *Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 274–284, 2008.

[KL07] Adila Krisnadhi and Carsten Lutz. Data complexity in the $\mathcal{EL}$ family of description logics. In *Proceedings of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/Vol-250/`, 2007.

[KM08] Yevgeny Kazakov and Boris Motik. A resolution-based decision procedure for SHOIQ. *J. of Automated Reasoning*, 40(2-3):89–116, 2008.

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KRH07] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Conjunctive queries for a tractable fragment of OWL 1.1. In al. Karl Aberer et. editor, *The Semantic Web, 6th Int. Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 2007.

[KSV02] Orna Kupferman, Ulrike Sattler, and Moshe Y. Vardi. The complexity of the graded μ-calculus. In Andrei Voronkov, editor, *Proc. of the 18th Int. Conf. on Automated Deduction (CADE 2002)*, volume 2392 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2002.

[KV98] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. of the 30th ACM SIGACT Symp. on Theory of Computing (STOC'98)*, pages 224–233. ACM Press, 1998.

[KV01] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2(3):408–429, 2001.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[LR98a] Alon Y. Levy and Marie-Christine Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[LR98b] Alon Y. Levy and Marie-Christine Rousset. Verification of knowledge bases based on containment checking. *Artificial Intelligence*, 101(1-2):227–250, 1998.

[LST05] Carsten Lutz, Ulrike Sattler, and Lidia Tendera. The complexity of finite model reasoning in description logics. *Inf. Comput.*, 199(1-2):132–171, 2005.

[Lut07] Carsten Lutz. Inverse roles make conjunctive queries hard. In *Proceedings of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/Vol-250/`, pages 100–111, 2007.

[Lut08a] Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2008.

[Lut08b] Carsten Lutz. Two upper bounds for conjunctive query answering in SHIQ. In Baader et al. [BLM08].

[LWZ08] Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporal description logics: A survey. In Stéphane Demri and Christian S. Jensen, editors, *Proc. 15th International Symposium on Temporal Representation and Reasoning (TIME 2008)*, pages 3–14. IEEE Computer Society, 2008.

[MHRS06] Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can OWL and logic programming live together happily ever after? In *Proceedings ISWC-2006*, volume 4273 of *LNCS*, pages 501–514. Springer, 2006.

[MLF00] Todd D. Millstein, Alon Y. Levy, and Marc Friedman. Query containment for data integration systems. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 67–75, 2000.

[MM07] Yde Venema Maarten Marx. Local variations on a loose theme: Modal logic and decidability. In *Finite Model Theory and Its Applications*, chapter 7, pages 371–429. Springer, June 2007.

[Mot06] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesitaet Karlsruhe, Karlsruhe, Germany, January 2006.

[MS87] D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995.

[Ném86] István Németi. Free algebras and decidability in algebraic logic. DSc. thesis, Mathematical Institute of The Hungarian Academy of Sciences, Budapest, 1986.

[Noy04] Natalya F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[NS97] Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal lp. In Dix et al. [DFN97], pages 421–430.

[OCE06] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*. AAAI Press, July 2006.

[OCE08] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. of Automated Reasoning*, 41(1):61–98, 2008.

[ORŠ10] Magdalena Ortiz, Sebastian Rudolph, and Mantas Šimkus. Query answering is undecidable in dls with regular expressions, inverses, nominals, and counting. Technical Report INFSYS RR-1843-10-03, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, April 2010.

[Ort08] Magdalena Ortiz. An automata-based algorithm for description logics around $\mathcal{SRIQ}$. In *Proc. of LANMR 2008*, volume 408 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/Vol-408/`, 2008.

[OŠE08a] Magdalena Ortiz, Mantas Šimkus, and Thomas Eiter. Conjunctive query answering in SH using knots. In Baader et al. [BLM08].

[OŠE08b] Magdalena Ortiz, Mantas Šimkus, and Thomas Eiter. Worst-case optimal conjunctive query answering for an expressive description logic without inverses. In Fox and Gomes [FG08], pages 504–510.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PH05] Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *J. of Logic, Language and Information*, 14(3):369–395, 2005.

[PH09] Ian Pratt-Hartmann. Data-complexity of the two-variable fragment with counting quantifiers. *Information and Computation*, 207(8):867–888, 2009.

[PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[Pra79] Vaughan R. Pratt. Models of program logics. In *20th Annual Symposium on Foundations of Computer Science, 29-31 October 1979, San Juan, Puerto Rico*, pages 115–122. IEEE, 1979.

[PSHH04] Peter Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language semantics and abstract syntax – W3C recommendation. Technical report, World Wide Web Consortium, February 2004. Available at `http://www.w3.org/TR/owl-semantics/`.

[PST00] Leszek Pacholski, Wiesław Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. on Computing*, 29(4):1083–1117, 2000.

[PSV06] Guoqiang Pan, Ulrike Sattler, and Moshe Y. Vardi. BDD-based decision procedures for the modal logic k. *Journal of Applied Non-Classical Logics*, 16(1-2):169–208, 2006.

[RKH08a] Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Cheap Boolean role constructors for description logics. In *Proceedings of the Eleventh European Workshop on Logics in Artificial Intelligence (JELIA 2008)*, volume 5293 of *Lecture Notes in Computer Science*, pages 362–374, 2008.

[RKH08b] Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Terminological reasoning in $\mathcal{SHIQ}$ with ordered binary decision diagrams. In Fox and Gomes [FG08], pages 529–534.

[Ros07] Riccardo Rosati. On conjunctive query answering in $\mathcal{EL}$. In *Proceedings of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/Vol-250/`, 2007.

[Sat00] Ulrike Sattler. Description logics for the representation of aggregated objects. In Werner Horn, editor, *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, pages 239–243. IOS Press, 2000.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. ISSN 1439-2275.

[Sch91] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[Sch94a] Andrea Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues.* PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1994.

[Sch94b] Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[SE89] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81:249–264, 1989.

[ŠE07] Mantas Šimkus and Thomas Eiter. FDNC: Decidable non-monotonic disjunctive logic programs with function symbols. In N. Dershowitz and A. Voronkov, editors, *Proceedings 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2007)*, number 4790 in LNCS, pages 514–530. Springer, 2007. Full paper Tech. Rep. INFSYS RR-1843-08-01, TU Vienna. `http://www.kr.tuwien.ac.at/research/reports/rr0801.pdf`.

[SFdB09] Inanç Seylan, Enrico Franconi, and Jos de Bruijn. Effective query rewriting with ontologies over DBoxes. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pages 923–925, 2009.

[SV01] Ulrike Sattler and Moshe Y. Vardi. The hybrid μ-calculus. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, pages 76–91, 2001.

[Tes01] Sergio Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, University of Manchester, Department of Computer Science, April 2001.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–192. Elsevier Science Publishers, 1990.

[Tob00] Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. of Artificial Intelligence Research*, 12:199–217, 2000.

[Tob01] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[Ull00] Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[Var82] Moshe Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, pages 137–146, 1982.

[Var85] Moshe Y. Vardi. The taming of converse: Reasoning about two-way computations. In R. Parikh, editor, *Proc. of the 4th Workshop on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 413–424. Springer, 1985.

[Var97] Moshe Y. Vardi. Why is modal logic so robustly decidable. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 31, pages 149–184. American Mathematical Society, 1997.

[Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[VW86] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986.