The Complexity of Description Logics with Concrete Domains

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von
Diplom-Informatiker Carsten Lutz
aus Hamburg

Berichter: Universitätsprofessor Dr.-Ing. Franz Baader
Privatdozent Dr. rer.-nat. Frank Wolter

Tag der mündlichen Prüfung: 8. März 2002

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

To my parents

Acknowledgements

First of all, I would like to thank Franz Baader, my thesis supervisor, for his support and advice. My work during the last three years was shaped by my teachers Franz Baader, Ulrike Sattler, and Frank Wolter. Without any of them, this thesis would not have been possible. Moreover, I owe to my colleagues from Aachen: Sebastian Brandt, Ralf Küsters, Ralf Molitor, Ramin Sadre, Stephan Tobies, and Anni-Yasmin Turhan. Working with you was a great pleasure. A special mention deserves Uli Sattler: I owe her countless beers for supporting me in just as many ways. Thanks a lot!

On the private side, most importantly I would like to thank Anja for bearing with the distance; Jakob for providing the best possible reason for starting to write my thesis; my parents for always supporting me without ever pushing; my brother Patrick for running the best "hotel" in Hamburg; and Daniel and Petra for never growing tired of me being tired. There are many other people from Aachen, Hamburg, and the scientific community who made the last three years worth living.

Contents

Chapter 1

Introduction

Description Logics (DLs) are knowledge representation formalisms that allow to represent and reason about conceptual and terminological knowledge in a structured and semantically well-understood manner. The basic entities for representing knowledge using DLs are so-called *concepts*, which correspond to formulas with one free variable in mathematical logic. Complex concepts are built from *concept names* (unary predicates), *role names* (binary predicates), and concept constructors. For example, using the basic propositionally closed Description Logic $\mathcal{ALC}$, we can describe fathers having at least one daughter using the concept

$$\mathsf{Male} \sqcap \forall \mathsf{child}.\mathsf{Human} \sqcap \exists \mathsf{child}.\mathsf{Female}.$$

In this example, Male, Human, and Female are concept names while child is a role name.

A major limitation of knowledge representation with Description Logics such as $\mathcal{ALC}$ is that "concrete qualities" of real world entities cannot be adequately represented. For example, if we want to describe husbands that are younger than their spouses, we can only do this by using an abstract, unary predicate such as $\mathsf{YoungerThanSpouse}$:

$$\mathsf{Male} \sqcap \exists \mathsf{spouse}.\mathsf{Female} \sqcap \mathsf{YoungerThanSpouse}.$$

However, this approach is by no means satisfactory since it does not really capture the semantics of what we were trying to say: we can at the same time demand that a husband is $\mathsf{YoungerThanSpouse}$ and 50 years old (using another predicate $\mathsf{50YearsOld}$), and that his spouse is $\mathsf{40YearsOld}$ without obtaining a contradiction. Other concrete qualities whose representation leads to similar problems include weights, temperatures, durations, and shape of real world entities.

To allow an adequate representation of concrete qualities, Description Logics can be extended by so-called *concrete domains*. A concrete domain consists of a set such as the natural numbers and a set of n-ary predicates such as the binary "$<$" with the obvious (fixed) extension. The integration of concrete domains into Description Logics is achieved by adding (i) a concrete domain-based concept constructor and (ii) a new sort of role names that allows to associate values from the concrete domain with abstract, logical objects. For example, the husbands that are younger than their spouse can be described using the concept

$$\mathsf{Male} \sqcap \exists \mathsf{spouse}.\mathsf{Female} \sqcap \exists \mathsf{age}, \mathsf{spouse}\ \mathsf{age}.<,$$

where spouse is a functional role, age is one of the "concrete domain roles", and $\exists \mathsf{age}, \mathsf{spouse}\ \mathsf{age}.<$ is an application of the concrete domain concept constructor, which must not be confused with the existential value restriction constructor used in the subconcept $\exists \mathsf{spouse}.\mathsf{Female}$—both constructors start with the same symbol "$\exists$".

Some form of concrete domain can be found in many Description Logic formalisms and in many implemented DL systems used in applications. However, although it is generally considered very important to determine the decidability and computational complexity of reasoning with Description Logics, the complexity of reasoning with concrete domains has never been formally investigated. In this thesis, we close this gap by determining the complexity of a many common Description Logics providing for concrete domains.

The remainder of this introduction is concerned with a more detailed description of the addressed research problems. Formal definitions and bibliographic references are deferred until Chapter 2.

Description Logics

A Description Logic usually consists of a concept language and so-called *TBox* and *ABox* formalisms. While the concept language is used for constructing complex concepts and roles, TBoxes allow the representation of terminological knowledge and of background knowledge from the application domain, and ABoxes store assertional knowledge about the current state of affairs in a particular "world". For example, the concept language $\mathcal{ALC}$ mentioned above provides the concept constructors negation ($\neg$), conjunction ($\sqcap$), disjunction ($\sqcup$), universal value restriction of roles ($\forall$), and existential value restriction of roles ($\exists$). As we shall see later, there exist many other concept and role constructors giving rise to more powerful concept languages.

Let us briefly describe the use of TBoxes and ABoxes for knowledge representation. There exist various flavors of TBoxes with vast differences in expressivity. However, even the weakest form of TBox, called *acyclic TBox*, can be used to represent terminological knowledge about the application domain. For example, we can assign the notion "younger husband" to the husbands describe above:

$$\mathsf{YoungerHusband} \doteq \mathsf{Male} \sqcap \exists \mathsf{spouse}.\mathsf{Female} \sqcap \exists \mathsf{age}, \mathsf{spouse}\ \mathsf{age}.<.$$

Intuitively, acyclic TBoxes can be thought of as (acyclic) macro definitions. Using a more expressive type of TBox, so called *general TBoxes*, complex background knowledge can be described. For example, we can express that all humans are either male or female and no human is both male and female:

$$\begin{array}{rcl} \mathsf{Human} & \doteq & \mathsf{Male} \sqcup \mathsf{Female} \\ \top & \doteq & \neg(\mathsf{Male} \sqcap \mathsf{Female}). \end{array}$$

Here, $\top$ is a special concept that, intuitively, stands for "everything".

In contrast to TBoxes, which represent knowledge of a general nature, ABoxes store knowledge about particular situations. If, for example, we want to describe that John is 42 years old and married to Mary, who is 40 years old, we can use the following ABox:

$$\begin{array}{rclcrcl} \mathsf{John} & : & \mathsf{Male} & \quad & \mathsf{John} & : & \exists \mathsf{age}.{=_{42}} \\ \mathsf{Mary} & : & \mathsf{Female} & & \mathsf{Mary} & : & \exists \mathsf{age}.{=_{40}} \\ (\mathsf{John}, \mathsf{Mary}) & : & \mathsf{spouse} & & & & \end{array}$$

Note that, if we add the fact $\mathsf{John} : \mathsf{YoungerHusband}$, then the resulting ABox describes an "impossible" world since spouse is functional and John is *not* younger than his (only) spouse Mary. Such inconsistencies can be detected by the Description Logic reasoning services. More precisely, standard reasoning services offered by Description Logics include the following:

- decide whether a concept C is satisfiable, i.e., whether it can have any instances (*concept satisfiability*);
- decide whether a concept C is subsumed by a concept D, i.e., whether every instance of C is necessarily also an instance of D (*concept subsumption*); and
- decide whether a given ABox is consistent, i.e., whether a world as described by the ABox may exist (*ABox consistency*).

All these reasoning tasks can be considered with and without reference to TBoxes. Several other reasoning services have been considered in the literature, but many of them can be reduced to the basic ones listed above. Note that the top-most task in the list corresponds to formula satisfiability in mathematical logic.

Apart from being a standard tool for knowledge representation and reasoning, Description Logics are used in several other application areas such as reasoning about entity relationship (ER) diagrams or reasoning about ontologies for the semantic web. In all these application areas, DLs are expected to meet the following two demands:

1. Reasoning should be decidable. Moreover, since it is desirable to implement reasoning services in DL systems with an acceptable run-time behavior, reasoning should be of low worst-case complexity.
2. The logic should be as expressive as possible to allow capturing all relevant aspects of the application domain.

The trade-off between complexity and expressivity induced by these two demands is one of the driving forces behind Description Logic research: the task is to develop logics that are sufficiently expressive, yet for which reasoning is of an acceptable complexity. During the last years, the reading of "acceptable complexity" has changed dramatically. In contrast to the early years of DL research, where researchers were striving to develop logics for which reasoning is tractable, nowadays there exist implementations of Description Logics for which reasoning is ExpTime-complete, and these implementations exhibit a reasonable run-time behavior on "real-world" problems. Nevertheless, the investigation of the worst-case complexity of reasoning with Description Logics is still one of the most important research topics in the field. Knowing the exact complexity class of a problem is necessary for devising optimal algorithms and it provides useful information concerning the run-time behavior to be expected from implemented DL reasoners.

Concrete Domains

Although several Description Logic systems also provided for some form of concrete domain, the first formal treatment was given by Baader and Hanschke in [1991a]. The authors propose to extend the basic Description Logic $\mathcal{ALC}$ with a concrete domain $\mathcal{D}$, thus obtaining the logic $\mathcal{ALC}(\mathcal{D})$. More precisely, $\mathcal{ALC}(\mathcal{D})$ extends $\mathcal{ALC}$ with (i) functional roles called *abstract features*, (ii) the above mentioned "concrete domain roles", which are called *concrete features* and are also required to be functional, and (iii) a concrete domain concept constructor. The difference between abstract and concrete features is that the former are just functional binary predicates in a standard first-order sense while the latter provide the link between abstract, logical objects and objects from the concrete domain. The concrete domain concept constructor provided by $\mathcal{ALC}(\mathcal{D})$ has the form $\exists u_1, \ldots, u_n.P$, where the u_i are *concrete paths*, i.e., sequences $f_1 \cdots f_k g$ of finitely many abstract features followed by a single concrete feature, and P is a predicate of arity n from the concrete domain $\mathcal{D}$. For example, using $\mathcal{ALC}(\mathcal{D})$ together with an appropriate concrete domain, we can describe people for whom the wage of the boss of the father is smaller than the wage of the mother:

$$\mathsf{Human} \sqcap \exists(\mathsf{father}\ \mathsf{boss}\ \mathsf{wage}), (\mathsf{mother}\ \mathsf{wage}).<$$

In this example, father, boss, and mother are abstract features while wage is a concrete feature and concrete paths are written in parenthesis. It is important to note that the concrete domain $\mathcal{D}$ can be viewed as a parameter to the logic $\mathcal{ALC}(\mathcal{D})$, i.e., Baader and Hanschke's approach is not committed to any particular concrete domain. Instead, their logic can be instantiated with any concrete domain suitable for representing knowledge from the application domain at hand. In the literature, one can find a broad spectrum of concrete domains that allow, e.g., to represent knowledge about numbers (ages, weights, temperatures), temporal relationships, or spatial extensions.

In their 1991 paper, Baader and Hanschke prove that reasoning with $\mathcal{ALC}(\mathcal{D})$ is decidable if the satisfiability of finite conjunctions of predicates from $\mathcal{D}$ is decidable (and, additionally, $\mathcal{D}$ satisfies some minor technical conditions). However, although concrete domains play an important role in DL research and $\mathcal{ALC}(\mathcal{D})$ can be regarded as the basic Description Logic with concrete domains, this fundamental result has never been further elaborated: first, the exact complexity of reasoning with $\mathcal{ALC}(\mathcal{D})$ has never been determined; second, extensions of $\mathcal{ALC}(\mathcal{D})$ with standard concept and role constructors not available in $\mathcal{ALC}$ or with TBox and ABox formalisms have only rarely been defined. Complexity results for such logics were not available. The main goal of this thesis is to explore the realm of Description Logics with concrete domains, focusing on decidability and computational complexity.

What results can we expect to find? In [Baader & Hanschke 1991a], it is shown that the extension of $\mathcal{ALC}(\mathcal{D})$ with the transitive closure role constructor yields a logic for which reasoning is undecidable. It is not too hard to see that, with some modifications, the proof can also be applied to the extension of $\mathcal{ALC}(\mathcal{D})$ with general TBoxes. Thus, reasoning with this logic is also undecidable. These results are rather surprising since, for the vast majority of Description Logics considered in the literature (such as $\mathcal{ALC}$), the addition of both the transitive closure constructor and general TBoxes makes reasoning ExpTime-hard, but does not lead to undecidability. Taking these observations together, we have obtained a first indication for the fact that there exist intricate interactions between concrete domains and other means of expressivity that are usually considered "harmless" w.r.t. their impact on complexity. And indeed, one main result of this thesis is that the extension of standard Description Logics with concrete domains in many cases leads to a dramatic increase in the complexity of reasoning, and in some cases even to undecidability.

The Extensions

For defining expressive extensions of $\mathcal{ALC}(\mathcal{D})$, one has several degrees of freedom: there exist many standard concept and role constructors whose addition to the concept language significantly increases the expressive power. Having TBoxes is also rather desirable, and in admitting them we can choose among several variants that differ considerably w.r.t. expressivity and their impact on the complexity of reasoning. An ABox formalism should be included if particular worlds are to be described. Finally, when proving decidability and complexity results, we may either focus on a particular concrete domain, or, alternatively, try to obtain general results that capture large classes of concrete domains.

Let us describe some of the concept and role constructors considered in this thesis in more detail:

1. The *inverse role constructor* can be applied to roles and abstract features. For example, it allows to describe children in families comprised of happy people:

   $$\forall \mathsf{child}^{-}.(\mathsf{Happy} \sqcap \forall \mathsf{child}.\mathsf{Happy})$$

   Here, the expression child^{-} denotes the inverse of the child role.

2. *Qualifying number restrictions* are an important type of concept constructor and incorporate "counting" into Description Logics. Using number restrictions, we can describe persons having at most 4 children of which at least 2 are daughters:

   $$\mathsf{Human} \sqcap \leq 4\,\mathsf{child}.\top \sqcap \geq 2\,\mathsf{child}.\mathsf{Female}.$$

3. The *role conjunction constructor* allows to relate objects by more than a single role. For example, with this constructor we can talk about people whose spouse is also their boss:

   $$\mathsf{Human} \sqcap \exists(\mathsf{spouse} \sqcap \mathsf{boss}).\mathsf{Human}.$$

4. Using the *feature agreement* and *feature disagreement* concept constructors, it is possible to enforce (dis)agreements between successors of sequences of abstract features. For example, the following concept describes persons whose parents are married:

$$\mathsf{mother\ married\text{-}to} \downarrow \mathsf{father}$$

while the following one describes persons with divorced parents:

$$\exists \mathsf{mother}.\neg\exists\mathsf{spouse}.\top \ \sqcup \ \mathsf{mother\ married\text{-}to} \uparrow \mathsf{father}.$$

Several other concept and role constructors will be introduced in Chapter 2. The feature (dis)agreement constructors deserve some special attention. Similar to the concrete domain constructor, they take sequences of features as arguments. Apart from this obvious syntactic similarity, we shall see that logics including feature (dis)agreements are amenable to similar algorithmic techniques as logics including the concrete domain constructor. Moreover, we will find that, in many cases, Description Logics with concrete domains have the same complexity as the corresponding logics that can be obtained by admitting feature (dis)agreements instead of concrete domains. Because of this interesting connection, the complexity of DLs with feature (dis)agreements will be an important side-track in this thesis.

Some particular concrete domains will play a prominent role throughout this thesis. More specifically, the central lower bounds for extensions of $\mathcal{ALC}(\mathcal{D})$ rely on the fact that the concrete domain $\mathcal{D}$ allows to represent the natural numbers and offers (at least) the following predicates: a unary predicate expressing equality to zero, a binary equality predicate, a binary predicate for addition with the constant 1, and a ternary predicate expressing addition of two numbers. In what follows, such concrete domains are called *arithmetic*. It is important to note that most concrete domains proposed for knowledge representation such as the one in [Baader & Hanschke 1992] are in fact arithmetic and, thus, the obtained lower bounds are of practical relevance. The above mentioned undecidability result for $\mathcal{ALC}(\mathcal{D})$ extended with a transitive closure constructor also requires the concrete domain $\mathcal{D}$ to be arithmetic. However, there exist interesting concrete domains that are not arithmetic, and to which the main lower bounds obtained in this thesis do thus not apply. An important example is the concrete domain, which is comprised of the set of all intervals over some temporal structure and a binary predicate for each of the well-known Allen interval relations. Description Logics equipped with this temporal concrete domain are well-suited for interval-based conceptual temporal reasoning. As we shall see, there exist rather powerful Description Logics that are undecidable if extended with arithmetic concrete domains, but decidable (in ExpTime) if extended with the concrete domain $\mathtt{I}$.

Structure of the Thesis

Apart from the introduction, the preliminaries, and the conclusion, this thesis can be divided into three parts. In the first part (Chapter 3), we establish the basic results that settle the complexity of reasoning with $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}(\mathcal{D})$, where the latter is $\mathcal{ALC}(\mathcal{D})$ extended with feature (dis)agreements. In the second part (Chapters 4 and 5), we investigate the complexity of reasoning with several natural extensions of $\mathcal{ALC}(\mathcal{D})$. The obtained results are not limited to a particular concrete domain but do apply to all arithmetic concrete domains. Finally, in the third part (Chapter 7) we consider Description Logics equipped with concrete domains and general TBoxes. The main result of this part concerns the above mentioned temporal concrete domain $\mathtt{I}$.

More precisely, this thesis is structured as follows:

- Chapter 2 contains the preliminaries. We start with introducing the basic Description Logic $\mathcal{ALC}$ together with the relevant reasoning problems, several variants of TBoxes, and ABoxes. We also define several additional concept and role constructors that may be used to devise DLs more powerful than $\mathcal{ALC}$. Next, we introduce the basic Description Logic providing for concrete domains, $\mathcal{ALC}(\mathcal{D})$, and discuss some important, concrete domain-related extensions of $\mathcal{ALC}(\mathcal{D})$. Finally, we define several example concrete domains, including $\mathtt{I}$. In this context, we make a short excursion to so-called unary concrete domains and a restricted version of the concrete domain constructor and show that, in this simple setting, the complexity of reasoning is rather easily determined.

- In Chapter 3, we prove the basic result that reasoning with $\mathcal{ALCF}(\mathcal{D})$ is PSpace-complete if reasoning with the concrete domain $\mathcal{D}$ is in PSpace. Since reasoning with $\mathcal{ALC}$ is also PSpace-complete, this means that adding both concrete domains and feature (dis)agreements to $\mathcal{ALC}$ does not increase the complexity of reasoning. The result is established by using a tableau algorithm and adapting the so-called trace technique to Description Logics with concrete domains.

- Chapter 4 investigates the impact of acyclic TBoxes on the complexity of reasoning with Description Logics. This chapter, which is not explicitly concerned with concrete domains, can be seen as a preparation for the succeedings chapter, where (among other things) the combination of concrete domains and acyclic TBoxes is investigated. We show how the standard PSpace tableau algorithm for $\mathcal{ALC}$ can be modified to take into account acyclic TBoxes such that it can still be executed in polynomial space. A rule of thumb is given, which indicates that the described modification technique can be applied to a large class of Description Logics. Thus, we show that, in many cases, acyclic TBoxes do not increase the complexity of reasoning. To demonstrate that this is not always the case, we prove that, quite surprisingly, the extension of $\mathcal{ALC}$ with feature (dis)agreements and acyclic TBoxes is NExpTime-complete.

- In Chapter 5, we identify five seemingly simple extensions of $\mathcal{ALC}(\mathcal{D})$ for which reasoning is NExpTime-hard if the concrete domain $\mathcal{D}$ is arithmetic. These extensions are (i) with acyclic TBoxes, (ii) with role conjunction, (iii) with inverse roles, (iv) with a generalized concrete domain concept constructor, and (iv) with a concrete domain role constructor. The obtained hardness-results are rather surprising since the considered extensions are usually considered "harmless" w.r.t. complexity and, thus, a leap in complexity from PSpace-complete to

NExpTime-complete was not to be expected. We also establish a corresponding upper bound stating that reasoning with all five extensions of $\mathcal{ALC}(\mathcal{D})$ together is in NExpTime if reasoning with $\mathcal{D}$ is in NP. Finally, we consider the corresponding extensions of $\mathcal{ALCF}$ ($\mathcal{ALC}$ with feature (dis)agreements) and find that, in most cases, the complexity parallels that of the extensions of $\mathcal{ALC}(\mathcal{D})$.

- Chapter 6 is concerned with the combination of general TBoxes and concrete domains. First, we prove that reasoning with $\mathcal{ALC}(\mathcal{D})$ and general TBoxes is undecidable if $\mathcal{D}$ is arithmetic. Then, we demonstrate that, despite this general, discouraging result, there exist interesting concrete domains whose combination with general TBoxes does not lead to undecidability. More precisely, we investigate the extension of $\mathcal{ALC}(\mathtt{I})$—i.e. $\mathcal{ALC}(\mathcal{D})$ instantiated with the interval-based concrete domain $\mathtt{I}$—with general TBoxes. Using an automata-based approach, we are able to show that reasoning with this logic is decidable and ExpTime-complete. Several examples from the application domain of process engineering demonstrate that $\mathcal{ALC}(\mathtt{I})$ with general TBoxes is a powerful tool for temporal conceptual reasoning.
- In Chapter 7, we give an overview over the obtained results and sketch some perspectives for future research.

Some of the results in this thesis have previously been published: the PSpace upper bounds given in Chapter 3 appeared in [Lutz 1999b]; the modification technique for extending tableau algorithms to acyclic TBoxes and the NExpTime-completeness result for $\mathcal{ALCF}$ with acyclic TBoxes from Chapter 4 have been published in [Lutz 1999a]; the NExpTime-lower bounds proved in Chapter 5 already appeared in [Lutz 2000; 2001b]; finally, the automata-based decision procedure from Chapter 6 was published as [Lutz 2001a].

Chapter 2

Preliminaries

In this chapter, we introduce Description Logics in more detail. The chapter starts with a short introduction to the field, followed by a formal definition of the basic propositionally closed DL $\mathcal{ALC}$ and the inference problems that are relevant for this thesis. We discuss several extensions of $\mathcal{ALC}$ that are usually considered in the literature and will resurface in subsequent chapters. Additionally, several variants of TBox and ABox formalisms for Description Logics are presented. We then focus on Description Logics with concrete domains and give a detailed overview over this family of DLs. The chapter closes with a presentation of some example concrete domains.

For all formalisms defined in this chapter, we briefly discuss the known complexity results. A basic acquaintance with the standard complexity classes PTime, NP, PSpace, ExpTime, NExpTime, etc. is assumed. Nevertheless, some short conventional remarks are in order: completeness and hardness are defined on the basis of polynomial time reductions as usual. We abstract from issues such as the encoding of the problem at hand (in order to make it accessible for a Turing machine). This can be done without loss of generality since the encoding can always be done with at most a polynomial blowup in size if at least a binary input alphabet for Turing machines is assumed. Moreover, the complexity classes used are oblivious to such blowups. Since there exist at least two different definitions of the complexity class ExpTime, we make its definition more precise: a language L is in ExpTime iff there exists a Turing machine M and a constant k such that M accepts L terminating after at most 2^{n^k} steps if started on an input of size n. The definitions of NExpTime and ExpSpace are analogous. For a thorough introduction to complexity theory, we refer the reader to [Papadimitriou 1994].

2.1 Description Logics

Historically, Description Logics evolved from *semantic networks* [Quillian 1968] and *frame systems* [Minsky 1975], mainly to satisfy the need of giving a formal semantics to these formalisms (see [Baader *et al.* 2002b] for more historical notes). Nowadays, the field is situated on the intersection of mathematical and philoshopical logic, knowledge representation, and database theory. The most important tasks in Description Logic research may be described as follows:

1. Devise logical formalisms that are, on the one hand, sufficiently expressive to deal with interesting applications and, on the other hand, have reasoning problems that are at least decidable, but preferably in as small a complexity class as possible. This trade-off between expressivity and complexity gave rise to the definition of numerous Description Logics and the investigation of the complexity of their reasoning problems, e.g. [Baader & Hanschke 1991a; Donini *et al.* 1997; Baader *et al.* 1996; Calvanese *et al.* 1998a; Horrocks *et al.* 2000a]. Any Description Logic should be equipped with a decision procedure that has the potential to be optimized for implementation in a DL reasoner (see 2). Rather often, this decision procedure is a tableau algorithm since it has turned out that this kind of algorithms [D'Agostino *et al.* 1999; Baader & Sattler 2000] is particularly amenable to optimizations [Horrocks 1997; Horrocks & Patel-Schneider 1999].

2. Build DL systems that implement inference services for the DL formalisms mentioned under 1 and "behave reasonably well" on real-world application problems. The main challenge is to find optimization strategies for the algorithms that have been devised in a more theoretical setting to make them usable for implementation purposes. This area has been quite active since at least 1997, when the development of the seminal system FaCT showed that even DLs that are hard for complexity classes as large as ExpTime can be implemented in efficient DL systems [Horrocks 1997]. Since then, several expressive and efficient Description Logic systems have been built [Horrocks & Patel-Schneider 1999; Horrocks 1999; Patel-Schneider 1999; Haarslev & Möller 2000a].

3. Apply the logical formalisms from 1 and their implementations from 2 in appropriate application areas. The "classical" application of DLs is in knowledge representation, where they form an own subfield and are closely related to various other KR formalisms [Brachman 1978; Baader 1999; Baader *et al.* 1999]. The second important application area is for reasoning about database problems: Description Logics have been used for reasoning about, e.g., entity relationship diagrams [Calvanese *et al.* 1998b], semistructured data [Calvanese *et al.* 2000a], and query answering using views [Calvanese *et al.* 2000b]. Other application areas that have been considered include the use of DLs for providing a unifying framework for class-based formalisms in computer science [Calvanese *et al.* 1998b] and, more recently, for the modelling of ontologies for the semantic web [Fensel *et al.* 2000].

This thesis mainly addresses issues from 1: several new expressive Description Logics with concrete domains are introduced. For all these new logics, as well as for several existing ones, the complexity of reasoning is determined. In most cases, we also give (worst-case) optimal decision procedures using techniques that are commonly believed to be amenable to optimization strategies. Hence, the proposed algorithms can be expected to behave quite well in practice. The motivation for considering the concrete domain family of Description Logics comes from knowledge representation, as we shall see in Section 2.3. In the following section, we start the formal presentation of Description Logics by introducing $\mathcal{ALC}$, which is a fragment of all DLs considered in subsequent chapters.

2.1.1 Introducing $\mathcal{ALC}$

The Description Logic $\mathcal{ALC}$, which was first described by Schmidt-Schauß and Smolka [1991], is the "smallest" DL that is propositionally closed, i.e., that provides for all Boolean connectives. More precisely, $\mathcal{ALC}$ concepts are built from the Boolean connectives and so-called existential and universal value restrictions. Every Description Logic used in this thesis provides for at least these five concept constructors.

Definition 2.1 ($\mathcal{ALC}$ syntax). Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be disjoint and countably infinite sets of concept and role names. The set of $\mathcal{ALC}$-concepts is the smallest set such that

1. every concept name $A \in \mathsf{N_C}$ is an $\mathcal{ALC}$-concept and
2. if C and D are $\mathcal{ALC}$-concepts and $R \in \mathsf{N_R}$, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, and $\forall R.C$ are $\mathcal{ALC}$-concepts.

We use $\top$ as an abbreviation for some fixed propositional tautology such as $A \sqcup \neg A$, $\bot$ for $\neg\top$, $C \rightarrow D$ for $\neg C \sqcup D$, and $C \leftrightarrow D$ for $(C \rightarrow D) \sqcap (D \rightarrow C)$. ◇

We sometimes identify the name of a concept language and the set of all its concepts, e.g., we denote the set of all $\mathcal{ALC}$-concepts by $\mathcal{ALC}$. Throughout this thesis, we denote concept names by A and B, concepts by C and D, and roles by R and S.

The meaning of concepts is fixed by a Tarski-style semantics: concepts are interpreted as unary predicates over a set called the *domain* and roles as binary predicates over the domain. Together with the semantics, we introduce the standard reasoning problems on concepts.

Definition 2.2 ($\mathcal{ALC}$ semantics). An $\mathcal{ALC}$-*interpretation* $\mathcal{I}$ is a pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta_\mathcal{I}$ is a non-empty set called the *domain*, and $\cdot^\mathcal{I}$ is an *interpretation function* that maps

- every concept name A to a subset $A^\mathcal{I}$ of $\Delta_\mathcal{I}$ and
- every role name R to a binary relation $R^\mathcal{I}$ over $\Delta_\mathcal{I}$.

The interpretation function is extended to complex concepts as follows:

$$
\begin{aligned}
(\neg C)^\mathcal{I} &:= \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(C \sqcap D)^\mathcal{I} &:= C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} &:= C^\mathcal{I} \cup D^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &:= \{d \mid \text{there is some } e \in \Delta_\mathcal{I} \text{ such that } (d,e) \in R^\mathcal{I} \text{ and } e \in C^\mathcal{I}\} \\
(\forall R.C)^\mathcal{I} &:= \{d \mid \text{for all } e \in \Delta_\mathcal{I}, (d,e) \in R^\mathcal{I} \text{ implies } e \in C^\mathcal{I}\}
\end{aligned}
$$

$\mathsf{Engine} \sqcap \exists \mathsf{part}.\mathsf{GlowPlug} \sqcap \forall \mathsf{fuel}.\mathsf{Diesel}$
$\mathsf{Process} \sqcap \forall \mathsf{subprocess}.\neg(\mathsf{Noisy} \sqcup \mathsf{Dangerous})$

Figure 2.1: Two example concepts.

Let $\mathcal{I}$ be an interpretation and $d, e \in \Delta_{\mathcal{I}}$. Then e is called an *R-successor of d in $\mathcal{I}$* iff $(d, e) \in R^{\mathcal{I}}$.

A concept C is *satisfiable* iff there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a *model* of C. A concept D *subsumes* a concept C (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$. Two concepts are *equivalent* (written $C \equiv D$) iff they mutually subsume each other. $\diamond$

The reasoning problems on concepts most often considered in DL research are satisfiability and subsumption. Note that, in a Description Logic providing the Boolean connectives, subsumption can be reduced to (un)satisfiability since $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable. The converse also holds since C is unsatisfiable iff C is subsumed by $\bot$. Moreover, subsumption can be reduced to equivalence since $C \sqsubseteq D$ iff $C \sqcap \neg D \equiv \bot$ (and equivalence is defined in terms of subsumption). These simple observations imply that, when establishing lower and upper complexity bounds or proving decidability and undecidability, we may restrict ourselves to satisfiability since all the obtained results are also valid for subsumption and equivalence (note, however, that complexity classes have to be complemented).

It should be noted that there exist reasoning problems on concepts other than the ones mentioned above. The above problems are often called "standard" inferences while reasoning problems such as the computation of the least common subsumer of two concepts or unification of two concept patterns are frequently referred to as "non-standard" inferences [Küsters 2001; Molitor 2000]. However, in this thesis we are only concerned with standard inferences.

Let us now briefly demonstrate the use of $\mathcal{ALC}$ for knowledge representation. Two example concepts can be found in Figure 2.1, where—as always in the sequel—concepts start with an uppercase letter and roles with a lowercase one. Like most examples in this thesis, they are concerned with technical devices and production processes for such devices. Intuitively, the first concept describes a diesel engine: an engine that has a part that is a glow plug, and which uses only diesel fuel. The second concept describes a production process all of whose subprocesses are neither noisy nor dangerous.

The complexity of reasoning with $\mathcal{ALC}$ has first been determined by Schmidt-Schauß and Smolka [1991]:[1] satisfiability (and hence also subsumption and equivalence) of $\mathcal{ALC}$-concepts is PSPACE-complete. The lower bound is proved by reducing satisfiability of quantified Boolean formulas (QBF) [Stockmeyer & Meyer 1973] to $\mathcal{ALC}$-concept satisfiability and the upper bound is given by a tableau-like algorithm. In the early days of Description Logics, the common opinion was that this seemingly high complexity precludes $\mathcal{ALC}$ from being used in knowledge representation and reasoning

[1] Although this result was already known in the field of Modal Logic, see below.

systems. However, the $\mathcal{KRIS}$ system [Baader & Hollunder 1991a] demonstrated that it is possible to implement $\mathcal{ALC}$-concept satisfiability in an efficient way and, several years later, it turned out that efficient implementations can be devised even for ExpTime-complete DLs [Horrocks & Patel-Schneider 1999]. Today, every "modern" Description Logic system implements at least $\mathcal{ALC}$.

There exists a very close connection between Description Logics and various other logics investigated in the areas of of mathematical logic and philosphical logic. This connection can be used to transfer complexity and (un)decidability results between the two fields. Since we will occasionally use results from, e.g., Modal Logic, and, moreover, the relationship between DLs and mathematical logic is of general importance, we give a short introduction to this issue. The close connection between Modal Logic (ML) [Blackburn *et al.* 2001; Chagrov & Zakharyaschev 1996] and Description Logics was first observed by Schild in [1991] and later investigated in more detail in [Giacomo & Lenzerini 1994; Schild 1994; Areces & de Rijke 2001]. Schild notes that the basic DL $\mathcal{ALC}$ is a notational variant of K_ω, i.e., the minimal normal Modal Logic K [Chagrov & Zakharyaschev 1996] with infinitely many modal operators: existential value restrictions can be seen as diamond operators (one for each role), and universal value restrictions can be viewed as box operators. Kripke structures and DL interpretations can straightforwardly be translated into one another. Hence, DL concept satisfiability is just ML formula satisfiability. This clearly implies that an alternative proof of the PSpace-completeness of $\mathcal{ALC}$ can be given by using the correspondence between $\mathcal{ALC}$ and K_ω together with the known PSpace-completeness of K_ω [Ladner 1977; Spaan 1993a]. Most extensions of $\mathcal{ALC}$ also have a counterpart in Modal Logic, and we will keep track of this correspondence throughout this chapter.

However, Modal Logic is not the only close relative of Description Logics: there exists a very important connection to (several decidable fragments of) First Order Logic (FO). The "standard translation" of Modal Logic [van Benthem 1983; Vardi 1997] to FO can also be applied to Description Logics which was first done by Borgida [1996] and later refined in [Lutz *et al.* 2001a; 2001b]. More precisely, a function $\cdot^*$ that takes $\mathcal{ALC}$-concepts to FO-formulas with one free variable x can be defined inductively as follows:

$$
\begin{array}{rcl}
A^* & := & P_A(x) \\
(\neg C)^* & := & \neg(C^*) \\
(C \sqcap D)^* & := & (C^* \wedge D^*) \\
(C \sqcup D)^* & := & (C^* \vee D^*) \\
(\exists R.C)^* & := & (\exists y(P_R(x,y) \wedge \exists x(x = y \wedge C^*))) \\
(\forall R.C)^* & := & (\forall y(P_R(x,y) \rightarrow \forall x(x = y \rightarrow C^*)))
\end{array}
$$

where x and y are FO variables, the P_A are unary predicates, and the P_R are binary predicates. Models of $\mathcal{ALC}$-concepts C and first order models of C^* can easily be translated back and forth (in particular without a blowup in size). Let $\mathcal{ALC}^*$ be the range of $\cdot^*$, i.e., the fragment of FO that is obtained by the translation of $\mathcal{ALC}$-concepts. At least two very important properties of $\mathcal{ALC}^*$ can be observed.

Firstly, every $\varphi \in \mathcal{ALC}^*$ contains at most two variables. Hence, $\mathcal{ALC}^*$ is a fragment of FO^2, the *two-variable fragment* of First Order Logic which is known to be NExpTime-complete [Grädel *et al.* 1997].[2] This fact can be used to transfer properties, such as decidability, the NExpTime upper bound, and the bounded modely property, from FO^2 to $\mathcal{ALC}$ and several extensions of $\mathcal{ALC}$. For example, it is known that every satisfiable FO^2-formula φ has a model whose size is at most exponential in the length of φ [Grädel *et al.* 1997]. Since FO^2-models can be translated into $\mathcal{ALC}$-interpretations without a blowup in size, $\mathcal{ALC}$ also has this *bounded model property* (and so do many of its extensions).

Secondly, in each $\varphi \in \mathcal{ALC}^*$, the quantifiers appear only in the form

$$\forall x(P(x,y) \rightarrow \psi(x)) \text{ and } \exists x(P(x,y) \wedge \psi(x))$$

where P may be equality. This observation led to the definition of the *guarded fragment* of First Order Logic, in which quantifiers must appear in a restricted way that can be obtained from the above quantification schema by generalization [Andréka *et al.* 1998]. Again, results concerning the guarded fragment, such as the ExpTime upper bound (if a bounded arity of predicates is assumed) [Grädel 1999], can be transfered to $\mathcal{ALC}$ and several of its extensions.

There also exist other logics like the *μ-calculus* [Kozen 1982; Sattler & Vardi 2001] or the *μ-guarded fragment* [Grädel & Walukiewicz 1999] which may be used to transfer results to Description Logics [Schild 1994]. However, we close our contemplation of the connection between DL and mathematical logic at this point and only sometimes return to Modal Logic in what follows.

Before we turn our attention towards extensions of $\mathcal{ALC}$, we introduce some common notions and a normal form for $\mathcal{ALC}$-concepts that will frequently be used in subsequent chapters. Like many notions introduced in this section, the normal form can also be used for extensions of $\mathcal{ALC}$ and we will do so without further notice.

Definition 2.3 (NNF). An $\mathcal{ALC}$-concept C is in *negation normal form (NNF)* iff negation occurs only in front of concept names. Every $\mathcal{ALC}$-concept can be converted into an equivalent one in NNF by exhaustively applying the following rewrite rules:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\neg\neg C & \rightsquigarrow & C & & & \\
\neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D & \neg(\exists R.C) & \rightsquigarrow & \forall R.\neg C \\
\neg(C \sqcup D) & \rightsquigarrow & \neg C \sqcap \neg D & \neg(\forall R.C) & \rightsquigarrow & \exists R.\neg C
\end{array} \qquad \diamond$$

The *length* $|C|$ of a concept C is defined as the number of symbols used to write down C, e.g. the length of the concept $\forall R.(C \sqcap D)$ is 8. Observe that, if a concept C is converted into an equivalent concept D in NNF using the above rewrite rules, then the number of rule applications is bounded by $|C|$. Moreover, $|D|$ is linear in $|C|$ which means that, for proving complexity results, we can w.l.o.g. assume that concepts are in NNF.

The set of *subconcepts* of an $\mathcal{ALC}$-concept C is denoted by $\mathsf{sub}(C)$ and defined inductively in the obvious way (in particular $C \in \mathsf{sub}(C)$). For any set S, we use $|S|$ to denote its cardinality. We clearly have

[2]More precisely, FO^2 with equality. It is also possible to translate $\mathcal{ALC}$-concepts to FO^2 without using equality [Borgida 1996].

1. $|\mathsf{sub}(C)| \leq |C|$ and

2. if C is in NNF, then all $C' \in \mathsf{sub}(C)$ are also in NNF

The first point is important for proving complexity results while the second is crucial for devising decision procedures that take advantage of negation normal form since these decision procedures usually work by breaking down the input concepts into subconcepts in some appropriate way.

A notion that will be useful for proving termination of decision procedures is the *role depth* of concepts. The role depth of a concept C is the maximum nesting depth of exists and value restrictions in C. For example, the role depth of $\exists R.C \sqcap \forall S.(D \sqcap \exists R.C)$ is 2.

2.1.2 Extensions of $\mathcal{ALC}$

Since the expressive power of $\mathcal{ALC}$ is too weak for many applications, various extensions of this basic formalism have been considered. In this section, we introduce extensions of $\mathcal{ALC}$ that are relevant for the remaining chapters. The introduction of concrete domains, however, is delayed until Section 2.3, which is devoted entirely to this topic.

Extensions of $\mathcal{ALC}$ can be divided into (at least) three groups:

1. restrictions of interpretations,

2. additional concept constructors, and

3. role constructors.

Let us first consider restrictions of interpretations. One can, for example, enforce that the set of roles $\mathsf{N_R}$ is partitioned into two infinite sets $\mathsf{N_{R1}}$ and $\mathsf{N_{R2}}$, and that every role $R \in \mathsf{N_{R2}}$ is interpreted by a transitive relation $R^\mathcal{I}$. This extension of $\mathcal{ALC}$ by *transitive roles* is called $\mathcal{ALC}_{R^+}$ [Sattler 1996]. Alternatively, we may demand that each role $R \in \mathsf{N_{R2}}$ is interpreted by a partial function $R^\mathcal{I}$ [Baader *et al.* 1993; Borgida & Patel-Schneider 1994]. Such *abstract features* are denoted by the symbol f and will play an important role in the context of concrete domains.[3]

Figure 2.2 lists the syntax and semantics of additional concept and role constructors, where $\sharp R^\mathcal{I}(d, C)$ is an abbreviation for $|\{e \mid (d,e) \in R^\mathcal{I} \cap C^\mathcal{I}\}|$. The figure also indicates how the names of the resulting extensions of $\mathcal{ALC}$ are derived. For example, $\mathcal{ALC}$ extended with inverse roles and qualifying number restrictions is denoted by $\mathcal{ALCQ}^-$. Note that, in the presence of qualifying number restrictions, one can define "locally" functional roles by writing $\leq 1 R.\top$. In contrast, the functionality of abstract features is of a global nature. Feature (dis)agreements also deserve some special attention. The arguments p_1 and p_2 are sequences $f_1 \cdots f_k$ of abstract features. Such *abstract paths* are in the following denoted by the symbol p, and their semantics is defined in the obvious way: if $p = f_1 \cdots f_k$ is an abstract path, then $p^\mathcal{I}$ is defined as the composition of the abstract features $f_k^\mathcal{I}(\cdots(f_1^\mathcal{I}(\cdot)))$. Obviously, it is

[3]Sometimes, abstract features are called *attributes* [Baader & Hanschke 1992].

Name	Syntax	Semantics	Symbol
Role Conjunction	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$	$\cdot^{\sqcap}$
Inverse Role	R^-	$\{(d,e) \mid (e,d) \in R^{\mathcal{I}}\}$	$\cdot^-$
Qualifying number restrictions	$\geq nR.C$ $\leq nR.C$	$\{d \mid \sharp R^{\mathcal{I}}(d,C) \geq n\}$ $\{d \mid \sharp R^{\mathcal{I}}(d,C) \leq n\}$	$\mathcal{Q}$
Feature agreement and disagreement	$p_1{\downarrow}p_2$ $p_1{\uparrow}p_2$	$\{d \mid \exists e \in \Delta_{\mathcal{I}} : p_1^{\mathcal{I}}(d) = p_2^{\mathcal{I}}(d) = e\}$ $\{d \mid \exists e_1, e_2 \in \Delta_{\mathcal{I}} : p_1^{\mathcal{I}}(d) = e_1, p_2^{\mathcal{I}}(d) = e_2, e_1 \neq e_2\}$	$\mathcal{F}$

Figure 2.2: Description Logic role and concept constructors.

$\mathsf{Engine} \sqcap \geq 4\ \mathsf{part.GlowPlug} \sqcap \leq 4\ \mathsf{part.GlowPlug} \sqcap \forall \mathsf{fuel.Diesel}$

$\mathsf{DieselEngine} \sqcap \exists \mathsf{part}^-.\mathsf{Car}$

$\mathsf{Process} \sqcap (\mathsf{drill\text{-}subprocess\ workpiece} \downarrow \mathsf{paint\text{-}subprocess\ workpiece})$

$\mathsf{Process} \sqcap \forall(\mathsf{subprocess} \sqcap \mathsf{subprocess}^-).\bot$

Figure 2.3: Some example concepts.

implicitly assumed that the logic $\mathcal{ALCF}$ provides for (an infinite number of) abstract features. In $\mathcal{ALCF}$ and its extensions, we often write $\exists p.C$ ($\forall p.C$) as an abbreviation for $\exists f_1.\cdots.\exists f_k.C$ ($\forall f_1.\cdots.\exists f_k.C$) if $p = f_1 \cdots f_k$ is an abstract path.

Figure 2.3 gives some example concepts illustrating the expressive power provided by extensions of $\mathcal{ALC}$.[4] The first one is an $\mathcal{ALCQ}$-concept that refines the first concept from Figure 2.1 by demanding that diesel engines must have exactly four glow plugs. Note that it would be very natural to demand that the part relation is transitive, i.e., to use the DL $\mathcal{ALCQ}_{R^+}$ for this example [Sattler 2000]. The second example is an $\mathcal{ALC}^-$-concept (or $\mathcal{ALC}^-_{R^+}$-concept) describing diesel engines that are parts of cars. The third example uses $\mathcal{ALCF}$ to describe processes that have exactly one drilling subprocess and exactly one painting subprocess both involving only one and the same workpiece. Finally, the fourth example uses an $\mathcal{ALC}^{-,\sqcap}$-concept to describe processes for which there exists no other process that is both a subprocess and a superprocess.

Let us briefly review Description and Modal Logics related to the extensions of $\mathcal{ALC}$ just introduced and then discuss their complexity. The role conjunction constructor is frequently used in Description Logics such as $\mathcal{ALCNR}$ and the logic $\mathcal{ALB}$ which underlies the MSpass system [Donini *et al.* 1997; Hustadt & Schmidt 2000]. In Modal Logic, this constructor appears e.g. in Boolean Modal Logic, the extension of K_ω with the Boolean operators on modal parameters [Gargov *et al.* 1987; Lutz & Sattler 2001], where *modal parameters* are the Modal Logic counterpart of roles (other names from the literature are, e.g., modal indices and programs). The most prominent occurrence

[4]Here, as in the following, we refrain from formally defining the notion "expressive power" since this is out of the scope of this thesis. See, e.g., [Baader 1990a; Areces & de Rijke 1998] for a formalization.

of the inverse of modal parameters is probably in temporal logics with future and past operators [Goldblatt 1987], but also in some variants of Propositional Dynamic Logics [Harel 1984]. In Description Logics, the inverse role constructor is one of the most common role constructors and can be found, e.g., in [Calvanese *et al.* 1998a; Horrocks *et al.* 2000a]. The same references, together with [Hollunder & Baader 1991; Baader *et al.* 1996] are appropriate for qualifying number restrictions. Transitive roles can, e.g., be found in [Sattler 1996; Calvanese *et al.* 1998a; Horrocks *et al.* 2000a]. In Modal Logic, qualifying number restrictions are called graded modalities [Fine 1972], while transitive roles appear in the form of the Modal Logic $\mathsf{K4}$ [Chagrov & Zakharyaschev 1996]. It appears that feature (dis)agreements have not been considered in Modal Logic. In Description Logics, they have been introduced as a variant of the more powerful (dis)agreements on sequences of (non-functional) roles called *role value maps* [Brachman & Schmolze 1985]. Unlike role value maps, which usually lead to undecidable reasoning problems [Schmidt-Schauß 1989], Description Logics with feature (dis)agreements are often decidable [Hollunder & Nutt 1990; Borgida & Patel-Schneider 1994].

Concerning the complexity of reasoning, several results are known for the extensions of $\mathcal{ALC}$ introduced in this section:

- Tobies [2001b] shows that satisfiability of $\mathcal{ALCQ}^{-,\sqcap}$-concepts (or rather the Modal Logic equivalent of this problem) is PSPACE-complete. Thus, any logic "between" $\mathcal{ALC}$ and $\mathcal{ALCQ}^{-,\sqcap}$ also has this property;
- Horrocks et al. [2000a] prove that the satisfiability of $\mathcal{ALCN}^{-}_{R^+}$-concepts is in PSPACE, where $\mathcal{ALCN}^{-}_{R^+}$ is $\mathcal{ALC}$ extended with transitive roles, the inverse role constructor, and a weak form of number restrictions; and
- Hollunder and Nutt [1990] prove PSPACE-completeness of $\mathcal{ALCF}$-concept satisfiability.

Hence, many of the introduced extensions of $\mathcal{ALC}$ are in the same complexity class as $\mathcal{ALC}$ itself, albeit more expressive. This situation changes dramatically when TBoxes and concrete domains come into play. Note that (to the best of our knowledge) not much is known about the complexity of reasoning with extensions of $\mathcal{ALCF}$. We will determine the complexity of several such extensions in subsequent chapters. For example, in Section 5.5.2, we shall show that $\mathcal{ALCF}^{-}$ is undecidable.

2.2 TBox and ABox Formalisms

Most Description Logics are not only comprised of a concept language but, additionally, provide for a TBox and an ABox formalism. The TBox stores terminological knowledge and background knowledge about the application domain while assertional knowledge, i.e., knowledge about the state of affairs in a particular "world", is stored in the ABox. Most implemented DL reasoners have some sort of TBox component [Horrocks 1997; Patel-Schneider 1999; Haarslev & Möller 2000a], while ABoxes are provided only by some systems [Haarslev & Möller 2000a].

2.2.1 TBoxes

Several kinds of TBoxes have been considered in the DL literature. Let us first introduce general TBoxes which are one of the most expressive TBox formalisms available.

Definition 2.4 (General TBox). An expression of the form $C \doteq D$, where C and D are concepts, is called a *concept equation*. A finite set $\mathcal{T}$ of concept equations is called *general TBox* (or *TBox* for short).

An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ iff $C^\mathcal{I} = D^\mathcal{I}$ for all $C \doteq D \in \mathcal{T}$. A concept C is *satisfiable w.r.t. a TBox* $\mathcal{T}$ iff there exists a model of C and $\mathcal{T}$. A concept D *subsumes* a concept C *w.r.t. a TBox* $\mathcal{T}$ (written $C \sqsubseteq_\mathcal{T} D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{T}$. Two concept C and D are *equivalent w.r.t. a TBox* $\mathcal{T}$ (written $C \equiv_\mathcal{T} D$) iff $C \sqsubseteq_\mathcal{T} D$ and $D \sqsubseteq_\mathcal{T} E$.

The *size* $|\mathcal{T}|$ of a TBox $\mathcal{T}$ is defined as

$$|\mathcal{T}| := \sum_{(C \doteq D) \in \mathcal{T}} |C| + |D|.$$

◇

A TBox $\mathcal{T}$ that contains only concepts from a Description Logic $\mathcal{L}$ is called $\mathcal{L}$-*TBox*. Several notions from Section 2.1.1 can be generalized from concepts to TBoxes: $\mathcal{T}$ is in negation normal form iff all concepts appearing in $\mathcal{T}$ are in NNF. By $\mathsf{sub}(\mathcal{T})$, we denote $\bigcup_{(C \doteq D) \in \mathcal{T}} \mathsf{sub}(C) \cup \mathsf{sub}(D)$, and $\mathsf{sub}(C, \mathcal{T})$ is used as an abbreviation for $\mathsf{sub}(C) \cup \mathsf{sub}(\mathcal{T})$. Also note that the two properties of $\mathsf{sub}(C)$ stated at the end of Section 2.1.1 can obviously be generalized to $\mathsf{sub}(\mathcal{T})$ and $\mathsf{sub}(C, \mathcal{T})$.

General TBoxes have frequently been considered in the literature [Calvanese *et al.* 1998a; Horrocks *et al.* 2000a] and are a powerful tool for expressing background knowledge about application domains. However, since the presence of general TBoxes significantly increases the complexity of reasoning (see below), several weaker forms of TBoxes have been considered. The most common ones are acyclic TBoxes which we introduce next.

Definition 2.5 (Acyclic TBox). An expression of the form $A \doteq C$, where A is a concept name and C a concept, is called a *concept definition*.

Let $\mathcal{T}$ be a finite set of concept definitions. A concept name A *directly uses* a concept name B in $\mathcal{T}$ if there is a concept definition $A \doteq C \in \mathcal{T}$ such that B appears in C. By "*uses*", we denote the transitive closure of "directly uses". $\mathcal{T}$ is called *acyclic TBox* if

(i) there is no concept name A such that A uses itself and

(ii) the left-hand sides of all concept definitions in $\mathcal{T}$ are pairwise distinct.

◇

Obviously, every acyclic TBox is also a general TBox. However, in contrast to general TBoxes, which are usually used to formulate general constraints, acyclic TBoxes *define* concepts, i.e., they assign concept names to complex concepts thus defining abbreviations. Figure 2.4 gives an example concept definition and an example concept equation. The concept definition defines the notion "diesel engine" by associating the

$$\begin{array}{rcl} \mathsf{Dieselengine} & \doteq & \mathsf{Engine} \sqcap \exists \mathsf{part.GlowPlug} \sqcap \forall \mathsf{fuel.Diesel} \\ \top & \doteq & (\mathsf{Engine} \sqcap \exists \mathsf{fuel.Diesel}) \rightarrow \forall \mathsf{fuel.Diesel} \end{array}$$

Figure 2.4: A concept definition and a concept equation.

define procedure unfold($C, \mathcal{T}$)
  while C contains a concept name A defined in $\mathcal{T}$ **do**
    Let $A \doteq E \in \mathcal{T}$.
    Replace each occurrence of A in C with E.
  return C

Figure 2.5: The unfolding algorithm.

concept name Dieselengine with the description of a diesel engine from Figure 2.1. The concept equation formalizes the background knowledge that all engines running with diesel fuel run with nothing else but diesel fuel.

There exist various other forms of TBoxes that are not directly relevant for this thesis. For example, one can add counting to concept equations [Baader *et al.* 1996] or drop condition (i) from acyclic TBoxes and then use least or greatest fix point semantics [Baader 1990b; Nebel 1991; Küsters 1998; Calvanese *et al.* 1999]. Another variant is obtained by replacing the equality "$\doteq$" in concept definitions by a subset relation "$\sqsubseteq$" [Calvanese 1996a; Buchheit *et al.* 1998]. As is easily seen, this latter choice does only make a difference for acyclic TBoxes since, using general TBoxes, concept equations $C \sqsubseteq D$ can be expressed by $\top \doteq C \rightarrow D$ and concept equations $C \doteq D$ can be expressed by the two equations $C \sqsubseteq D$ and $D \sqsubseteq C$. We will pick up this issue again at the end of Chapter 4.

The concept definitions in acyclic TBoxes can be viewed as macro definitions for concepts. They can even be expanded like macros and thus satisfiability of concepts w.r.t. acyclic TBoxes can be rephrased as satisfiability of concepts without reference to TBoxes. More precisely, this can be done as follows. Let $\mathcal{T}$ be an acyclic TBox. A concept C is called *unfolded* w.r.t. $\mathcal{T}$ iff none of the concept names in C occurs on the left-hand side of a concept definition in $\mathcal{T}$. Every concept C can be transformed into a concept C' such that $C \equiv_{\mathcal{T}} C'$ and C' is unfolded w.r.t. $\mathcal{T}$ by using the simple *unfolding* algorithm in Figure 2.5. It is easily seen that C' is satisfiable (without reference to TBoxes) iff C' is satisfiable w.r.t. $\mathcal{T}$ iff C is satisfiable w.r.t. $\mathcal{T}$. Moreover, the algorithm clearly terminates due to the acyclicity of acyclic TBoxes.

Nebel shows that, if a $\mathcal{TL}$-concept C is unfolded w.r.t. a $\mathcal{TL}$-TBox $\mathcal{T}$ yielding C', where $\mathcal{TL}$ is the concept language comprised solely of conjunction and universal value restriction, then the length of C' may be exponential in $|C| + |\mathcal{T}|$ [1990]. Since $\mathcal{TL}$ is a fragment of $\mathcal{ALC}$, unfolding an $\mathcal{ALC}$-concept may also lead to an exponential blowup in concept size. It follows that unfolding is an appropriate method to obtain decidability results for the satisfiability of concepts w.r.t. TBoxes but it is not an adequate means for proving complexity bounds.

As in the previous sections, we now discuss complexity issues and the connection to Modal Logic. The satisfiability of $\mathcal{ALC}$-concepts w.r.t. general TBoxes is ExpTime-complete: the upper bound is proved in [Schild 1991], and the lower bound follows from "Complexity Theorem 1" in [Schild 1994]. However, we shall reprove ExpTime-completeness below to illustrate the connection between general TBoxes and their Modal Logic counterpart. For most Description Logics, reasoning with acyclic TBoxes is of lower complexity than reasoning with general TBoxes. In Section 4.1.1, we peform a detailed analysis of the impact of acyclic TBoxes on the complexity of reasoning with DLs that contain $\mathcal{ALC}$ as a fragment. For example, we will see that satisfiability of $\mathcal{ALC}$-concepts w.r.t. acyclic TBoxes is PSpace-complete, i.e., not harder than concept satisfiability without reference to TBoxes.

While acyclic TBoxes have no counterpart in Modal Logic, the closest relative of general TBoxes is the universal modality [Goranko & Passy 1992; Hemaspaandra 1996]. Just like $\mathcal{ALC}$ with TBoxes, the Modal Logic K^u_ω—i.e, K_ω enriched with the universal modality—is ExpTime-complete [Spaan 1993a]. Spaan establishes the lower bound by adopting the ExpTime-hardness proof for PDL given by Fisher and Ladner [Fischer & Ladner 1979] and the upper bound by using a standard algorithm based on formula types [Pratt 1979]. Both techniques can straightforwardly be adapted to $\mathcal{ALC}$ with general TBoxes. However, Spaan's results imply that another way to (re)prove ExpTime-completeness of $\mathcal{ALC}$ with general TBoxes is to mutually reduce satisfiability of $\mathcal{ALC}$-concepts w.r.t. general TBoxes and satisfiability of K^u_ω-formulas to one another. We do this in the following since it gives some insight into the relationship between general TBoxes and the universal modality.

Theorem 2.6. *Satisfiability of $\mathcal{ALC}$-concepts w.r.t. general TBoxes is* ExpTime-*complete.*

Proof. Throughout this proof, we use $[R]\varphi$ and $\langle R\rangle\varphi$ to denote the K_ω box and diamond operators with modal parameter R and $[u]\varphi$ and $\langle u\rangle\varphi$ for the universal modality.[5] Modal formulas without the universal modality can be translated straightforwardly: for a K_ω-formula φ, we use $\varphi^\$$ to denote the corresponding $\mathcal{ALC}$-concept that is obtained by replacing ML operators by their DL counterparts and propositional variables by concept names. Conversely, for an $\mathcal{ALC}$-concept C, $C^\$$ denotes the corresponding K_ω-formula.

The reduction of satisfiability of $\mathcal{ALC}$-concepts w.r.t. general TBoxes to satisfiability of K^u_ω-formulas (which establishes the upper bound) is quite straightforward and was first described by Schild in [1991]: it is easily verified that a concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ iff the K^u_ω-formula

$$C^\$ \wedge \bigwedge_{(D \doteq E)\in\mathcal{T}} [u](D^\$ \leftrightarrow E^\$)$$

is satisfiable.

[5] It is out of scope to introduce K^u_ω in full detail. Readers not familiar with Modal Logic may skip the proof of Theorem 2.6 or consult [Blackburn *et al.* 2001] for more information.

Now for the lower bound, i.e, reducing satisfiability of K^u_ω-formulas to satisfiability of $\mathcal{ALC}$-concepts w.r.t. TBoxes. Let φ be a K^u_ω-formula in negation normal form (defined analogously to the NNF of concepts). We construct a corresponding concept C_φ and TBox $\mathcal{T}_\varphi$ as follows:

1. For every K^u_ω-formula ψ, we use $\nu(\psi)$ to denote the result of replacing every subformula of the form $[u]\vartheta$, where ϑ is a K_ω-formula (i.e., no occurrence of the universal modality is allowed in ϑ), with a new propositional variable p_ϑ. We inductively define a sequence of formulas $\varphi_0, \varphi_1, \dots$ by setting $\varphi_0 := \varphi$ and $\varphi_{i+1} := \nu(\varphi_i)$. It is obvious that there exists a minimal k such that $\varphi_k = \varphi_{k+1}$.
   Now fix a modal parameter U not appearing in φ_k. For every K^u_ω-formula ψ, we use $\rho(\psi)$ to denote the result of replacing every subformula of ψ of the form $\langle u\rangle\vartheta$ with $\langle U\rangle\vartheta$. Set $\varphi_{k+1} := \rho(\varphi_k)$. It is easily seen that φ_{k+1} is a K_ω-formula.
   The concept C_φ is defined as $\varphi^\$_{k+1}$. Clearly, C_φ is in NNF. In the following, we denote the concept name corresponding to a "surrogate variable" p_ψ introduced in the above translation by A_ψ.
2. The TBox $\mathcal{T}_\varphi$ is defined as

$$\bigcup_{A_\psi \text{ occurs in } C_\varphi} \Big\{\top \doteq \Big(A_\psi \rightarrow \psi^\$ \sqcap \bigsqcap_{R \text{ occurs in } C_\varphi} \big((A_\psi \rightarrow \forall R.A_\psi)\sqcap(\exists R.A_\psi \rightarrow A_\psi)\big)\Big)\Big\}$$

It is easy to check that $|C_\varphi|$ is bounded by the length of φ and that $|\mathcal{T}_\varphi|$ is at most quadratic in the length of φ. We show that C_φ is satisfiable w.r.t. $\mathcal{T}_\varphi$ iff φ is satisfiable.

First for the "if" direction. Let $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, \dots, \pi)$ be a Kripke structure which is a model of φ. We first extend $\mathcal{M}$ to a new Kripke structure $\mathcal{M}'$ that interpretes the surrogate variables p_ψ introduced in the translation process such that, for each such variable p_ψ, we have

(i) either $\mathcal{M}', w \models p_\psi$ for all $w \in W$ or $\mathcal{M}', w \models p_\psi$ for no $w \in W$;

(ii) $\mathcal{M}', w \models p_\psi$ implies $\mathcal{M}', w \models \psi$ for all $w \in W$.

This can be done by inductively defining a sequence of structures $\mathcal{M}_0, \dots, \mathcal{M}_k$:

1. Induction start. Set $\mathcal{M}_0 := \mathcal{M}$.
2. Induction step. Assume that $\mathcal{M}_{i-1}$ is already defined. Construct $\mathcal{M}_i$ from $\mathcal{M}_{i-1}$ as follows: for all $w \in W$, set

$$\pi_i(w) := \pi_{i-1}(w) \cup \{p_\psi \mid p_\psi \text{ occurs in } \varphi_i \text{ and } \mathcal{M}_{i-1}, w \models [u]\psi\}.$$

Finally, set $\mathcal{M}' := \mathcal{M}_k$. It is easy to prove by structural induction that, for $i < k$, $\psi \in \mathsf{sub}(\varphi_i)$, and $w \in W$, we have that $\mathcal{M}_i, w \models \psi$ implies $\mathcal{M}_{i+1}, w \models \nu(\psi)$, where $\mathsf{sub}(\vartheta)$ denotes the set of subformulas of ϑ. This implies that $\mathcal{M}'$ is a model of φ_k since $\mathcal{M}$ is a model of φ. Moreover, together with the construction of $\mathcal{M}'$, it implies that (ii) is satisfied. Clearly, (i) is satisfied by construction of $\mathcal{M}'$.

Now translate $\mathcal{M}'$ into an $\mathcal{ALC}$-interpretation $\mathcal{I}$ in the usual way. Let $\mathcal{J} = (W, \cdot^{\mathcal{J}})$ be the interpretation obtained from $\mathcal{I}$ by setting $U^{\mathcal{J}} := W \times W$. It is straightforward to prove by structural induction that, for $\psi \in \mathsf{sub}(\varphi_k)$ and $w \in W$, $\mathcal{M}', w \models \psi$ implies $w \in (\rho(\psi)^{\$})^{\mathcal{J}}$. Since $\mathcal{M}'$ is a model of φ_k, this implies that $\mathcal{J}$ is a model of C_φ. Considering (i) and (ii) and the construction of $\mathcal{J}$, it is not hard to see that $\mathcal{J}$ is also a model of $\mathcal{T}_\varphi$.

Now for the "only if" direction. It is well-known that $\mathcal{ALC}$ with general TBoxes has the *connected model property* [Schild 1991; Baader 1991]: if a concept C is satisfiable w.r.t. a TBox $\mathcal{T}$, then there exists a model $\mathcal{I}$ of $\mathcal{T}$ and a $d \in \Delta_{\mathcal{I}}$ such that $d \in C^{\mathcal{I}}$ and, for every $d' \in \Delta_{\mathcal{I}}$ with $d \neq d'$, there exists a sequence of domain elements $e_0, \ldots, e_k \in \Delta_{\mathcal{I}}$ and a sequence of roles $R_0, \ldots, R_{k-1}$ occurring in C or $\mathcal{T}$ such that $d = e_0$, $d' = e_k$, and $(e_i, e_{i+1}) \in R_i$ for $i < k$. Let $\mathcal{I}$ be such a connected model of C_φ and $\mathcal{T}_\varphi$. Using connectedness and the definition of $\mathcal{T}_\varphi$, it is easy to show that, for all A_ψ occurring in C_φ, we have

(i) either $A_\psi^{\mathcal{I}} = \Delta_{\mathcal{I}}$ or $A_\psi^{\mathcal{I}} = \emptyset$ and

(ii) if $A_\psi^{\mathcal{I}} = \Delta_{\mathcal{I}}$, then $(\psi^{\$})^{\mathcal{I}} = \Delta_{\mathcal{I}}$.

Translate $\mathcal{I}$ into a Kripke structure $\mathcal{M} = (\Delta_{\mathcal{I}}, \mathcal{R}_1, \mathcal{R}_2, \ldots, \pi)$ in the usual way. Using structural induction, the fact that $\phi_0, \ldots, \phi_k$ are in NNF, and points (i) and (ii) from above, it is straightforward to show that

1. For $\psi \in \mathsf{sub}(\varphi_k)$ and $w \in \Delta_{\mathcal{I}}$, $w \in (\rho(\psi)^{\$})^{\mathcal{I}}$ implies $\mathcal{M}, w \models \psi$.
2. For $i < k$, $\psi \in \mathsf{sub}(\varphi_i)$, and $w \in W$, $\mathcal{M}, w \models \nu(\psi)$ implies $\mathcal{M}, w \models \psi$.

Since $\mathcal{I}$ is a model of C_φ, it follows that $\mathcal{M}$ is a model of φ. ❑

2.2.2 ABoxes

In contrast to TBoxes, there usually exists only a single "natural" ABox formalism for a given concept language. However, there may be small variations in the ABox formalisms of distinct concept languages. The ABox formalism for $\mathcal{ALC}$ is defined as follows.

Definition 2.7 (ABox). Let $\mathsf{O_a}$ be a countably infinite set of *abstract objects*. If C is a concept, R a role name, and $a, b \in \mathsf{O_a}$, then $a : C$ and $(a, b) : R$ are *$\mathcal{ALC}$-assertions*. A finite set of assertions is called an *$\mathcal{ALC}$-ABox*. Interpretations $\mathcal{I}$ can be extended to ABoxes by demanding that $\cdot^{\mathcal{I}}$ maps every abstract object a to an element $a^{\mathcal{I}}$ of $\Delta_{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* an assertion

$$\begin{array}{rcl} a : C & \text{iff} & a^{\mathcal{I}} \in C^{\mathcal{I}} \\ (a, b) : R & \text{iff} & (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}. \end{array}$$

An interpretation is a *model* of an ABox $\mathcal{A}$ iff it satisfies all assertions in $\mathcal{A}$. An ABox $\mathcal{A}$ is *consistent (ABox)* w.r.t. a TBox $\mathcal{T}$ iff there exists a model of $\mathcal{A}$ and $\mathcal{T}$. ◇

$\mathsf{c1} : \mathsf{car}$	$\mathsf{e1} : \mathsf{Engine} \sqcap \exists \mathsf{fuel}.\mathsf{Diesel}$
$(\mathsf{c1}, \mathsf{e1}) : \mathsf{part}$	
$\mathsf{e2} : \mathsf{Dieselengine}$	$\mathsf{c2} : \mathsf{car}$
$\mathsf{f2} : \mathsf{Kerosine}$	$\mathsf{f2} : \neg\mathsf{Diesel}$
$(\mathsf{c2}, \mathsf{e2}) : \mathsf{part}$	$(\mathsf{e2}, \mathsf{f2}) : \mathsf{fuel}$

Figure 2.6: An example ABox.

For an ABox $\mathcal{A}$, we write $\mathsf{sub}(\mathcal{A})$ to denote the set

$$\{C \mid C \in \mathsf{sub}(D) \text{ and } a : D \in \mathcal{A} \text{ for some } a \in \mathsf{O_a}\}.$$

In the sequel, we will also consider ABox consistency without reference to TBoxes, i.e., with reference to the empty TBox. Throughout this thesis, abstract objects are denoted by a and b. Note that we do not require $a^\mathcal{I} \neq b^\mathcal{I}$ for all $a, b \in \mathsf{O_a}$ with $a \neq b$, i.e., the *unique name assumption* is not imposed. Moreover, we make the *open world assumption*, which means that there may exist domain elements $d \in \Delta_\mathcal{I}$ such that $a^\mathcal{I} \neq d$ for all $a \in \mathsf{O_a}$. In the literature, there exist additional reasoning problems on ABoxes [Baader & Hollunder 1991b] such as instance checking: an abstract object a is an *instance* of a concept C in an ABox $\mathcal{A}$ iff $a^\mathcal{I} \in C^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{A}$. However, most such reasoning problems can be reduced to (in)consistency. For example, an object a is an instance of a concept C in an ABox $\mathcal{A}$ iff $\mathcal{A} \cup \{a : \neg C\}$ is inconsistent. In what follows, we will restrict ourselves to the ABox consistency problem.

It is not hard to see that concept satisfiability can also be reduced to ABox consistency: the concept C is satisfiable iff the ABox $\{a : C\}$ is consistent. For some DLs, the converse also holds: as shown by Hollunder in [1996], it is possible to reduce $\mathcal{ALC}$-ABox consistency to $\mathcal{ALC}$-concept satisfiability. However, the reduction is more involved and we will come back to it in Section 4.1.3.

A self-explanatory example ABox can be found in Figure 2.6. Note that the displayed ABox is inconsistent w.r.t. the TBox in Figure 2.4.

For most Description Logics, the complexity of ABox consistency coincides with the complexity of concept satisfiability.[6] This is also the case for $\mathcal{ALC}$, i.e., consistency of $\mathcal{ALC}$-ABoxes w.r.t. general TBoxes is ExpTime-complete [Giacomo & Lenzerini 1994] while consistency of $\mathcal{ALC}$-ABoxes without reference to TBoxes is PSpace-complete [Baader & Hollunder 1991b]. In Section 4.1.1, we will see that consistency of $\mathcal{ALC}$-ABoxes w.r.t. acyclic TBoxes is also PSpace-complete. There does not really exist a counterpart of ABoxes in Modal Logic. The closest relation is to logics that allow the use of *nominals* (the modal equivalent of abstract objects) in formulas. However, nominals provide strictly more expressive power than ABoxes. An important family of logics providing for nominals are the so-called *hybrid logics* [Areces *et al.* 1999]. The translation of ABoxes into hybrid logics is discussed in [Areces & de Rijke 2001].

[6] But see [Schaerf 1993] for a counterexample: in the Description Logic $\mathcal{ALE}$, concept satisfiability is co-NP-complete while ABox consistency is PSpace-complete.

2.3 Description Logics with Concrete Domains

Although the DL-based approach to knowledge representation is quite successful in general, it has been identified as a serious shortcoming of many Description Logics that *all* knowledge has to be described on an abstract logical level. More precisely, most DLs do not allow to adequately express knowledge about "concrete qualities" of entities from the application domain, such as their length, duration, or temperature. In order to facilitate the integration of such knowledge, Baader and Hanschke proposed the extension of $\mathcal{ALC}$ with concrete domains and predicates on these domains [1991a], where concrete domains are, e.g., the integers, the reals, sets of temporal intervals, or sets of spatial regions, and predicates include equality, temporal overlapping, and spatial disconnectedness. The interface between the Description Logic and the concrete domain is provided by a concrete domain concept constructor. Important application areas which have been found to depend on integrated reasoning with concrete domains are, e.g., mechanical engineering [Baader & Hanschke 1992], reasoning about aggregation in databases [Baader & Sattler 1998], reasoning with physical laws [Kamp & Wache 1996] as well as temporal and spatial reasoning [Haarslev *et al.* 1999].

2.3.1 Introducing $\mathcal{ALC}(\mathcal{D})$

To illustrate how knowledge representation with Description Logics can benefit from the use of concrete domains, let us view some examples before introducing concrete domains formally. Using Baader and Hanschke's concrete domain constructor, we can refine the diesel engine from Figure 2.4 by writing

$$\begin{array}{l}\mathsf{Dieselengine} \sqcap \exists \mathsf{carburettor}.\mathsf{Carburettor} \\ \sqcap\ \exists \mathsf{power}.=_{45} \sqcap \exists \mathsf{weight}, (\mathsf{carburettor\ weight}).> .\end{array}$$

Intuitively, this concept describes a diesel engine with a power of 45PS whose weight is greater than the weight of its carburettor. Here, the third and fourth conjunct are applications of the concrete domain constructor. More precisely, $\mathsf{carburettor}$ is an abstract feature as introduced in Section 2.1.2 while power and weight are so-called *concrete features*, which are interpreted as partial functions from the domain $\Delta_{\mathcal{I}}$ into the concrete domain (say, the reals). The expression "($\mathsf{carburettor\ weight}$)" is a sequence of features written in brackets for better readability and "$>$" is a predicate from the concrete domain.

When talking about processes, it seems appropriate to use a concrete domain comprised of time intervals and corresponding temporal predicates. The following concept describes a process with two subprocesses that are both running within the same time interval, which is contained in the time interval of the "mother" process.

$$\begin{array}{l}\mathsf{Process} \sqcap \exists \mathsf{subproc1}.\mathsf{Process} \sqcap \exists \mathsf{subproc2}.\mathsf{Process} \\ \sqcap\ \exists \mathsf{time}, (\mathsf{subproc1\ time}).\mathsf{contained} \sqcap \exists (\mathsf{subproc1\ time}), (\mathsf{subproc2\ time}).=\end{array}$$

Obviously, concrete domains must be taken into account when checking the satisfiability of concepts. If, for example, we take the conjunction of the above concept with the concept $\exists \mathsf{time}, (\mathsf{subproc2\ time}).\mathsf{disjoint}$ stating that the running time of the second subprocess is disjoint from the running time of the mother process, then we obtain a concept that is inconsistent due to the interplay of concrete domain predicates. Let us now make things more precise.

Definition 2.8. A *concrete domain* $\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set and $\Phi_\mathcal{D}$ a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n and an n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$. Let V be a set of variables. A predicate conjunction of the form

$$c = \bigwedge_{i<k} (x_0^{(i)}, \ldots, x_{n_i}^{(i)}) : P_i,$$

where P_i is an n_i-ary predicate for $i < k$ and the $x_j^{(i)}$ are variables from V, is called *satisfiable* iff there exists a function δ mapping the variables in c to elements of $\Delta_\mathcal{D}$ such that $(\delta(x_0^{(i)}), \ldots, \delta(x_{n_i}^{(i)})) \in P_i^\mathcal{D}$ for $i \leq k$. Such a function is called a *solution* for c. A concrete domain $\mathcal{D}$ is called *admissible* iff

1. its set of predicate names is closed under negation and contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$ and
2. the satisfiability problem for finite conjunctions of predicates is decidable.

By $\overline{P}$, we denote the name for the negation of the predicate P, i.e., if the arity of P is n, then $\overline{P}^\mathcal{D} = \Delta_\mathcal{D}^n \setminus P^\mathcal{D}$. ◇

In what follows, we refer to the satisfiability of finite conjunctions of predicates from a concrete domain $\mathcal{D}$ as *$\mathcal{D}$-satisfiability.* The basic Description Logic equipped with concrete domains is $\mathcal{ALC}(\mathcal{D})$—the extension of $\mathcal{ALC}$ with a concrete domain $\mathcal{D}$ [Baader & Hanschke 1991a]. It is important to note that the concrete domain $\mathcal{D}$ is not fixed but rather a parameter to the DL. In the following, we introduce $\mathcal{ALC}(\mathcal{D})$ in detail since all logics with concrete domains considered in this thesis are extensions of this logic.

Definition 2.9 ($\mathcal{ALC}(\mathcal{D})$ Syntax). Let $\mathsf{N_{aF}}$ be a countably infinite subset of $\mathsf{N_R}$ such that $\mathsf{N_R} \setminus \mathsf{N_{aF}}$ is also countably infinite. Elements of $\mathsf{N_{aF}}$ are called *abstract features.* Let $\mathsf{N_{cF}}$ be a countable infinite set of *concrete features* such that $\mathsf{N_R} \cap \mathsf{N_{cF}} = \emptyset$ and $\mathsf{N_C} \cap \mathsf{N_{cF}} = \emptyset$. A *concrete path* is a sequence $f_1 \cdots f_k g$, where $f_1, \ldots, f_k \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$.

Let $\mathcal{D}$ be a concrete domain. $\mathcal{ALC}(\mathcal{D})$ is obtained from $\mathcal{ALC}$ by allowing the use of the concrete domain constructor $\exists u_1, \ldots, u_n.P$ and the undefinedness constructor $g\uparrow$ in place of concept names, where $g \in \mathsf{N_{cF}}$, $u_1, \ldots, u_n$ are concrete paths, and $P \in \Phi_\mathcal{D}$ is a predicate with arity n.

Let $\mathsf{O_c}$ be a countably infinite set of *concrete objects* disjoint from the set of abstract objects $\mathsf{O_a}$. Let C be an $\mathcal{ALC}(\mathcal{D})$-concept, $R \in \mathsf{N_R}$ a role (possibly an abstract feature), g a concrete feature, $a \in \mathsf{O_a}$, $x_1, \ldots, x_n \in \mathsf{O_c}$, and $P \in \Phi_\mathcal{D}$ with arity n. Then

$$a : C, \quad (a, b) : R, \quad (a, x) : g, \quad \text{and} \quad (x_1, \ldots, x_n) : P$$

are *$\mathcal{ALC}(\mathcal{D})$-assertions.* An *$\mathcal{ALC}(\mathcal{D})$-ABox* is a finite set of $\mathcal{ALC}(\mathcal{D})$-assertions. ◇

We use $u\uparrow$ as an abbreviation for $\forall f_1.\cdots.\forall f_k.g\uparrow$ if $u = f_1 \cdots f_k g$ is a concrete path. In the sequel, we denote concrete features by g, concrete paths by u, and concrete objects by x and y. The presented syntax of the concrete domain constructor is different from Baader and Hanschke's who write $P(u_1, \ldots, u_n)$ instead of $\exists u_1, \ldots, u_n.P$. The semantics of the additional constructors illustrate how concrete domains are integrated into the Description Logic.

Definition 2.10 ($\mathcal{ALC}(\mathcal{D})$ Semantics). An $\mathcal{ALC}(\mathcal{D})$-*interpretation* $\mathcal{I}$ is a pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta_\mathcal{I}$ is a non-empty set called the *abstract domain* and $\cdot^\mathcal{I}$ is an *interpretation function* that maps

- every concept name A to a subset $A^\mathcal{I}$ of $\Delta_\mathcal{I}$,
- every role name R to a binary relation $R^\mathcal{I}$ over $\Delta_\mathcal{I}$,
- every abstract feature f to a partial function $f^\mathcal{I}$ from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{I}$, and
- every concrete feature g to a partial function $g^\mathcal{I}$ from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{D}$.

If $u = f_1 \cdots f_k g$ is a concrete path, then $u^\mathcal{I}$ is defined as the composition of the path's components: $g^\mathcal{I}(f_k^\mathcal{I}(\cdots(f_1^\mathcal{I}(\cdot))))$. The semantics of the additional concept constructors is as follows:

$$\begin{aligned}
(\exists u_1, \ldots, u_n.P)^\mathcal{I} \;&:=\; \{d \in \Delta_\mathcal{I} \mid \text{There exist } x_1, \ldots, x_n \text{ such that} \\
&\qquad u_i^\mathcal{I}(d) = x_i \text{ for } 1 \le i \le n \text{ and } (x_1, \ldots, x_n) \in P^\mathcal{D}\} \\
(g\uparrow)^\mathcal{I} \;&:=\; \{d \in \Delta_\mathcal{I} \mid g^\mathcal{I}(d) \text{ undefined}\}
\end{aligned}$$

Interpretations $\mathcal{I}$ are straightforwardly extended to ABoxes, i.e., $\mathcal{I}$ maps each $a \in \mathsf{O_a}$ to an element $a^\mathcal{I}$ of $\Delta_\mathcal{I}$ and each $x \in \mathsf{O_c}$ to an element $x^\mathcal{I}$ of $\Delta_\mathcal{D}$. Since the interpretation of assertions $a : C$ and $(a, b) : R$ is already given in Definition 2.7, we only need to deal with the additional kinds of assertions: $\mathcal{I}$ *satisfies*

$$\begin{aligned}
(a, x) : g \quad &\text{iff} \quad g^\mathcal{I}(a^\mathcal{I}) = x^\mathcal{I} \\
(x_1, \ldots, x_n) : P \quad &\text{iff} \quad P^\mathcal{D}(x_1^\mathcal{I}, \ldots, x_n^\mathcal{I}).
\end{aligned}$$

◇

According to our definition, $\mathcal{ALC}(\mathcal{D})$ provides for two different types of features: abstract and concrete ones. In contrast, Baader and Hanschke's original version of $\mathcal{ALC}(\mathcal{D})$ offers only a single type of feature that is interpreted as a partial function from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{I} \cup \Delta_\mathcal{D}$. Our logic is slightly less expressive than Baader and Hanschke's since, in our version, the "range" of features is fixed and expressions such as $\exists f.C \sqcup \exists f.P$ are not well-formed concepts. However, in knowledge representation it seems rather hard to find any cases in which the additional expressiveness is really needed. Furthermore, concepts like $\exists f.C \sqcup \exists f.P$ from Baader and Hanschke's logic can be "simulated" by writing

$$(\exists f.C \sqcap g\uparrow) \sqcup (\forall f.\bot \sqcap \exists g.P).$$

We keep concrete and abstract features separated since this allows a clearer algorithmic treatment and clearer proofs. Another difference to Baader and Hanschke's logic is the presence of the undefinedness constructor $g\uparrow$. Since we assume abstract and concrete features to be separated, this constructor is necessary to ensure that every $\mathcal{ALC}(\mathcal{D})$-concept can be converted into an equivalent one in negation normal form (see Chapter 3).

For examples of $\mathcal{ALC}(\mathcal{D})$-concepts, we refer to the beginning of this section: all examples given there are in fact $\mathcal{ALC}(\mathcal{D})$-concepts. Concrete domains have no direct counterpart in Modal Logic or other areas of mathematical logic but appear rather frequently as a part of Description Logics. Although Baader and Hanschke presented the first rigorous treatment of concrete domains in 1991, many early DL reasoners like MESON [Edelmann & Owsnicki 1986] or CLASSIC [Brachman *et al.* 1991] provided for more or less elaborate means to deal with "concrete domains" even before 1991. Baader and Hanschke's approach was implemented in the TAXON system [Abecker *et al.* 1991]. More recent Description Logics with concrete domains include the following:

- In [Baader & Sattler 1998], an extension of $\mathcal{ALC}(\mathcal{D})$ with aggregation functions is defined. More precisely, Baader and Sattler allow concrete domains to provide for aggregation functions like max and sum which can then be used to construct complex concrete features. The main motivation for integrating aggregation functions is to allow reasoning about database schemas.

- Kamp and Wache describe a Description Logic called CTL which is mainly a combination of Description Logics and constraint logic programming (CLP) [Kamp & Wache 1996]. The combination is realized using concept constructors rather similar to the concrete domain concept constructor. The authors motivate their logic by showing that it is a powerful tool for reasoning about technical devices.

- Very recent proposals combine general TBoxes with extremely restricted forms of concrete domains. In [Horrocks & Sattler 2001], only unary concrete domain predicates are admitted in an otherwise very expressive logic that is motivated as a tool for reasoning about ontologies. In [Haarslev *et al.* 2001], a slightly more general approach is pursued: predicates of arbitrary arity are admitted but only concrete features instead of paths may be used inside the concrete domain constructor. As in the previously mentioned approach, the restriction serves the purpose of allowing an easy combination of concrete domains with a very expressive Description Logic. We will return to both approaches in Section 2.4.1.

Two additional extensions of $\mathcal{ALC}(\mathcal{D})$ will be discussed in Section 2.3.2.

The complexity of Description Logics with concrete domains has, until now, not been investigated in any detail. Baader and Hanschke only show that satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts and consistency of $\mathcal{ALC}(\mathcal{D})$-ABoxes is decidable if the concrete domain $\mathcal{D}$ is admissible [1991a]. Admissibility of concrete domains is usually required for devising general decision procedures, i.e., decision procedures that work for *any* (admissible) concrete domain and not just for a fixed one. It is obvious that the complexity of $\mathcal{D}$-satisfiability is closely related to the complexity of $\mathcal{ALC}(\mathcal{D})$-concept satisfiability: if $\mathcal{D}$-satisfiability is hard for some complexity class C, then $\mathcal{ALC}(\mathcal{D})$-concept satisfiability is also hard for C. However, the converse obviously does not hold in general. In Chapter 3, we prove that satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts and consistency of $\mathcal{ALC}(\mathcal{D})$-ABoxes (both without reference to TBoxes) is PSPACE-complete if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in PSPACE. In subsequent sections, we add TBoxes and more concept and role constructors and investigate the complexity of the resulting logics. Let us jump ahead by saying that the addition of acyclic TBoxes significantly increases complexity yet retaining decidability while, for many concrete domains $\mathcal{D}$, the addition of general TBoxes leads to undecidability.

There is an obvious syntactic similarity between the concrete domain constructor of $\mathcal{ALC}(\mathcal{D})$ and the feature (dis)agreements of $\mathcal{ALCF}$ since both are defined in terms of sequences of features. Indeed, as mentioned by Baader and Hanschke [1991a], the introduction of the concrete domain constructor was motivated by feature (dis)agreements. Moreover, we shall see later that both constructors can be treated with similar algorithmic techniques and behave largely identically w.r.t. complexity and decidability if combined with TBoxes and other role or concept constructors. Throughout this thesis, the investigation of (extensions of) $\mathcal{ALCF}$ will thus be an important side-track.

2.3.2 Extensions of $\mathcal{ALC}(\mathcal{D})$

Although concrete domains have, until now, only rarely been combined with the standard Description Logic concept and role constructors mentioned in Section 2.1.2, several extensions of and additions to the concrete domain part itself have been proposed. Some of these extensions, such as the integration of aggregation functions, have already been described in the previous section. In this section, we introduce two extensions of $\mathcal{ALC}(\mathcal{D})$ in more detail since they are relevant for later chapters.

Generalized Concrete Domain Constructors

Hanschke [1992] proposed to extend $\mathcal{ALC}(\mathcal{D})$ in various directions: he adds (i) a more general form of concrete domain constructor, (ii) so-called abstract predicates, and (iii) feature (dis)agreements. We shall only introduce the generalized concrete domain constructors here since abstract predicates are irrelevant for our purposes and feature (dis)agreements have already been discussed in Section 2.1.2. The generalized constructors allow the use of roles where the standard concrete domain constructor allows only features. This extension makes it rather natural to introduce a universal version of the concrete domain constructor as well.

Definition 2.11 (Syntax and semantics of $\mathcal{ALCP}(\mathcal{D})$). A sequence $U = R_1 \dots R_k g$ where $R_1, \dots, R_k \in \mathsf{N_R}$ and $g \in \mathsf{N_{cF}}$ is called a *role path*. For an interpretation $\mathcal{I}$, $U^\mathcal{I}$ is defined as

$$\{(d,x) \subseteq \Delta_\mathcal{I} \times \Delta_\mathcal{D} \mid \exists d_1, \dots, d_{k+1} : d = d_1, \\ (d_i, d_{i+1}) \in R_i^\mathcal{I} \text{ for } 1 \leq i \leq k, \text{ and } g^\mathcal{I}(d_{k+1}) = x\}.$$

$\mathcal{ALCP}(\mathcal{D})$ is obtained from $\mathcal{ALC}(\mathcal{D})$ by allowing the use of concepts $\forall U_1, \ldots, U_n.P$ and $\exists U_1, \ldots, U_n.P$ in place of concept names, where $P \in \Phi_\mathcal{D}$ is of arity n and $U_1, \ldots, U_n$ are role paths.

The semantics of the generalized concrete domain constructors is defined as follows:

$$(\forall U_1, \ldots, U_n.P)^\mathcal{I} := \{d \in \Delta_\mathcal{I} \mid \text{For all } x_1, \ldots, x_n \text{ with } (d, x_i) \in U_i^\mathcal{I} \text{ for } 1 \leq i \leq n, \text{ we have } (x_1, \ldots, x_n) \in P^\mathcal{D}\}$$

$$(\exists U_1, \ldots, U_n.P)^\mathcal{I} := \{d \in \Delta_\mathcal{I} \mid \text{There exist } x_1, \ldots, x_n \text{ with } (d, x_i) \in U_i^\mathcal{I} \text{ for } 1 \leq i \leq n \text{ and } (x_1, \ldots, x_n) \in P^\mathcal{D}\}$$

◇

In the following, we denote role paths by U. Obviously, every concrete path is also a role path. Hence, the $\exists U_1, \ldots, U_n.P$ constructor of $\mathcal{ALCP}(\mathcal{D})$ is a generalization of the $\exists u_1, \ldots, u_n.P$ constructor of $\mathcal{ALC}(\mathcal{D})$. For concrete paths $u_1, \ldots, u_n$, the $\mathcal{ALCP}(\mathcal{D})$-concept $\forall u_1, \ldots, u_n.P$ is equivalent to the $\mathcal{ALC}(\mathcal{D})$-concept $u_1 \uparrow \sqcup \cdots \sqcup u_n.\uparrow \sqcup \exists u_1, \ldots, u_n.P$. This is the reason why $\mathcal{ALC}(\mathcal{D})$ does not need a counterpart of the $\forall U_1, \ldots, U_n.P$ constructor.

Satisfiability of $\mathcal{ALCP}(\mathcal{D})$-concepts and consistency of $\mathcal{ALCP}(\mathcal{D})$-ABoxes are decidable if $\mathcal{D}$ is admissible [Hanschke 1992]. The generalized constructors are not just syntactic sugar but truly increase the expressivity of the logic. For example, if we do not want to fix the number of cylinders a diesel engine has, we have to relate the engine to the cylinders using a role:[7]

$$\mathsf{Dieselengine} \sqcap \forall \mathsf{cylinder}.\mathsf{Cylinder}$$

In $\mathcal{ALCP}(\mathcal{D})$, we can now state that all cylinders of the engine have the same maximum operating temperature by writing

$$\forall (\mathsf{cylinder\ maxOpTemp}), (\mathsf{cylinder\ maxOpTemp}). = .$$

It is not hard to see that this construction is not possible without the generalized constructors (nor can it easily be "simulated"). We investigate the complexity of $\mathcal{ALCP}(\mathcal{D})$ in Chapter 5.

A Concrete Domain Role Constructor

The Description Logic $\mathcal{ALC}^{rp}(\mathcal{D})$ introduced in [Haarslev *et al.* 1999] extends $\mathcal{ALC}(\mathcal{D})$ with a constructor that allows the definition of roles with reference to the concrete domain.

Definition 2.12 ($\mathcal{ALC}^{rp}(\mathcal{D})$). A *predicate role* is an expression of the form

$$\exists (u_1, \ldots, u_n), (v_1, \ldots, v_m).P$$

where $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ are concrete paths and P is an $n+m$-ary predicate. The semantics is given as follows:

$$(\exists (u_1, \ldots, u_n), (v_1, \ldots, v_m).P)^\mathcal{I} := \{(d, e) \in \Delta_\mathcal{I} \times \Delta_\mathcal{I} \mid \text{There exist } x_1, \ldots, x_n \text{ and } y_1, \ldots, y_n \text{ such that } u_i^\mathcal{I}(d) = x_i \text{ for } 1 \leq i \leq n, v_i^\mathcal{I}(d) = y_i \text{ for } 1 \leq i \leq m, \text{ and } (x_1, \ldots, x_n, y_1, \ldots, y_m) \in P^\mathcal{D}\}$$

[7]If we had a fixed number k of cylinders, we could use abstract features $\mathsf{cylinder}_1, \ldots, \mathsf{cylinder}_k$.

$\mathcal{ALC}^{rp}(\mathcal{D})$ is obtained from $\mathcal{ALC}(\mathcal{D})$ by allowing the use of predicate roles in place of role names from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$. $\diamond$

In [Lutz & Möller 1997], it is proved that there exists a large class of concrete domains $\mathcal{D}$ such that satisfiability of $\mathcal{ALC}^{rp}(\mathcal{D})$-concepts is undecidable. However, in [Haarslev *et al.* 1999] a fragment of $\mathcal{ALC}^{rp}(\mathcal{D})$ is identified that is closed under negation, strictly extends $\mathcal{ALC}(\mathcal{D})$, and is decidable for all admissible concrete domains. In the following, we introduce this fragment. To do so, we need a way to convert $\mathcal{ALC}^{rp}(\mathcal{D})$-concepts into equivalent ones in NNF. This conversion is done by exhaustively applying the rewrite rules from Definition 2.3.1 together with the following ones:

$$\begin{array}{rcl} \neg(\exists u_1,\ldots,u_n.P) & \rightsquigarrow & \exists u_1,\ldots,u_n.\overline{P} \sqcup u_1\uparrow \sqcup \cdots \sqcup u_n\uparrow \\ \neg(g\uparrow) & \rightsquigarrow & \exists g.\top_{\mathcal{D}} \end{array}$$

Definition 2.13 (Restricted $\mathcal{ALC}^{rp}(\mathcal{D})$-concept). An $\mathcal{ALC}^{rp}(\mathcal{D})$-concept C is called *restricted* iff the result C' of converting C to NNF satisfies the following conditions:

1. For any $\forall R.D \in \mathsf{sub}(C')$, where R is a predicate role,
   (a) $\mathsf{sub}(D)$ does not contain any concepts $\exists S.E$ with S a predicate role, and
   (b) if $\mathsf{sub}(D)$ contains a concept $\exists u_1,\ldots,u_n.P$, then $u_1,\ldots,u_n \in \mathsf{N_{cF}}$.
2. For any $\exists R.D \in \mathsf{sub}(C')$, where R is a predicate role,
   (a) $\mathsf{sub}(D)$ does not contain any concepts $\forall S.E$ with S a predicate role, and
   (b) if $\mathsf{sub}(D)$ contains a concept $\exists u_1,\ldots,u_n.P$, then $u_1,\ldots,u_n \in \mathsf{N_{cF}}$.

$\diamond$

It is easily seen that the set of restricted $\mathcal{ALC}^{rp}(\mathcal{D})$-concepts is closed under negation. Hence, subsumption of restricted $\mathcal{ALC}^{rp}(\mathcal{D})$-concepts can still be reduced to satisfiability of restricted $\mathcal{ALC}^{rp}(\mathcal{D})$-concepts (and vice versa). Since all $\mathcal{ALC}^{rp}(\mathcal{D})$-concepts we use in this thesis (also inside TBoxes and ABoxes) are restricted, in what follows we simply write "$\mathcal{ALC}^{rp}(\mathcal{D})$-concept" for "restricted $\mathcal{ALC}^{rp}(\mathcal{D})$-concept".

Let us demonstrate the expressive power of predicate roles by using a temporal concrete domain to model a process. The concept

$$\mathsf{Process} \sqcap \forall(\exists(\mathsf{time}),(\mathsf{time}).\mathsf{overlaps}).\neg\mathsf{Dangerous\text{-}Process}$$

describes processes that do not temporally overlap any dangerous processes. Here, $\exists(\mathsf{time}),(\mathsf{time}).\mathsf{overlaps}$ is a predicate role. Unrestricted $\mathcal{ALC}^{rp}(\mathcal{D})$ with a temporal concrete domain based on time intervals is very closely related to the (undecidable) Modal Logic of time intervals presented by Halpern and Shoham [1991]. This logic is discussed in more detail in Section 6.3. The complexity of reasoning with (restricted) $\mathcal{ALC}^{rp}(\mathcal{D})$ is investigated in Chapter 5.

2.4 Examples of Concrete Domains

All example $\mathcal{ALC}(\mathcal{D})$-concepts that we presented so far depended on the intuition of the reader in that we used predicates from several concrete domains without ever introducing them formally. In this section, we present a selection of useful concrete domains ranging from rather inexpressive to very expressive. The description of each concrete domain $\mathcal{D}$ is accompanied by some examples and a brief discussion of the complexity of $\mathcal{D}$-satisfiability.

2.4.1 Unary Concrete Domains and $\mathcal{ALCf}(\mathcal{D})$

We call a concrete domain $\mathcal{D}$ *unary* if all predicates $P \in \Phi_\mathcal{D}$ are unary. Unary concrete domains offer only very limited expressive power but have the advantage that existing DL reasoning algorithms can rather easily be extended to take such concrete domains into account. Despite their limited expressivity, unary concrete domains are of interest in several application areas such as reasoning about ontologies [Horrocks & Sattler 2001].

Let us introduce a typical unary concrete domain. The concrete domain E is defined by setting

$$\begin{array}{lll} \Delta_\mathsf{E} & := & \mathbb{N} \text{ (i.e., the naturals)} \\ \Phi_\mathsf{E} & := & \{\top_\mathsf{E}, \bot_\mathsf{E}\} \cup \{P_r \mid P \in \{=, \neq, <, >, \leq, \geq\} \text{ and } r \in \Delta_\mathsf{E}\} \end{array}$$

where all predicates are unary and have the obvious extension. For example,

$$(\geq_4)^\mathcal{D} = \{r \in \mathbb{Q} \mid r \geq 4\}.$$

It is readily checked that E is admissible: the extension of $\top_\mathsf{E}$ is Δ_E, Φ_E is clearly closed under negation, and it is straightforward to devise a polynomial time algorithm for deciding E-satisfiability. What kind of expressivity is offered by the Description Logic $\mathcal{ALC}(\mathsf{E})$? Since all predicates are unary, E allows to describe concrete values attached to elements of the abstract domain via concrete features but it does not allow to describe the *relationship* between such concrete values. For example, the $\mathcal{ALC}(\mathsf{E})$-concept

$$\mathsf{Engine} \sqcap (\exists \mathsf{weight}. \geq_{200} \sqcap \exists \mathsf{power}. \geq_{50})$$

describes an engine that weights at least 200kg and whose power is at least 50PS (we assume that units are fixed). However, using $\mathcal{ALC}(\mathsf{E})$, it is impossible to relate, say, the weight of the engine to the weight of its carburettor.

In [Haarslev *et al.* 2001], a Description Logic with concrete domains is presented that admits only concrete features inside the concrete domain constructor instead of concrete paths. This restriction serves the purpose of allowing an easy extension of reasoning algorithms for expressive Description Logics to take into account concrete domains. It is not hard to see that there exists a close connection between unary concrete domains and the described restriction of the concrete domain constructor. More precisely, in the case of unary concrete domains, the mentioned restriction can be made without loss of generality since $\exists u.P$ can be replaced with $\exists f_1.\cdots.\exists f_k.\exists g.P$ for $u = f_1 \ldots f_k g$.

As already noted, the use of unary concrete domains does not significantly enhance the expressive power of $\mathcal{ALC}$, and the same holds for the restricted concrete domain constructor. We illustrate this by (re)proving decidability of $\mathcal{ALC}f(\mathcal{D})$-concept satisfiability w.r.t. general TBoxes, where $\mathcal{ALC}f(\mathcal{D})$ is obtained from $\mathcal{ALC}(\mathcal{D})$ by restricting the concrete domain constructor to concrete features in place of concrete paths. The weakness of the restricted concrete domain constructor is illustrated by the proof itself rather than by the result: it shows that, intuitively, the restricted concrete domain constructor can be "simulated" by concept names.

For the proof, it is convenient to assume the input concept and TBox to be in negation normal form. Every $\mathcal{ALC}f(\mathcal{D})$-concept C and TBox $\mathcal{T}$ can be transformed into an equivalent one in NNF using the rewrite rules from Definition 2.3.1 together with the following ones:

$$\begin{array}{rcl}\neg(\exists g_1,\ldots,g_n.P) & \rightsquigarrow & \exists g_1,\ldots,g_n.\overline{P} \sqcup g_1\uparrow \sqcup \cdots \sqcup g_n\uparrow \\ \neg(g\uparrow) & \rightsquigarrow & \exists g.\top_{\mathcal{D}}\end{array}$$

We can now prove the decidability result.

Theorem 2.14. *If $\mathcal{D}$ is an admissible concrete domain, then satisfiability of $\mathcal{ALC}f(\mathcal{D})$-concepts w.r.t. general TBoxes is decidable.*

Proof. We prove the theorem by reducing satisfiability of $\mathcal{ALC}f(\mathcal{D})$-concepts w.r.t. TBoxes to satisfiability of $\mathcal{ALC}$-concepts w.r.t. TBoxes. Let C be an $\mathcal{ALC}f(\mathcal{D})$-concept and $\mathcal{T}$ be a TBox whose satisfiability is to be decided. W.l.o.g., we assume that C and $\mathcal{T}$ are in NNF and that all concept equations in $\mathcal{T}$ are of the form $D \doteq \top$. The latter can be done since any concept equation $D \doteq E$ can be rewritten as $(D \sqcap E) \sqcup (\neg D \sqcap \neg E) \doteq \top$.

Define the set of "relevant" concrete domain concepts as

$$\Gamma := \{\exists g_1,\ldots,g_k.P \mid \exists g_1,\ldots,g_k.P \text{ occurs in } C \text{ or } \mathcal{T}\}.$$

Now define the set $\Psi \subseteq 2^\Gamma$ as follows:

$$\Psi := \{\Phi \mid \Phi \subseteq \Gamma \text{ and } \bigwedge_{\exists g_1,\ldots,g_k.P\in\Phi} P(x_{g_1},\ldots,x_{g_k}) \text{ is unsatisfiable}\}$$

where x_g is a variable for each $g \in \mathsf{N_{cF}}$. Intuitively, each element of Ψ describes an inconsistent "configuration" of relevant concrete domain concepts, i.e., a configuration that cannot appear in models of C and $\mathcal{T}$. For each $D \in \Gamma$, let A_D be a concept name not occurring in C or $\mathcal{T}$. Similarly, for each concrete feature g appearing in C and $\mathcal{T}$, let B_g be a concept name not appearing in C and $\mathcal{T}$. If D is an $\mathcal{ALC}f(\mathcal{D})$-concept, then we use $D^\$$ to denote the result of replacing each subconcept

1. $\exists g_1,\ldots,g_k.P$ of D with $A_{\exists g_1,\ldots,g_k.P}$ and

2. each subconcept $g\uparrow$ of D with $\neg B_g$.

Analogously, if $\mathcal{T}$ is an $\mathcal{ALC}f(\mathcal{D})$-TBox, then we use $\mathcal{T}^\$$ to denote the result of replacing every concept D in $\mathcal{T}$ with $D^\$$. Intuitively, the concepts A_D are surrogates for the concrete domain constructor, and the concepts B_g represent the presence of successors for concrete features g. Define a new TBox $\mathcal{T}'$ as follows:

$$\mathcal{T}' := \mathcal{T}^\$ \cup \Big\{ \top \doteq \bigsqcap_{\Phi \in \Psi} \neg \bigsqcap_{D \in \Phi} A_D \ , \ \ \top \doteq \bigsqcap_{\exists g_1,\ldots,g_k.P \in \Gamma} \big(A_{\exists g_1,\ldots,g_k.P} \rightarrow B_{g_1} \sqcap \cdots \sqcap B_{g_k}\big) \Big\}$$

We claim that C is satisfiable w.r.t. $\mathcal{T}$ iff $C^\$$ is satisfiable w.r.t. $\mathcal{T}'$. First for the "if" direction. Let $\mathcal{I}$ be a model of $C^\$$ and $\mathcal{T}'$. For every $d \in \Delta_\mathcal{I}$, define a predicate conjunction

$$\zeta_d := \bigwedge_{A_{\exists g_1,\ldots,g_k.P} \text{ with } d \in (A_{\exists g_1,\ldots,g_k.P})^\mathcal{I}} P(x_{g_1}, \ldots, x_{g_k}).$$

Since it follows from the definition of Ψ and $\mathcal{T}'$ that ζ_d is satisfiable, we may fix a solution δ_d. Define an $\mathcal{ALC}f(\mathcal{D})$-interpretation $\mathcal{J}$ by setting

- $\Delta_\mathcal{J} := \Delta_\mathcal{I}$,
- $A^\mathcal{J} := A^\mathcal{I}$ for $A \in \mathsf{N_C}$,
- $R^\mathcal{J} := R^\mathcal{I}$ for $R \in \mathsf{N_R}$, and
- $g^\mathcal{J}(d) := \delta_d(x_g)$ if x_g appears in ζ_d for $g \in \mathsf{N_{cF}}$ and $d \in \Delta_\mathcal{J}$.

It is straightforward to check by structural induction that, for all $d \in \Delta_\mathcal{I}$ and $D \in \mathsf{sub}(C, \mathcal{T})$, $d \in (D^\$)^\mathcal{I}$ implies $d \in D^\mathcal{J}$ (it is crucial here that C and $\mathcal{T}$ are in NNF). Since all concept equations in $\mathcal{T}$ are of the form $D \doteq \top$, this implies that $\mathcal{J}$ is a model of C and $\mathcal{T}$.

For showing the "only if" direction, it is straightforward to convert models $\mathcal{I}$ of C and $\mathcal{T}$ into models $\mathcal{J}$ of $C^\$$ and $\mathcal{T}'$ by setting $(A_{\exists g_1,\ldots,g_k.P})^\mathcal{J} := (\exists g_1, \ldots, g_k.P)^\mathcal{I}$ for all surrogates $A_{\exists g_1,\ldots,g_k.P}$ and $B_g^\mathcal{J} := (\neg g\uparrow)^\mathcal{I}$ for all concepts B_g. ❑

Note that we were able to obtain this result only since we considered the restricted concrete domain constructor: the semantics of the general constructor involves more than a single domain element and also their relationship via abstract features. This complex semantics cannot be simulated using concept names. As we shall see in Chapter 6, satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t. general TBoxes is in many cases undecidable.

The above proof illustrates that the restricted concrete domain constructor does not significantly add to the expressive power of $\mathcal{ALC}$. It does, however, not provide us with a tight upper complexity bound: a lower ExpTime-bound stems from $\mathcal{ALC}$ with general TBoxes [Schild 1991], but the algorithm induced by the proof needs double exponential time in the worst case since the size of $\mathcal{T}'$ may be exponential in the size of C and $\mathcal{T}$. To close this gap, we sketch another decision procedure that is less illustrative w.r.t. expressivity but yields an ExpTime upper bound for $\mathcal{ALC}f(\mathcal{D})$-concept satisfiability.

Theorem 2.15. *Let $\mathcal{D}$ be an admissible concrete domain.*

1. *If $\mathcal{D}$-satisfiability is in* ExpTime, *then satisfiability of $\mathcal{ALC}f(\mathcal{D})$-concepts w.r.t. general TBoxes is* ExpTime*-complete.*

2. *If $\mathcal{D}$-satisfiability is in* $\mathrm{C} \in \{\text{NExpTime}, \text{ExpSpace}\}$, *then the satisfiability of $\mathcal{ALC}f(\mathcal{D})$-concepts w.r.t. general TBoxes is also in* C.

Proof. We prove the upper bounds using a standard elimination technique from Modal Logic that can, e.g., be found in [Halpern & Moses 1992; Spaan 1993a]. First for Point 1 of the theorem: fix a concrete domain $\mathcal{D}$ for which $\mathcal{D}$-satisfiability is in ExpTime. Let C be an $\mathcal{ALC}f(\mathcal{D})$-concept and $\mathcal{T}$ an $\mathcal{ALC}f(\mathcal{D})$-TBox, both in NNF. A *concept type* t for $(C, \mathcal{T})$ is a maximal subset of $\mathsf{sub}(C, \mathcal{T})$ such that

1. $\neg A \in t$ iff $A \notin t$ for all $\neg A \in \mathsf{sub}(C, \mathcal{T})$,
2. $D_1 \sqcap D_2 \in t$ iff $D_1, D_2 \in t$ for all $D_1 \sqcap D_2 \in \mathsf{sub}(C, \mathcal{T})$,
3. $D_1 \sqcup D_2 \in t$ iff $D_1 \in t$ or $D_2 \in t$ for all $D_1 \sqcup D_2 \in \mathsf{sub}(C, \mathcal{T})$,
4. $D_1 \in t$ iff $D_2 \in t$ for all $D_1 \doteq D_2 \in \mathcal{T}$,
5. if $\exists g_1, \ldots, g_k.P \in t$, then $\{g_1\uparrow, \ldots, g_k\uparrow\} \cap t = \emptyset$, and
6. $\displaystyle\bigwedge_{(\exists g_1,\ldots,g_k.P) \in t} P(x_{g_1}, \ldots, x_{g_k})$ is satisfiable (where x_g is a variable for each $g \in \mathsf{N_{cF}}$).

We use T to denote the set of all concept types for $(C, \mathcal{T})$ and n as an abbreviation for $|C| + |\mathcal{T}|$. Obviously, $|T| \leq 2^{|\mathsf{sub}(C,\mathcal{T})|}$ and hence $|T| \leq 2^n$. Moreover, T can be constructed in time exponential in n since $\mathcal{D}$-satisfiability can be decided in exponential time.

The set T is the starting point for the satisfiability algorithm to be devised: the algorithm first computes T and then repeatedly deletes concept types that, intuitively, cannot be "instantiated". More precisely, the algorithm computes a sequence of sets of concept types $T = T_0 \supseteq T_1 \supseteq T_2 \supseteq \cdots$ as follows:

$$
\begin{aligned}
T_i := T_{i-1} \setminus \Big(& \{t \in T_{i-1} \mid \text{there exists } (\exists R.D) \in t \text{ with } R \in \mathsf{N_R} \setminus \mathsf{N_{aF}} \text{ s.t. there} \\
& \qquad\qquad\quad \text{is no } t' \in T_{i-1} \text{ with } \{D\} \cup \{E \mid (\forall R.E) \in t\} \subseteq t'\} \cup \\
& \{t \in T_{i-1} \mid \text{there exists an } (\exists f.D) \in t \text{ with } f \in \mathsf{N_{aF}} \text{ s.t. there is no} \\
& \qquad\qquad\quad t' \in T_{i-1} \text{ with } \{E \mid (\forall f.E) \in t \text{ or } (\exists f.E) \in t\} \subseteq t'\} \Big).
\end{aligned}
$$

The algorithm terminates iff $T_k = T_{k-1}$. It returns $\mathsf{satisfiable}$ if there is some $t \in T_k$ with $C \in t$ and $\mathsf{unsatisfiable}$ otherwise.

We now prove the correctness of the algorithm. If it returns $\mathsf{satisfiable}$, we may use the set T_k to define a model $\mathcal{I}$ of C w.r.t. $\mathcal{T}$. Assume an ordering on the set of types T_k and fix a solution δ_t for each predicate conjunction ζ_t corresponding to a type t as in Point 6 above. Then $\mathcal{I}$ is defined as follows:

- $\Delta_\mathcal{I} := T_k$,
- $A^\mathcal{I} := \{t \mid A \in t\}$ for all $A \in \mathsf{N_C}$,
- $R^\mathcal{I} := \{(t, t') \mid (\forall R.D) \in t \text{ implies } D \in t'\}$ for all $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$,
- $f^\mathcal{I}(t) := t'$ if t' is the minimal type (w.r.t. the assumed ordering) satisfying $\{D \mid (\exists f.D) \in t \text{ or } (\forall f.D) \in t\} \subseteq t'$,
- $g^\mathcal{I}(t) := \delta_t(x_g)$ if x_g occurs in ζ_t.

Concerning the definition of $f^\mathcal{I}$, note that, if t contains no concept of the form $\exists f.D$, then there may exist no type t' as required and hence $f^\mathcal{I}$ may be undefined. It is straightforward to show by structural induction that $D \in t$ implies $t \in D^\mathcal{I}$ for all $t \in T_k$ and $D \in \mathsf{sub}(C, \mathcal{T})$. It follows that $\mathcal{I}$ is a model of C w.r.t. $\mathcal{T}$.

On the other hand, if there is a model $\mathcal{I}$ of C and $\mathcal{T}$, then the algorithm returns satisfiable. This may be proved by showing that concept types t for which there exists a $d \in \Delta_\mathcal{I}$ such that $t = \{D \in \mathsf{sub}(C, \mathcal{T}) \mid d \in D^\mathcal{I}\}$ are never deleted (using induction on the number k of steps made by the algorithm). It then follows immediately that there exists a $t \in T_k$ with $C \in t$.

Since $|T| \leq 2^n$ and at least one type is eliminated in each elimination round, the algorithm clearly stops after at most 2^n rounds. Together with the fact that the initial set of types T can be constructed in time exponential in n, we thus have proved an ExpTime upper bound for $\mathcal{ALC}f(\mathcal{D})$-concept satisfiability w.r.t. general TBoxes for the case that $\mathcal{D}$-satisfiability is in ExpTime. Together with the ExpTime lower bound for $\mathcal{ALC}$-concept satisfiability w.r.t. general TBoxes from [Schild 1991], we obtain Point 1 of the Theorem.

Now let $\mathcal{D}$-satisfiability be in C $\in$ {NExpTime, ExpSpace}. Since the number of $\mathcal{D}$-satisfiability tests performed by the above algorithm is at most exponential in n and the size of each tested conjunction is linear in n, it is straightforward to check that $\mathcal{ALC}(\mathcal{D})$-concept satisfiability w.r.t. TBoxes is in C.

❑

Since lower complexity bounds transfer from $\mathcal{D}$-satisfiability to $\mathcal{ALC}f(\mathcal{D})$-concept satisfiability, this theorem yields tight complexity bounds if $\mathcal{D}$-satisfiability is NExpTime-complete or ExpSpace-complete.

Both algorithms sketched above can presumably not be efficiently implemented in DL systems as demanded at the beginning of this chapter. It is, however, not hard to devise a tableau algorithm for $\mathcal{ALC}(\mathcal{D})$ with general TBoxes if $\mathcal{D}$ is unary or if the concrete domain constructor is restricted to concrete features [Horrocks & Sattler 2001; Haarslev *et al.* 2001]. As already mentioned, tableau algorithms are usually amenable to optimizations and well-suited for implementation. This nice behaviour of the restricted concrete domain constructor is investigated in [Baader *et al.* 2002a], where it is explained by the fact that the extension of a Description Logic $\mathcal{L}$ with the restricted concrete domain constructor can be viewed as the "fusion" of $\mathcal{L}$ with a rather trivial logic. Fusions of Description Logics are discussed in some more detail in Section 5.6. We did not use a tableau algorithm in this section since, as demonstrated by Donini and Massacci in [2000], tableau algorithms are a rather bad tool for establishing ExpTime upper bounds. It is out of scope to also investigate the complexity of ABox consistency here, but we conjecture that the complexity of this problem is identical to the complexity of concept satisfiability.

2.4.2 Expressive Concrete Domains

We now present an example from the other end of the expressivity spectrum of concrete domains. The rather expressive concrete domain R, which was first proposed by Baader and Hanschke in [1991a], is based on the first order theory of real closed fields or, in other words, on Tarski algebra [Tarski 1951]. Formally, it is defined as follows:

- the set Δ_{R} is $\mathbb{R}$ and
- the set Φ_{R} of predicates is built by first order means (i.e., by using logical connectives and quantifiers) from equalities and inequalities (using $=$, $\neq$, $<$, $>$, $\leq$, or $\geq$) between integer polynomials in several indeterminates.

It is not hard to check that Φ_{R} is closed under negation and contains a predicate for Δ_{R} (e.g., $x = x$). Since logical conjunction is available for building complex predicates, satisfiability of finite conjunctions of R-predicates can be reduced to satisfiability of single R-predicates. It was shown in [Mayr & Meyer 1982] that this problem, i.e., satisfiability of formulas in the first order theory of real closed fields, is ExpSpace-complete. Summing up, we can conclude that R is admissible.

In [Baader & Hanschke 1992], the appropriateness of $\mathcal{ALC}(\mathsf{R})$ for modelling rotational-symmetric workpieces of CNC lathe machines is demonstrated. Indeed, $\mathcal{ALC}(\mathsf{R})$ seems very appropriate for defining concepts related to geometry and stereometry. For example, we can describe the edge ratio in right-angled triangles by writing

$$\exists \mathsf{edge1}, \mathsf{edge2}, \mathsf{edge3}.x^2 + y^2 = z^2.$$

Here, $x^2 + y^2 = z^2$ is a ternary predicate from Φ_{R} with first argument x, second argument y, and third argument z. Analogously, we can describe the surface of a cube as

$$\exists \mathsf{edgelength}, \mathsf{surface}.y = 6x^2.$$

Note that R allows to assign precise values to concrete feature successors by using predicates such as $x = 3$. However, R also has several limitations w.r.t. expressivity. For example, we cannot define a predicate stating "$x \in \mathbb{N}$" or "$x \in \mathbb{Z}$" since this would allow to express Hilbert's tenth problem using an R-predicate thus contradicting the decidability of R-satisfiability [Davis 1973].

We obtain tight complexity bounds for the satisfiability of $\mathcal{ALC}(\mathsf{R})$-concepts in a more general setting in Chapters 3 (without TBoxes) and 5 (with acyclic TBoxes).

2.4.3 Temporal Concrete Domains

During the last 10 years, various approaches to temporal reasoning with Description Logics have been proposed [Artale & Franconi 2001]. An important one is to use a

Description Logic with a temporal concrete domain, which has first been suggested in [Lutz *et al.* 1997]. In this section, we introduce several such temporal concrete domains.

When defining a temporal logic, several ontological decisions have to be made. For example, a temporal structure needs to be fixed and one has to choose whether time points or time intervals are the basic temporal entity [Gabbay *et al.* 1994; 2001]. In this thesis, a *temporal structure* is a set T equipped with a a strict total ordering $\prec$. The elements of T are called *time points*. An *interval* over T is a pair (t_1, t_2) of elements of T such that $t_1 \prec t_2$. Hence, we consider linear time (as opposed to branching time) with either bounded or unbounded past and future. In this section, we generally use the temporal structure $(\mathbb{R}, <)$ for defining concrete domains. However, all concrete domains can also be defined with other structures such as $(\mathbb{N}, <)$, $(\mathbb{Z}, <)$, or even $(\{0, \ldots, 10\}, <)$.

Let us start with defining a concrete domain P that uses points as its basic temporal entity. Set

$$\begin{array}{lcl} \Delta_\mathsf{P} & := & \mathbb{R} \\ \Phi_\mathsf{P} & := & \{\top_\mathsf{P}, \bot_\mathsf{P}, =, \neq, <, >, \leq, \geq\} \end{array}$$

where $\top_\mathsf{P}$ and $\bot_\mathsf{P}$ are unary predicates, the other predicates are binary, and all predicates have the obvious meaning induced by the usual total ordering "$<$" on the reals. Obviously, Φ_P contains a predicate for Δ_P and is closed under negation. As follows from results in [van Beek & Cohen 1990; Ladkin & Maddux 1994], P-satisfiability is decidable in deterministic polynomial time, and hence P is admissible. Using P, we can define the concept

$$\mathsf{Process} \sqcap \exists \mathsf{start}, (\mathsf{subprocess}\ \mathsf{start}).<$$

describing processes which have a subprocess that starts after the start of the "mother" process.

Let us now define an interval-based concrete domain. The main advantage of interval-based concrete domains over point-based ones is that the former provide for a richer set of predicates. More precisely, we define a concrete domain $\mathtt{I}$ that is based on the set of 13 interval relations usually referred to as Allen relations [Allen 1983]. The Allen relations describe the relationship between any two intervals over some temporal structure and can be defined in terms of the interval endpoints. We refer to [Allen 1983] for such a definition and confine ourselves to the graphical presentation of the relations given in Figure 2.7. In what follows, we denote the set of Allen relations by $\mathfrak{A}$. Let $(T, \prec)$ be a temporal structure. The set $\mathfrak{A}$ has several important properties of which we state two explicitly:

- it is *exhaustive*, i.e., for each $t_1, t_2 \in T$, there exists at least one $\mathsf{r} \in \mathfrak{A}$ such that $t_1 \,\mathsf{r}\, t_2$;
- the relations are *mutually exclusive*, i.e., for each $t_1, t_2 \in T$, there exists at most one $\mathsf{r} \in \mathfrak{A}$ such that $t_1 \,\mathsf{r}\, t_2$.

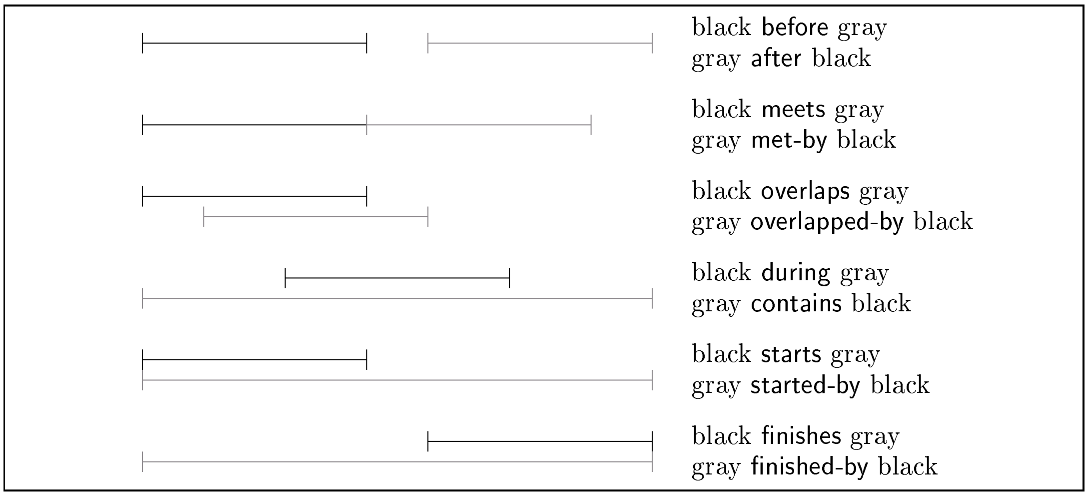

Figure 2.7: The Allen relations (without equal).

The concrete domain $\mathtt{I}$ is based on the reals and defined as follows:

$$\begin{aligned}
\Delta_{\mathtt{I}} &:= \{(t_1, t_2) \mid t_1, t_2 \in \mathbb{R} \text{ and } t_1 < t_2\} \\
\Phi_{\mathtt{I}} &:= \{\top_{\mathtt{I}}, \bot_{\mathtt{I}}, \mathsf{equals}, \mathsf{before}, \mathsf{after}, \mathsf{meets}, \mathsf{met\text{-}by}, \mathsf{overlaps}, \mathsf{overlapped\text{-}by}, \mathsf{during}, \\
&\quad\ \mathsf{contains}, \mathsf{starts}, \mathsf{started\text{-}by}, \mathsf{finishes}, \mathsf{finished\text{-}by}, \\
&\quad\ \mathsf{nequals}, \mathsf{nbefore}, \mathsf{nafter}, \mathsf{nmeets}, \mathsf{nmet\text{-}by}, \mathsf{noverlaps}, \mathsf{noverlapped\text{-}by}, \mathsf{nduring}, \\
&\quad\ \mathsf{ncontains}, \mathsf{nstarts}, \mathsf{nstarted\text{-}by}, \mathsf{nfinishes}, \mathsf{nfinished\text{-}by}\}
\end{aligned}$$

Again, $\top_{\mathtt{I}}$ and $\bot_{\mathtt{I}}$ are the only unary predicates and all other predicates are binary. The binary predicates not prefixed with "n" are interpreted as the corresponding Allen relations and the predicates that are prefixed with "n" are interpreted as the negation of the corresponding Allen relations.

Let us analyze the complexity of $\mathtt{I}$-satisfiability. A *generalized Allen relation* is a disjunction $\mathsf{r}_1 \vee \cdots \vee \mathsf{r}_k$, where each r_i is one of the 13 Allen relations introduced previously. To distinguish them from the generalized relations, we sometimes call the 13 relations *base relations*. Note that the predicates in $\Phi_{\mathtt{I}}$ starting with "n" correspond to generalized relations, e.g., nbefore corresponds to

$$\begin{gathered}
\mathsf{after} \vee \mathsf{meets} \vee \mathsf{met\text{-}by} \vee \mathsf{overlaps} \vee \mathsf{overlapped\text{-}by} \vee \mathsf{during} \\
\vee \mathsf{contains} \vee \mathsf{starts} \vee \mathsf{started\text{-}by} \vee \mathsf{finishes} \vee \mathsf{finished\text{-}by}.
\end{gathered}$$

In the area of qualitative temporal reasoning, so-called *interval algebra networks (IANs)* are used to describe temporal configurations [van Beek & Cohen 1990]. An IAN is a directed graph whose edges are labeled with generalized Allen relations and it is *satisfiable* iff there exists a function mapping each node to an interval over $\mathbb{R}$ (i.e., to an element of $\Delta_{\mathtt{I}}$) such that the edge labels are "respected". We will determine the complexity of $\mathtt{I}$-satisfiability by using results concerning IAN-satisfiability. It is easily seen that $\mathtt{I}$-satisfiability can be reduced to the satisfiability of IANs. The converse does also hold if only relations from $\Phi_{\mathtt{I}}$ are admitted as IAN-edge labels. This restriction is important since the complexity of IAN-satisfiability depends on which generalized relations are allowed as edge labels. If all generalized relations are admitted, IAN-satisfiability is known to be NP-complete [Vilain *et al.* 1990]. This clearly implies that $\mathtt{I}$-satisfiability is in NP. Moreover, $\mathtt{I}$-satisfiability is NP-hard since (i) the ORD-Horn class of generalized Allen relations defined by Nebel and Bürckert does not contain all the relations corresponding to predicates in $\Phi_\mathtt{I}$ and (ii) Nebel and Bürckert show that ORD-Horn is the unique greatest class of generalized relations that contains all 13 base relations and for which IAN-satisfiability is tractable [1995]. Given the NP-completeness, it is easy to check that $\mathtt{I}$ is admissible.

As an example for knowledge representation with $\mathcal{ALC}(\mathtt{I})$, consider the $\mathcal{ALC}(\mathtt{I})$-concept

$$\begin{array}{l}\mathsf{Process}\\ \sqcap\ \exists(\mathsf{subp1\ time}),(\mathsf{subp2\ time}).\mathsf{before}\\ \sqcap\ \exists(\mathsf{subp2\ time}),(\mathsf{subp3\ time}).\mathsf{before}\\ \sqcap\ \big(\exists(\mathsf{subp1\ time}),(\mathsf{subp3\ time}).\mathsf{overlaps} \sqcup \exists(\mathsf{subp1\ time}),(\mathsf{subp3\ time}).\mathsf{meets}\big)\end{array}$$

describing a process with three subprocesses and the temporal relation between these subprocesses. The third conjunct of this concept demonstrates that there is no need to introduce additional generalized Allen relations into the concrete domain since this would not enhance the expressivity of the language. For example, we can replace $\exists u_1,u_2.\mathsf{overlaps}{\vee}\mathsf{meets}$ (where $\mathsf{overlaps}{\vee}\mathsf{meets}$ is from a fictitious concrete domain providing predicates for generalized relations) by the $\mathcal{ALC}(\mathtt{I})$-concept $\exists u_1,u_2.\mathsf{overlaps} \sqcup \exists u_1,u_2.\mathsf{meets}$. Moreover, the concept illustrates that the Allen relations have rich relational properties that must be taken into account for reasoning: if the first subprocess is before the second one and the second subprocess is before the third one, then it can neither be the case that the first subprocess $\mathsf{overlaps}$ the third one nor that the first subprocess meets the third one. Hence, the above concept is unsatisfiable.

It is interesting to note that, despite the different complexity of P-satisfiability and $\mathtt{I}$-satisfiability, $\mathcal{ALC}(\mathsf{P})$ has just the same expressivity as $\mathcal{ALC}(\mathtt{I})$. We illustrate this by sketching a straightforward reduction of $\mathcal{ALC}(\mathsf{P})$-concept satisfiability to $\mathcal{ALC}(\mathtt{I})$-concept satisfiability and vice versa (again, we do not define a formal notion of expressivity since this is out of the scope of this thesis).

1. Let C be an $\mathcal{ALC}(\mathsf{P})$-concept. A corresponding $\mathcal{ALC}(\mathtt{I})$-concept C^* is obtained from C by replacing

$$\begin{array}{rll rll}\exists u.\top_\mathsf{P} & \text{with} & \exists u.\top_\mathtt{I} & \exists u.\bot_\mathsf{P} & \text{with} & \exists u.\bot_\mathtt{I}\\ \exists u_1,u_2.< & \text{with} & \exists u_1,u_2.\mathsf{before} & \exists u_1,u_2.> & \text{with} & \exists u_1,u_2.\mathsf{after}\\ \exists u_1,u_2.= & \text{with} & \exists u_1,u_2.\mathsf{equals} & & & \\ \exists u_1,u_2.\neq & \text{with} & \exists u_1,u_2.\mathsf{before} \sqcup \exists u_1,u_2.\mathsf{after} & & & \\ \exists u_1,u_2.\leq & \text{with} & \exists u_1,u_2.\mathsf{before} \sqcup \exists u_1,u_2.\mathsf{equals} & & & \\ \exists u_1,u_2.\geq & \text{with} & \exists u_1,u_2.\mathsf{after} \sqcup \exists u_1,u_2.\mathsf{equals} & & & \end{array}$$

It is not hard to see that models of C can be straightforwardly translated into models of C^* and vice versa.

2. Let C be an $\mathcal{ALC}(\mathtt{I})$-concept. We define a corresponding $\mathcal{ALC}(\mathsf{P})$-concept C^* such that models of C can be easily translated into models of C^* and vice versa. For every path $u = f_1 \ldots f_k g$ in C, we use two paths $u^\ell = f_1 \cdots f_k g^\ell$ and $u^r = f_1 \cdots f_k g^r$ in C^*. Intuitively, u^ℓ describes the left endpoint of the interval described by u and u^r describes the right endpoint. The concept C^* is obtained from C by replacing

   - $\exists u.\top_\mathtt{I}$ with $\exists u^\ell.\top_\mathsf{P} \sqcap \exists u^r.\top_\mathsf{P}$,
   - $\exists u.\bot_\mathtt{I}$ with $\exists u^\ell.\bot_\mathsf{P} \sqcap \exists u^r.\bot_\mathsf{P}$,
   - $\exists u_1, u_2.\mathsf{r}$, where r is a base relation, with a concept expressing (1) the relationship between left and right endpoints of intervals and (2) the relation r in terms of endpoints, and
   - $\exists u_1, u_2.\mathsf{nr}$, where r is a base relation, with $\bigsqcup\limits_{\mathsf{r}' \in \mathfrak{A} \setminus \{\mathsf{r}\}} (\exists u_1, u_2.\mathsf{r}')^*$.

   For example,

   $$\exists u_1, u_2.\mathsf{after} \text{ is replaced with } \exists u_1^\ell, u_1^r.< \sqcap \exists u_2^\ell, u_2^r.< \sqcap \exists u_1^\ell, u_2^r.>.$$

   and

   $$\exists u_1, u_2.\mathsf{nbefore} \text{ is replaced with } \bigsqcup_{\mathsf{r}' \in \mathfrak{A} \setminus \{\mathsf{before}\}} (\exists u_1, u_2.\mathsf{r}')^*.$$

   Again, it is easy to prove that C^* is as required.

We will pick up the theme of temporal reasoning with Description Logics again in Chapter 6 where more modelling examples and a thorough discussion of related approaches are presented.

There exists a close and important relative of the concrete domain $\mathtt{I}$ whose existence should at least be mentioned here: in qualitative spatial reasoning, a set of 8 relations called RCC-8 is used to describe the relationship between regions in (topological) space [Randell *et al.* 1992; Bennett 1997]. In several aspects, such as exhaustiveness and mutual exclusiveness, these relations resemble the Allen relations. Hence, it is natural to define a concrete domain based on RCC-8 in analogy to the definition of $\mathtt{I}$. We cannot go into more detail here and refer to [Haarslev *et al.* 1999] for a formal definition of such a spatial concrete domain. In general, it seems that the concrete domain $\mathtt{I}$ and the RCC-8 based concrete domain "behave" rather similarly, i.e., results for a Description Logics $\mathcal{L}$ equipped with the concrete domain $\mathtt{I}$ also hold for the Description Logic obtained from $\mathcal{L}$ by replacing the concrete domain $\mathtt{I}$ with the RCC-8 based concrete domain and vice versa.

Chapter 3

Reasoning with $\mathcal{ALCF}(\mathcal{D})$

This chapter is devoted to establishing tight complexity bounds for reasoning with the basic Description Logic with concrete domains, $\mathcal{ALC}(\mathcal{D})$. In [1991a], Baader and Hanschke devise a tableau algorithm that decides $\mathcal{ALC}(\mathcal{D})$-concept satisfiability (provided that the concrete domain $\mathcal{D}$ is admissible) and needs both exponential time and exponential space in the worst case. Hence, the upper complexity bound induced by this algorithm does not match the known PSPACE lower bound which stems from $\mathcal{ALC}$-concept satisfiability [Schmidt-Schauß & Smolka 1991]. In this chapter, we develop a tableau algorithm that is similar to the one of Baader and Hanschke but uses the *tracing* technique from [Schmidt-Schauß & Smolka 1991] to reduce the space requirement. This algorithm yields a PSPACE upper bound for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in PSPACE. We then use the *precompletion* technique from [Hollunder 1996] to extend the PSPACE upper bound to ABox consistency. For both concept satisfiability and ABox consistency, we assume that no TBoxes are present, since, as we shall prove in Chapter 5, the presence of the rather weak acyclic TBoxes is already sufficient to make concept satisfiability NEXPTIME-hard for a large class of concrete domains. When using the tracing technique, which is explained in detail below, we need to take special care of the fact that *sequences* of features may be used inside the concrete domain constructor. It turns out that the necessary adaptations are quite similar to the ones needed for the Description Logic $\mathcal{ALCF}$ [Hollunder & Nutt 1990]. Because of this and because we treat the complexity of (extensions of) $\mathcal{ALCF}$ as a side-track in this thesis, we prove the afore mentioned upper bounds for $\mathcal{ALCF}(\mathcal{D})$, i.e., for the extension of $\mathcal{ALC}$ with both concrete domains and feature (dis)agreements.

3.1 Concept Satisfiability

In this section, we devise a tableau algorithm for deciding satisfiability of $\mathcal{ALCF}(\mathcal{D})$-concepts that needs at most polynomial space if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in PSPACE. The algorithm also yields tight complexity bounds if $\mathcal{D}$-satisfiability is NEXPTIME-compete or EXPSPACE-complete. Syntax and semantics of the Description Logic $\mathcal{ALCF}(\mathcal{D})$ are defined in the obvious way, see Sections 2.1.2 and 2.3.1.

Note, however, that the feature (dis)agreement constructors take *abstract* paths as arguments while the concrete domain constructor takes *concrete* paths as arguments. Hence, feature (dis)agreements are not concerned with elements from the concrete domain. However, if the concrete domain provides for equality and inequality predicates, it is obvious that we can express (dis)agreement of concrete paths using the concrete domain constructor.

3.1.1 Overview

Since there exist rather different variants of tableau algorithms in Modal Logic and First Order Logic, we call the family of tableau algorithms commonly used for Description Logics (including all tableau algorithms in this thesis) *completion algorithms*. The reader is referred to [D'Agostino *et al.* 1999] for an overview over tableau algorithms in general and to [Baader & Sattler 2000] for an overview over completion algorithms. The closest relative in Modal Logic of completion algorithms are so-called labeled tableaux [Goré 1999].

Completion algorithms are characterized by an underlying data structure, a set of *completion rules* operating on this data structure, and a (possibly trivial) strategy for applying the rules. In principle, a completion algorithm starts with an initial data structure induced by the concept D whose satisfiability is to be decided and repeatedly applies completion rules according to the strategy. Repeated rule application can be thought of as making implicit knowledge explicit or as constructing a canonical model for the input concept (represented in terms of the underlying data structure). The algorithm stops if it encounters a contradiction or if no more completion rule is applicable. It returns satisfiable iff the latter is the case *and* no obvious contradiction was found, i.e., if the algorithm succeeds in constructing a model for the input concept. Otherwise, it returns unsatisfiable.

If a PSpace upper bound is to be proved using a completion algorithm, some additional efforts have to be made. To simplify discussion, let us consider the logic $\mathcal{ALC}$ for the moment. A naive completion algorithm for $\mathcal{ALC}$ does not yield a PSpace upper bound since there exist satisfiable $\mathcal{ALC}$-concepts all of whose models are of size exponential in the concept length [Halpern & Moses 1992]. Thus, an algorithm keeping the entire (representation of a) model in memory needs exponential space in the worst case. However, there exists a well-known way to overcome this problem: the key observation is that canonical models $\mathcal{I}$ constructed by completion algorithms are *tree models*, i.e., they have the form of a tree if viewed as a graph with $\Delta_{\mathcal{I}}$ the set of vertices and $\bigcup_{R \in \mathsf{N_R}} R^{\mathcal{I}}$ the set of edges. It is sufficient to consider only such tree models since $\mathcal{ALC}$ has the *tree model property*, i.e., each satisfiable concept has a tree model [Halpern & Moses 1992]. To check for the existence of tree models for a given concept, we may try to construct one by performing depth-first search over role successors keeping only paths of the tree model in memory. Since, in the case of $\mathcal{ALC}$, the length of paths is at most polynomial in the length of the input concept [Halpern & Moses 1992], this technique—which is known as *tracing* [Schmidt-Schauß & Smolka 1991]—yields an algorithm that needs at most polynomial space in the worst case. Completion algorithms for $\mathcal{ALC}$-concept satisfiability that use tracing are very

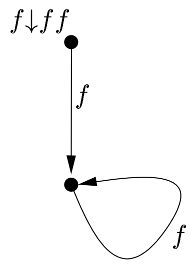

Figure 3.1: A model of the $\mathcal{ALCF}(\mathcal{D})$-concept $f{\downarrow}ff$.

similar to the well-known K-world algorithm from Modal Logic [Ladner 1977].

The tracing technique has to be modified to deal with $\mathcal{ALCF}(\mathcal{D})$-concepts since it is not hard to see that $\mathcal{ALCF}(\mathcal{D})$ does not enjoy the tree model property: for example, the concept $f{\downarrow}ff$ is satisfiable but has only non-tree models such as the one in Figure 3.1. The failure of the tree model property is obviously due to the feature (dis)agreement constructors and the "non-treen-shape" of models is due to substructures that are exclusively comprised of features (and not of roles from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$). Based on this observation, we define *generalized tree models.*

Definition 3.1 (Generalized Tree Model). Let $\mathcal{I}$ be a model of an $\mathcal{ALCF}(\mathcal{D})$-concept C and define a relation $\sim$ on $\Delta_\mathcal{I}$ as follows:

$d \sim e$ iff $d = e$ or there exists an abstract path $f_1 \cdots f_k$ and domain elements $d_0, \ldots, d_k \in \Delta_\mathcal{I}$ such that $d_0 = d$, $d_k = e$, and either $d_{i+1} = f_{i+1}^\mathcal{I}(d_i)$ or $d_i = f_{i+1}^\mathcal{I}(d_{i+1})$ for $i < k$.

It is easy to see that $\sim$ is an equivalence relation. By $[d]_\sim$, we denote the equivalence class of $d \in \Delta_\mathcal{I}$ w.r.t. $\sim$. The model $\mathcal{I}$ is a *generalized tree model* of C iff $\mathcal{I}$ is a model of C and the graph $(V_\mathcal{I}, E_\mathcal{I})$ defined as

$$\begin{array}{rcl} V_\mathcal{I} & := & \{[d]_\sim \mid d \in \Delta_\mathcal{I}\} \\ E_\mathcal{I} & := & \{([d]_\sim, [e]_\sim) \mid \exists d' \in [d]_\sim, e' \in [e]_\sim \text{ such that} \\ & & \qquad (d', e') \in R^\mathcal{I} \text{ for some } R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}\} \end{array}$$

is a tree. $\diamond$

It will be a byproduct of the results obtained in this section that $\mathcal{ALCF}(\mathcal{D})$ has the *generalized tree model property*, i.e., that every satisfiable $\mathcal{ALCF}(\mathcal{D})$-concept C has a generalized tree model. Note that the identification of some kind of tree model property is usually very helpful for devising decision procedures [Vardi 1997; Grädel 1999]. Our completion algorithm for $\mathcal{ALCF}(\mathcal{D})$ uses tracing on generalized tree models: it keeps only fragments of models $\mathcal{I}$ in memory that induce paths in the abstraction $(V_\mathcal{I}, E_\mathcal{I})$. Intuitively, such a fragment consists of a sequence of "clusters" of domain elements, where each cluster is an equivalence class w.r.t. the relation $\sim$, i.e., a set of elements connected by abstract features. Succeeding clusters in the sequence are connected by roles from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$. Fortunately, as we shall see later, there always exists a generalized tree model $\mathcal{I}$ in which the cardinality of clusters and the depth of the tree $(V_\mathcal{I}, E_\mathcal{I})$ is at most polynomial in the length of the input concept. We use these facts to devise a completion algorithm for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability running in polynomial space.

Before we start the formal presentation, let us briefly discuss how completion algorithms can deal with concrete domains. Along with constructing the "abstract part" of the model for the input concept, Baader and Hanschke's completion algorithm for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability builds up a predicate conjunction describing the "concrete part" of the constructed model [Baader & Hanschke 1991a]. This predicate conjunction is required to be satisfiable in order for the constructed model to be non-contradictory (see the general description of completion algorithms above). However, the number of conjuncts in the predicate conjunction is exponential in the length of the input concept in the worst case, which is another reason why Baader and Hanschke's algorithm needs exponential space. In our algorithm, we address this problem as follows: domain elements that are in different clusters of the generalized tree model are not connected through abstract paths. Hence, it cannot be enforced that concrete successors of domain elements from different clusters are related by a concrete predicate. This, in turn, means that it is sufficient to *separately* check the satisfiability of predicate conjunctions associated with clusters. Since the size of predicate conjunctions associated with a cluster is at most polynomial in the length of the input concept, this separate checking allows to devise a PSPACE algorithm (if $\mathcal{D}$-satisfiability is in PSPACE).

3.1.2 The Completion Algorithm

In the following, we assume that the concrete domain $\mathcal{D}$ is admissible and that concepts are in negation normal form. Every $\mathcal{ALCF}(\mathcal{D})$-concept C can be transformed into an equivalent one in NNF using the rewrite rules from Definition 2.3.1 together with the following ones:

$$\begin{array}{rcl}
\neg(\exists u_1, \ldots, u_n.P) & \rightsquigarrow & \exists u_1, \ldots, u_n.\overline{P} \sqcup u_1\uparrow \sqcup \cdots \sqcup u_n\uparrow \\
\neg(g\uparrow) & \rightsquigarrow & \exists g.\top_\mathcal{D} \\
\neg(p_1\uparrow p_2) & \rightsquigarrow & p_1\downarrow p_2 \sqcup \forall p_1.\bot \sqcup \forall p_2.\bot \\
\neg(p_1\downarrow p_2) & \rightsquigarrow & p_1\uparrow p_2 \sqcup \forall p_1.\bot \sqcup \forall p_2.\bot
\end{array}$$

Let us start the presentation of the completion algorithm by introducing $\mathcal{ALCF}(\mathcal{D})$-ABoxes as the underlying data structure.

Definition 3.2 ($\mathcal{ALCF}(\mathcal{D})$-ABox). Let C be an $\mathcal{ALCF}(\mathcal{D})$-concept, $R \in \mathsf{N_R}$ a role, p_1 and p_2 abstract paths, g a concrete feature, $a \in \mathsf{O_a}$, $x_1, \ldots, x_n \in \mathsf{O_c}$, and $P \in \Phi_\mathcal{D}$ with arity n. Then

$$a : C, \quad (a, b) : R, \quad (a, x) : g, \quad (x_1, \ldots, x_n) : P, \quad \text{and} \quad a \neq b$$

are $\mathcal{ALCF}(\mathcal{D})$-*assertions*. An $\mathcal{ALCF}(\mathcal{D})$-*ABox* is a finite set of $\mathcal{ALCF}(\mathcal{D})$-assertions. Since the definitions of the other kinds of assertions are already given in Definitions 2.7 and 2.10, we only note that an interpretation $\mathcal{I}$ satisfies an assertion $a \neq b$ iff $a^\mathcal{I} \neq b^\mathcal{I}$.

Let $\mathcal{A}$ be an $\mathcal{ALCF}(\mathcal{D})$-ABox, $a, b \in \mathsf{O_a}$ and $x \in \mathsf{O_c}$. We write $\mathcal{A}(a)$ to denote the set of concepts $\{C \mid a : C \in \mathcal{A}\}$. The abstract object b is called *R-successor of a in $\mathcal{A}$* iff $(a, b) : R$ is in $\mathcal{A}$. Analogously, x is a *g-successor* of a iff $(a, x) : g$ is in $\mathcal{A}$. The notion "successor" generalizes to abstract and concrete paths in the obvious way.

The *size* $|\alpha|$ of an assertion α is defined as $|C|$ if $\alpha = a : C$ and 1 otherwise. The size $|\mathcal{A}|$ of an ABox $\mathcal{A}$ is defined as the sum of the sizes of its assertions. ◇

Note that the notion "successor in ABoxes" as defined above is syntactical while the notion "successor in models" from Definition 2.2 is semantical. It should be obvious how ABoxes can be used to represent models. If the satisfiability of a concept D is to be decided, the completion algorithm is started with the *initial ABox* for D defined as $\mathcal{A}_D = \{a : D\}$. Before the completion rules are defined, we introduce an operation that is used in the formulation of the rules for introducing new assertions of the form $(a, b) : R$ and $(a, x) : g$.

Definition 3.3 ("+" operation). An abstract or concrete object is called *fresh* w.r.t. an ABox $\mathcal{A}$ if it does not appear in $\mathcal{A}$. Let $\mathcal{A}$ be an ABox. By

$$\mathcal{A} + aRb$$

with $a \in \mathsf{O_a}$ used in $\mathcal{A}$, $R \in \mathsf{N_R}$, and $b \in \mathsf{O_a}$ we denote the ABox $\mathcal{A} \cup \{(a, b) : R\}$. Similarly, we use

$$\mathcal{A} + agx$$

with $g \in \mathsf{N_{cF}}$ and $x \in \mathsf{O_c}$ to denote $\mathcal{A} \cup \{(a, x) : g\}$.

When nesting the + operation, we omit brackets writing, e.g., $\mathcal{A} + aR_1b + bR_2c$ for $(\mathcal{A} + aR_1b) + bR_2c$. Let $p = f_1 \cdots f_n$ be an abstract path (resp. $u = f_1 \cdots f_n g$ be a concrete path). By $\mathcal{A} + apb$ (resp. $\mathcal{A} + aux$), where $a \in \mathsf{O_a}$ is used in $\mathcal{A}$ and $b \in \mathsf{O_a}$ (resp. $x \in \mathsf{O_c}$), we denote the ABox $\mathcal{A}'$ which can be obtained from $\mathcal{A}$ by choosing distinct objects $b_1, \ldots, b_n \in \mathsf{O_a}$ which are fresh in $\mathcal{A}$ and setting

$$\begin{array}{rcl} \mathcal{A}' & := & \mathcal{A} + af_1b_1 + \cdots + b_{n-1}f_nb \\ (\text{resp. } \mathcal{A}' & := & \mathcal{A} + af_1b_1 + \cdots + b_{n-1}f_nb_n + b_ngx). \end{array}$$

◇

The completion rules can be found in Figure 3.2. Note that the R⊔ rule is nondeterministic, i.e., it has more than one possible outcome. Thus, the described completion algorithm is a nondeterministic decision procedure. Such an algorithm accepts its input (i.e. returns satisfiable) iff there is *some* way to make the nondeterministic decisions such that a positive result is obtained. A convenient way to think of nondeterministic rules is that they "guess" the correct outcome, i.e., if there is an outcome which, if chosen, leads to a positive result, then this outcome is in fact considered.

Most completion rules are standard and known from [Baader & Hanschke 1991b] and [Hollunder & Nutt 1990]. The R∃f and R∀f rules are special in that they only deal with concept $\exists f.C$ and $\forall f.C$ where f is an abstract feature. As we will see later, concepts $\exists R.C$ and $\forall R.C$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ are not treated by completion rules but through recursion calls of the algorithm. The Rfe rule also deserves some attention:

R$\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{A}(a)$ then $\mathcal{A} := \mathcal{A} \cup \{a : C_1, a : C_2\}$
R$\sqcup$	if $C_1 \sqcup C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \cap \mathcal{A}(a) = \emptyset$ then $\mathcal{A} := \mathcal{A} \cup \{a : C\}$ for some $C \in \{C_1, C_2\}$
R$\exists$f	if $\exists f.C \in \mathcal{A}(a)$ and there is no f-successor b of a with $C \in \mathcal{A}(b)$ then set $\mathcal{A} := (\mathcal{A} + afb) \cup \{b : C\}$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R$\forall$f	if $\forall f.C \in \mathcal{A}(a)$, b is an f-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$
Rc	if $\exists u_1, \ldots, u_n.P \in \mathcal{A}(a)$ and there exist no $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is u_i-successor of a for $1 \le i \le n$ and $(x_1, \ldots, x_n) : P \in \mathcal{A}$ then set $\mathcal{A} := (\mathcal{A} + au_1x_1 + \cdots + au_nx_n) \cup \{(x_1, \ldots, x_n) : P\}$ with $x_1, \ldots, x_n \in \mathsf{O_c}$ fresh in $\mathcal{A}$
R$\downarrow$	if $p_1{\downarrow}p_2 \in \mathcal{A}(a)$ and there is no b that is both a p_1-successor of a and a p_2-successor of a then set $\mathcal{A} := \mathcal{A} + ap_1b + ap_2b$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R$\uparrow$	if $p_1{\uparrow}p_2 \in \mathcal{A}(a)$ and there are no b_1, b_2 with b_1 p_1-successor of a, b_2 p_2-successor of a, and $(b_1 \neq b_2) \in \mathcal{A}$ then set $\mathcal{A} := (\mathcal{A} + ap_1b_1 + ap_2b_2) \cup \{(b_1 \neq b_2)\}$ for $b_1, b_2 \in \mathsf{O_a}$ distinct and fresh in $\mathcal{A}$
Rfe	if $\{(a, b) : f, (a, c) : f\} \subseteq \mathcal{A}$ and $b \neq c$ (resp. $\{(a, x) : g, (a, y) : g\} \subseteq \mathcal{A}$ and $x \neq y$) then replace b by c in $\mathcal{A}$ (resp. x by y)

Figure 3.2: Completion rules for $\mathcal{ALCF}(\mathcal{D})$.

it ensures that, for any object $a \in \mathsf{O_a}$, there exists at most a single f-successor for each $f \in \mathsf{N_{aF}}$ and at most a single g-successor for each $g \in \mathsf{N_{cF}}$. Redundant successors are eliminated by identification. This process is often referred to as *fork elimination* (hence the name of the rule). In many cases, fork elimination is not explicitly formulated as a completion rule but viewed as an integral part of the other completion rules. In the presence of feature (dis)agreements, this latter approach seems to be less transparent. Consider for example the ABox

$$\{a : \exists f_1.\top,\ a : \exists f_2.\top,\ a : f_1{\downarrow}f_2\}.$$

Assume the R$\exists$f rule is applied twice adding the assertions $(a, b) : f_1$ and $(a, c) : f_2$. Now, the R$\downarrow$ rule is applied adding $(a, b') : f_1$ and $(a, b') : f_2$. Clearly, we may now apply the Rfe rule to the assertions $(a, b) : f_1$ and $(a, b') : f_1$. Say the rule application replaces b' by b, and we obtain the ABox

$$\{a : \exists f_1.\top,\ a : \exists f_2.\top,\ a : f_1{\downarrow}f_2,\ (a, b) : f_1,\ (a, c) : f_2,\ (a, b) : f_2\}.$$

Obviously, we may now apply Rfe to $(a, c) : f_2$ and $(a, b) : f_2$ replacing b by c. Observe that this latter fork elimination does not involve any objects generated by

```
define procedure sat(A)
  A := fcompl(A)
  if A contains a clash then
    return unsatisfiable
  forall objects a in O_a and ∃R.C ∈ A(a) with R ∈ N_R \ N_aF do
    Fix b ∈ O_a
    if sat({b : C} ∪ {b : E | ∀R.E ∈ A(a)}) = unsatisfiable then
      return unsatisfiable
  return satisfiable

define procedure fcompl(A)
  while a rule from Figure 3.2 is applicable to A do
    Choose an applicable rule R s.t. R = Rfe if Rfe is applicable
    Apply R to A
  return A
```

Figure 3.3: The $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability algorithm.

the last "non-Rfe" rule application. To make such effects more transparent, we chose to formulate fork elimination as a separate rule.

Let us now formalize what it means for an ABox to be contradictory.

Definition 3.4 (Clash). With each ABox $\mathcal{A}$, we associate a predicate conjunction

$$\zeta_{\mathcal{A}} = \bigwedge_{(x_1,\ldots,x_n):P\in\mathcal{A}} P(x_1,\ldots,x_n).$$

The ABox $\mathcal{A}$ is called *concrete domain satisfiable* iff $\zeta_{\mathcal{A}}$ is satisfiable. It is said to contain a *clash* iff one of the following conditions applies:

1. $\{A,\neg A\} \subseteq \mathcal{A}(a)$ for a concept name A and object $a \in \mathsf{O_a}$,
2. $(a \neq a) \in \mathcal{A}$ for some object $a \in \mathsf{O_a}$,
3. $g\uparrow \in \mathcal{A}(a)$ for some $a \in \mathsf{O_a}$ such that there exists a g-successor of a, or
4. $\mathcal{A}$ is not concrete domain satisfiable.

If $\mathcal{A}$ does not contain a clash, then $\mathcal{A}$ is called *clash-free*. ◇

The completion algorithm itself can be found in Figure 3.3. We briefly summarize the strategy followed by the algorithm. The argument to sat is an ABox containing exactly one object $a \in \mathsf{O_a}$ and only assertions of the form $a : C$. The algorithm uses the fcompl function to create all feature successors of a, all feature successors of these feature successors and so on. However, fcompl does not generate any R-successors for roles $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. In other words, fcompl generates a cluster of objects as described in Section 3.1.1. After the call to the fcompl function, the algorithm makes a recursion

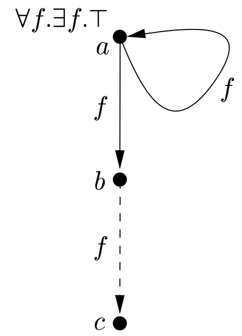

Figure 3.4: The "jojo" effect.

call for each object in the cluster and each role successor enforced via an $\exists R.C$ assertion (with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$). A single such recursion call corresponds to moving along a path in a generalized tree model, i.e, to moving to a successor cluster of the cluster under consideration. Each cluster of objects is checked separately for contradictions. Note that, due to Definition 3.4, checking for a clash involves checking whether the predicate conjunction $\zeta_{\mathcal{A}}$ is satisfiable. This, in turn, is a decidable problem since we assume $\mathcal{D}$ to be admissible.

Observe that fcompl applies the Rfe rule with highest priority. Without this strategy, the algorithm would not terminate: consider the ABox

$$\mathcal{A} = \{a : \forall f.\exists f.\top,\ (a,a) : f,\ (a,b) : f\}.$$

This ABox, which is depicted in the upper part of Figure 3.4, is encountered if, for example, the algorithm is started on the input concept $f'\downarrow f'f \sqcap \exists f'.(\forall f.\exists f.\top \sqcap \exists f.\top)$. Now assume that the completion rules are applied to $\mathcal{A}$ without giving Rfe the highest priority. This means that we can apply the R$\forall$f rule and obtain $b : \exists f.\top$. We can then apply R$\exists$f generating $(b,c) : f,\ c : \top$. Fork elimination may now identify a and b and thus we are back at the initial situation (possibly up to renaming). Clearly, this sequence of rule applications may be repeated indefinitely—the algorithm does not terminate. This "jojo" effect was also described, e.g., in [Baader & Sattler 2000].

Apart from enforcing that Rfe is applied with highest priority, we do not specify the order of rule application inside the fcompl function. This (seemingly) introduces additional nondeterminism into the decisision procedure. It is, however, a different form of nondeterminism than the one introduced by the R$\sqcup$ rule: nondeterminism as in the fcompl function is called *don't care nondeterminism* since the nondeterministic choice has no influence on the result of the algorithm; to the contrary, nondeterminsm as in the R$\sqcup$ rule is called *don't know nondeterminism* since the nondeterministic choice has an impact on the result of the algorithm. Obviously, it is trivial to convert an algorithm with don't care nondeterminism only into a deterministic one. The difference the algorithms may lie in their efficiency, but not in the result.

R∃r	if $\exists R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ and there is no R-successor b of a with $C \in \mathcal{A}(b)$ then set $\mathcal{A} := (\mathcal{A} + aRb) \cup \{b : C\}$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R∀r	if $\forall R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, b is a R-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$

Figure 3.5: Virtual completion rules for $\mathcal{ALCF}(\mathcal{D})$.

3.1.3 Correctness and Complexity

In this section, we prove that the completion algorithm is sound, complete, and terminating and can be executed using only polynomial space provided that $\mathcal{D}$-satisfiability is in PSpace. By D, we denote the input concept to the completion algorithm whose satisfiability is to be decided.

We first prove termination of the algorithm. It is convenient to start with establishing an upper bound for the number of rule applications performed by the fcompl function and, closely related, an upper bound for the size of ABoxes generated by the fcompl function. Before we do this, let us introduce the two additional completion rules displayed in Figure 3.5, which will play an important role in the termination and correctness proofs. These rules are not applied explicitly by the algorithm, but, as we shall see later, recursion calls of the sat function can be viewed as a single application of the R∃r rule together with multiple applications of the R∀r rule. Let us now return to the upper bounds for the fcompl function. With foresight to the ABox consistency algorithm to be devised in the next section, we consider the precompl function instead of the fcompl function, where precompl is defined exactly as fcompl except that it also applies the R∀r rule. A formal definition of the precompl function can be found in Figure 3.7. It is not hard to see that upper bounds for the number of rule applications performed by precompl or the size of ABoxes generated by precompl also apply to the fcompl function.

Lemma 3.5. *For any input $\mathcal{A}$, the function* precompl *terminates after at most $|\mathcal{A}|^3$ rule applications and constructs an ABox $\mathcal{A}'$ with $|\mathcal{A}'| \leq |\mathcal{A}|^5$.*

Proof. In the following, we call assertions of the form $a : C$ *concept assertions*, assertions of the form $(a, b) : f$ or $(a, x) : g$ *feature assertions*, and assertions of the form $(a, b) : R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ *role assertions.*

The main task is to show that

$$\text{precompl terminates after at most } |\mathcal{A}|^3 \text{ rule applications.} \qquad (*)$$

For suppose that $(*)$ has been shown. We can then prove the lemma as follows. We have $|\alpha| < |\mathcal{A}|$ for each new assertion α added by rule application since

(i) concept assertions are the only kind of assertions that may have a size greater one and

(ii) if a concept assertion $a : C$ is added by rule application, then $C \in \mathsf{sub}(\mathcal{A})$.

Moreover, as is easily checked, each rule application adds either no new assertions (the Rfe rule) or at most $|C|$ new assertions, where $a : C$ is the concept assertion appearing in the (instantiated) rule premise.[1] Hence, by (ii), each rule application adds at most $|\mathcal{A}|$ new assertions. Together with $(*)$, these observations imply that the size of the ABox $\mathcal{A}'$ generated by precompl is bounded by $|\mathcal{A}|^5$.

Hence, let us prove $(*)$. Let $\mathcal{A}_0, \mathcal{A}_1, \ldots$ be the sequence of ABoxes computed by precompl. More precisely, $\mathcal{A}_0 = \mathcal{A}$ and $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by the i-th rule application performed by precompl.

We first introduce some notions. For $i \geq 0$ and $a \in \mathsf{O_a} \cup \mathsf{O_c}$, we use $\mathsf{nm}_i(a)$ to denote the set of names that a had "until $\mathcal{A}_i$". More precisely, $\mathsf{nm}_0(a) = \{a\}$ for all $a \in \mathsf{O_a}$. If the Rfe rule is applied to an ABox $\mathcal{A}_i$ renaming an object a to b, then $\mathsf{nm}_{i+1}(b) = \mathsf{nm}_i(a) \cup \mathsf{nm}_i(b)$ and $\mathsf{nm}_{i+1}(c) = \mathsf{nm}_i(c)$ for all $c \neq b$. For all other rule applications, we simply have $\mathsf{nm}_{i+1}(a) = \mathsf{nm}_i(a)$ for all $a \in \mathsf{O_a} \cup \mathsf{O_c}$. The following properties, which we summarize under the notion *persistence*, are easily proved using the fact that assertions are never deleted:

- If $a : C \in \mathcal{A}_i$ and $a \in \mathsf{nm}_j(a')$ for some $j > i$ and $a' \in \mathsf{O_a}$, then $a' : C \in \mathcal{A}_j$.
- if $(a, b) : R \in \mathcal{A}_i$, $a \in \mathsf{nm}_j(a')$, and $b \in \mathsf{nm}_j(b')$ for some $j > i$ and $a', b' \in \mathsf{O_a}$, then $(a', b') : R \in \mathcal{A}_j$.
- If $(a, x) : g \in \mathcal{A}_i$, $a \in \mathsf{nm}_j(a')$, and $x' \in \mathsf{nm}_j(x)$ for some $j > i$, $a' \in \mathsf{O_a}$, and $x' \in \mathsf{O_c}$, then $(a', x') : g \in \mathcal{A}_j$.
- If $(x_1, \ldots, x_n) : P \in \mathcal{A}_i$ and $x_i' \in \mathsf{nm}_j(x_i)$ for $1 \leq i \leq n$, then $(x_1', \ldots, x_n') : P \in \mathcal{A}_j$.

A concept assertion $a : C$ is called *touched* in $\mathcal{A}_i$ if there exists an $a' \in \mathsf{nm}_i(a)$ such that one of the first i rule applications involved $a' : C$ in the (instantiated) rule premise and *untouched* otherwise. By $\sharp_{\mathsf{feat}}(\mathcal{A})$, we denote the number of feature assertions in $\mathcal{A}$. For role assertions $(a, b) : R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, we use $\lambda_{\mathcal{A}_i}(a, b : R)$ to denote the number of concepts $\forall R.C$ in $\mathsf{sub}(\mathcal{A})$ for which there exist no $a' \in \mathsf{nm}_i(a)$ and $b' \in \mathsf{nm}_i(b)$ such that one of the first i rule applications involved both $a' : \forall R.C$ and $(a', b') : R$ in the (instantiated) rule premise.

For $i \geq 0$, define

$$w(\mathcal{A}_i) := \sum_{a:C \text{ is untouched in } \mathcal{A}_i} |a : C| \quad + \quad \sharp_{\mathsf{feat}}(\mathcal{A}_i) \quad + \sum_{(a,b):R \in \mathcal{A}_i} \lambda_{\mathcal{A}_i}(a, b : R) \cdot |\mathcal{A}|.$$

We show that $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$ for $i \geq 0$, which implies that the length of the sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots$ is bounded by $|\mathcal{A}|^3$ since, as is easily checked, $w(\mathcal{A}_0) \leq |\mathcal{A}|^3$. A case distinction is made according to the completion rule applied.

- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the R$\sqcap$ rule. By definition of this rule and due to persistence, it is applied to an untouched

[1] Recall that the length of a concept is the number of symbols used to write it, see Section 2.1.1.

assertion $a : C_1 \sqcap C_2$ in $\mathcal{A}_i$: for suppose that $a : C_1 \sqcap C_2$ is touched in $\mathcal{A}_i$. By definition of "touched", this implies that there exists an $a' \in \mathsf{nm}_i(a)$ such that $\mathsf{R}\sqcap$ has been applied to $a' : C_1 \sqcap C_2$ in the j-th rule application for some $j < i$. By definition of $\mathsf{R}\sqcap$, this implies $\{a' : C_1, a' : C_2\} \subseteq \mathcal{A}_j$. By persistence, we have $\{a : C_1, a : C_2\} \subseteq \mathcal{A}_i$ and, thus, the $\mathsf{R}\sqcap$ rule is not applicable to $a : C_1 \sqcap C_2$ in $\mathcal{A}_i$ which is a contradiction. Hence, we have shown that $a : C_1 \sqcap C_2$ is untouched in $\mathcal{A}_i$. Moreover, this assertion is clearly touched in $\mathcal{A}_{i+1}$. The rule application generates new concept assertions $a : C_1$ and $a : C_2$ which are untouched in $\mathcal{A}_{i+1}$ and it generates no new feature and role assertions. By definition of the size of assertions and the length of concepts, we have $|a : C_1 \sqcap C_2| > |a : C_1| + |a : C_2|$.

- The $\mathsf{R}\sqcup$ case is analogous to the previous case.

- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R\forall f}$ rule. The rule is applied to assertions $a : \forall f.C$ and $(a, b) : f$. Suppose that $a : \forall f.C$ is touched in $\mathcal{A}_i$, i.e., that the $\mathsf{R\forall f}$ rule has been applied in a previous step to an assertion $a' : \forall f.C$ with $a' \in \mathsf{nm}_i(a)$. It then added $c : C$ for an f-successor c of a'. The facts that (i) Rfe is applied with highest priority, (ii) b is an f-successor of a in $\mathcal{A}_{i+1}$, and (iii) the $\mathsf{R\forall f}$ rule is applicable imply that we have $c \in \mathsf{nm}_i(b)$. This, in turn, implies $b : C \in \mathcal{A}_i$ by persistence and we have obtained a contradiction to the assumption that $\mathsf{R\forall f}$ is applicable. Hence, we have shown that $a : \forall f.C$ is untouched in $\mathcal{A}_i$. The assertion is touched in $\mathcal{A}_{i+1}$. Rule application generates a new assertion $b : C$ that is untouched in $\mathcal{A}_{i+1}$. However, $|a : \forall f.C| > |b : C|$. No new feature or role assertions are generated.

- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R\forall r}$ rule. The rule is applied to assertions $a : \forall R.C$ and $(a, b) : R$ in $\mathcal{A}_i$. Due to persistence, there do not exist $a' \in \mathsf{nm}_i(a)$ and $b' \in \mathsf{nm}_i(b)$ such that the $\mathsf{R\forall r}$ rule has previously been applied to $a' : \forall R.C$ and $(a', b') : R$. Hence, $\lambda_{\mathcal{A}_{i+1}}(a, b : R) = \lambda_{\mathcal{A}_i}(a, b : R) - 1$ and the third summand of $w(\mathcal{A}_i)$ exceeds the third summand of $w(\mathcal{A}_{i+1})$ by $|\mathcal{A}|$. The rule application adds no feature or role assertions and a single concept assertion $b : C$. Since $\forall R.C \in \mathsf{sub}(\mathcal{A})$, we have $|b : C| < |\mathcal{A}|$ and hence $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$.

- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R\exists f}$ rule. As in the $\mathsf{R}\sqcap$ case, it is easy to show that the rule is applied to an untouched assertion $a : \exists f.C$. It generates new assertions $(a, b) : f$ and $b : C$ (and no new role assertions). The assertion $b : C$ is untouched in $\mathcal{A}_{i+1}$ and $a : \exists f.C$ is touched in $\mathcal{A}_{i+1}$. The new feature assertion $(a, b) : f$ yields $\sharp_{\mathsf{feat}}(\mathcal{A}_{i+1}) = \sharp_{\mathsf{feat}}(\mathcal{A}_i) + 1$. On the other hand, no role assertion is added and we clearly have $|a : \exists f.C| > |b : C| + 1$.

- The Rc, $\mathsf{R}\!\downarrow$, and $\mathsf{R}\!\uparrow$ rules touch a (due to persistence) previously untouched concept assertion $a : C$ appearing in the instantiated premise and do not add new concept or role assertions. It is readily checked that the number of feature assertions added by rule application smaller than $|a : C|$.

- Assume that the Rfe rule is applied to an ABox $\mathcal{A}_i$. This obviously implies $\sharp_{\mathsf{feat}}(\mathcal{A}_{i+1}) < \sharp_{\mathsf{feat}}(\mathcal{A}_i)$, i.e., the second summand of $w(\mathcal{A}_{i+1})$ is strictly smaller than the second summand of $w(\mathcal{A}_i)$. If the rule application renames a concrete object, these are the only changes and we are done. If an abstract object is renamed, some work is necessary to show that the first and third summand of $w(\mathcal{A}_{i+1})$ are not greater than the corresponding summands of $w(\mathcal{A}_i)$. Assume that $a \in \mathsf{O_a}$ is renamed to b. We then have $\mathsf{nm}_{i+1}(b) = \mathsf{nm}_i(a) \cup \mathsf{nm}_i(b)$.

- First summand. Every concept assertion $c : C \in \mathcal{A}_{i+1} \cap \mathcal{A}_i$ is touched in $\mathcal{A}_{i+1}$ iff it is touched in $\mathcal{A}_i$: this is trivial if $c \neq b$ since this implies $\mathsf{nm}_{i+1}(c) = \mathsf{nm}_i(c)$. If $c = b$, it follows from $\mathsf{nm}_i(b) \subseteq \mathsf{nm}_{i+1}(b)$. Moreover, if there exists an assertion $b : C \in \mathcal{A}_{i+1} \setminus \mathcal{A}_i$ due to variable renaming, then $a : C \in \mathcal{A}_i \setminus \mathcal{A}_{i+1}$, and $b : C$ being untouched in $\mathcal{A}_{i+1}$ implies $a : C$ being untouched in $\mathcal{A}_i$ since $\mathsf{nm}_i(a) \subseteq \mathsf{nm}_{i+1}(b)$. Hence, the first summand does not increase.
- Third summand. Let $(c, d) : R \in \mathcal{A}_{i+1} \cap \mathcal{A}_i$ (implying $c \neq a$ and $d \neq a$). We distinguish several subcases:
  1. $c \neq b$ and $d \neq b$. Then, clearly, $\lambda_i(c, d : R) = \lambda_{i+1}(c, d : R)$.
  2. $c = b$ and $d \neq b$. By definition of λ_i, $\mathsf{nm}_i(b) \subseteq \mathsf{nm}_{i+1}(b)$ implies $\lambda_i(b, d : R) \geq \lambda_{i+1}(b, d : R)$.
  3. $c \neq b$ and $d = b$. As previous case.
  4. $c = e = b$. As previous case.

  Now let $(c, d) : R \in \mathcal{A}_{i+1} \setminus \mathcal{A}_i$ (implying $c = b$ or $d = b$). We can distinguish the cases (i) $c = b$, $d \neq b$, (ii) $d = b$, $c \neq b$, and (iii) $c = d = b$. Since all cases are similar, we concentrate on (i). In this case, $(a, d) : R \in \mathcal{A}_i \setminus \mathcal{A}_{i+1}$. Moreover, $\mathsf{nm}_i(a) \subseteq \mathsf{nm}_{i+1}(b)$ implies $\lambda_{\mathcal{A}_{i+1}}(b, d : R) \leq \lambda_{\mathcal{A}_i}(a, d : R)$.

  Summing up, the third summand may only decrease but not increase. ❑

The role depth of $\mathcal{ALCF}(\mathcal{D})$-concepts is defined analogously to the role depth of $\mathcal{ALC}$-concepts (see Section 2.1.1) with the only difference that, in the case of $\mathcal{ALCF}(\mathcal{D})$, both roles (including abstract features) and concrete features contribute to the role depth. For example, the role depth of $\forall R.\exists fg.\top_{\mathcal{D}}$ is 3. We now prove a technical lemma that, together with Lemma 3.5, immediately yields termination.

Lemma 3.6. *Assume that the completion algorithm was started with input D. Then*

1. *in each recursion call, the size $|\mathcal{A}|$ of the argument $\mathcal{A}$ passed to* sat *is bounded by $|D|^2$;*
2. *in each recursion step of* sat*, at most $p(|D|)$ recursion calls are made, where p is a polynomial; and*
3. *the recursion depth of* sat *is bounded by $|D|$.*

Proof. Let us first prove Point 1. ABoxes passed to sat contain assertions of the form $a : C$ for a single object a. Since only concepts from $\mathsf{sub}(D)$ are generated during rule application, the number of distinct assertions of this form is bounded by $|\mathsf{sub}(D)| \leq |D|$. Obviously, the size of each such assertion is also bounded by $|D|$ which yields an upper bound of $|D|^2$ for the size of arguments to sat.

For Point 2, note that in each recursion step, the number of recursion calls made is bounded by the number of assertions $a : \exists R.C$ in the ABox $\mathcal{A}$ obtained by application of fcompl. By Point 1, the size of argument ABoxes to sat is bounded by $|D|^2$. Hence, by Lemma 3.5, the size of $\mathcal{A}$ is bounded by $p(|D|)$ where p is a polynomial and the same bound applies to the number of recursion calls made in each recursion step.

Let us turn to Point 3. As a consequence of (i) the fact that rule application performed by fcompl may not introduce concepts with a role depth greater than the role depth of concepts that have already been in the ABox and (ii) the way in which the argument ABoxes for recursion calls to sat are constructed, we have that the role depth of concepts in the argument ABoxes passed to sat strictly decreases with recursion depth. It follows that the role depth of D is an upper bound for the recursion depth, i.e., the recursion depth is bounded by $|D|$. ❑

Proposition 3.7. *The completion algorithm terminates on any input $\mathcal{A}_D$.*

Proof. Immediate consequence of Lemma 3.5 and Points 2 and 3 from Lemma 3.6. ❑

We now come to proving soundness and completeness of the completion algorithm. Recall that, intuitively, the completion algorithm traverses a generalized tree model in a depth-first manner without keeping the entire model in memory. For the proofs, it is convenient to make the model traversed by the algorithm explicit—or more precisely the ABox representing it. To do this, we define an extended version of the completion algorithm. This extended algorithm is identical to the original one but additionally constructs a sequence of ABoxes $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$ collecting all assertions that the algorithm generates. Hence, it returns satisfiable if and only if the original algorithm does. We will show that, if the extended algorithm is started on an initial ABox $\mathcal{A}_D$ and terminates after n steps returning satisfiable, then the ABox $\mathcal{A}^n_\cup$ defines a canonical model for $\mathcal{A}_D$. Since the extended algorithm returns satisfiable if the original one does, this yields soundness. Completeness can also be shown using the correspondence between the two algorithms. Note that the extended version of the algorithm is defined just to prove soundness and completeness of the original version and we do not claim that the extended version itself can be executed in polynomial space.

The extended algorithm can be found in Figure 3.6. The extensions are marked with asterisks. If the algorithm is started on the initial ABox $\mathcal{A}_D = \{a : D\}$, we set $\mathcal{A}^0_\cup := \{a_0 : D\}$. The algorithm uses two *global* variables sc and rc, which are both initialized with the value 0. The first one is a counter for the number of calls to the sat function. The second one counts the number of ABoxes $\mathcal{A}^i_\cup$ that have already been generated. The introduction of the global variable sc is necessary due to the following technical problem: the object names created by the algorithm are unique only within the ABox considered in a single recursion step. For the cumulating ABoxes $\mathcal{A}^i_\cup$ that collect assertions from many recursion steps, we have to ensure that an object a from one recursion step can be distinguished from a in a different step since these two objects do clearly not represent the same domain element in the constructed model. To achieve this, objects are renamed before new assertions are added to an ABox $\mathcal{A}^i_\cup$ by indexing with the value of the counter sc.

```
*  Initialization:
*  rc := sc := 0
*  A_∪^0 := {a_0 : D} if A_D = {a : D}

   define procedure sat(A)
      A := fcompl(A)
      if A contains a clash then
         return unsatisfiable
      forall objects a in O_a and ∃R.C ∈ A(a) with R ∈ N_R \ N_aF do
*        sc := sc + 1
*        rc := rc + 1
         Fix b ∈ O_a
*        A_∪^rc := A_∪^{rc-1} ∪ {(a_{sc-1}, b_sc) : R} ∪ {b_sc : C} ∪
*                                  {b_sc : E | a : ∀R.E ∈ A(a)}
         if sat({b : C} ∪ {b : E | ∀R.E ∈ A(a)}) = unsatisfiable then
            return unsatisfiable
      return satisfiable

   define procedure fcompl(A)
*     A_0 := A
      while a rule R from Figure 3.2 is applicable to A do
         Choose an applicable rule R s.t. R = Rfe if Rfe is applicable
         Apply R to A
*     rc := rc + 1
*     N := A \ A_0
*     Replace each a ∈ O_a (resp. x ∈ O_c) in N with a_sc (resp. x_sc)
*     A_∪^rc := A_∪^{rc-1} ∪ N
      return A
```

Figure 3.6: The extended satisfiability algorithm.

Observe that, for $i > 0$, the ABox $\mathcal{A}_\cup^i$ is obtained either

1. by multiple applications of completion rules from Figure 3.2 to the ABox $\mathcal{A}_\cup^{i-1}$ or

2. by a recursion call made while the counter rc has value $i - 1$.

Let us be a little bit more precise about the second point. W.r.t. the sequence of ABoxes $\mathcal{A}_\cup^0, \mathcal{A}_\cup^1, \ldots$, recursion calls can be viewed as applications of the completion rules displayed in Figure 3.5: if $\mathcal{A}_\cup^i$ is obtained from $\mathcal{A}_\cup^{i-1}$ by a recursion call, then this is equivalent to a single application of the R∃r rule and exhaustive application of the R∀r rule.

Non-applicability of all completion rules to an ABox will be an important property in what follows.

Definition 3.8 (Complete ABox). An ABox $\mathcal{A}$ is *complete* iff no completion rule from Figures 3.2 and 3.5 is applicable to $\mathcal{A}$. $\diamond$

The following lemma is central for proving soundness and completeness.

Lemma 3.9. *Let $\mathcal{A}$ be an $\mathcal{ALCF}(\mathcal{D})$-ABox and* R *be a completion rule from Figure 3.2 or Figure 3.5 such that* R *is applicable to $\mathcal{A}$.*

1. *$\mathcal{A}$ is consistent iff* R *can be applied such that the resulting ABox $\mathcal{A}'$ is consistent.*

2. *if $\mathcal{A}$ is complete and clash-free, then it is consistent.*

Proof. The two parts are proved separately.

(1) Let us first deal with the "if" direction. This is trivial if $\mathsf{R} \neq \mathsf{Rfe}$ since this implies $\mathcal{A} \subseteq \mathcal{A}'$ and, hence, every model of $\mathcal{A}'$ is also a model of $\mathcal{A}$. Assume that the Rfe rule is applied to assertions $\{(a,b) : f, (a,c) : f\} \in \mathcal{A}$ and replaces c with b. Let $\mathcal{I}$ be a model of $\mathcal{A}'$. Construct an interpretation $\mathcal{I}'$ from $\mathcal{I}$ by setting $c^{\mathcal{I}'} := b^{\mathcal{I}}$. It is straightforward to check that $\mathcal{I}'$ is a model of $\mathcal{A}$. The case that Rfe is applied to assertions $\{(a,x) : g, (a,y) : g\} \in \mathcal{A}$ is analogous.

Now for the "only if" direction. We make a case distinction according to the completion rule R.

- The $\mathsf{R}\sqcap$ rule is applied to an assertion $a : C_1 \sqcap C_2$ and $\mathcal{A}' = \mathcal{A} \cup \{a : C_1, a : C_2\}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (C_1 \sqcap C_2)^{\mathcal{I}}$, we have $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$ and $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$ by the semantics of $\mathcal{ALCF}(\mathcal{D})$, which implies that $\mathcal{I}$ is also a model of $\mathcal{A}'$.

- The $\mathsf{R}\sqcup$ rule is applied to an assertion $a : C_1 \sqcup C_2$. The rule can be applied such that either $\mathcal{A}' = \mathcal{A} \cup \{a : C_1\}$ or $\mathcal{A}' = \mathcal{A} \cup \{a : C_2\}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}}$, we have either $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$ or $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$ by the semantics of $\mathcal{ALCF}(\mathcal{D})$. Hence, we can apply the rule such that $\mathcal{I}$ is a model of $\mathcal{A}'$.

- The $\mathsf{R}\exists\mathsf{f}$ rule is applied to an assertion $a : \exists f.C$ yielding the ABox $\mathcal{A}'$. Then $\mathcal{A}' = \mathcal{A} \cup \{(a,b) : f, b : C\}$ where b is fresh in $\mathcal{A}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\exists f.C)^{\mathcal{I}}$, there exists a $d \in \Delta_{\mathcal{I}}$ such that $f^{\mathcal{I}}(a^{\mathcal{I}}) = d$ and $d \in C^{\mathcal{I}}$. Let $\mathcal{I}'$ be the interpretation obtained from $\mathcal{I}$ by setting $a^{\mathcal{I}'} := d$. It is easily checked that $\mathcal{I}'$ is a model of $\mathcal{A}'$.

- The $\mathsf{R}\exists\mathsf{r}$ rule is treated analogously to the previous case.

- The $\mathsf{R}\forall\mathsf{f}$ rule is applied to an assertion $a : \forall f.C$ and $\mathcal{A}' = \mathcal{A} \cup \{b : C\}$ where b is an f-successor of a in $\mathcal{A}$ and $\mathcal{A}'$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\forall f.C)^{\mathcal{I}}$ and $f^{\mathcal{I}}(a^{\mathcal{I}}) = b^{\mathcal{I}}$, we have $b \in C^{\mathcal{I}}$. Hence, $\mathcal{I}$ is also a model of $\mathcal{A}'$.

- The $\mathsf{R}\forall\mathsf{r}$ rule is treated analogously to the previous case.

- The Rc rule is applied to an assertion $a : \exists u_1, \ldots, u_n.P$ with $u_i = f_1^{(i)} \cdots f_{k_i}^{(i)} g_i$ yielding the ABox $\mathcal{A}'$. Then there exist abstract objects $a_j^{(i)}$ with $1 \leq i \leq n$ and $1 \leq j \leq k_i$ which are fresh in $\mathcal{A}$ and concrete objects $x_1, \ldots, x_n$ which are fresh in $\mathcal{A}$ such that, for $1 \leq i \leq n$,

(i) $a_1^{(i)}$ is $f_1^{(i)}$-successor of a,

(ii) $a_j^{(i)}$ is $f_j^{(i)}$-successor of $a_{j-1}^{(i)}$ for $1 < j \leq k_i$,

(iii) x_i is g_i-successor of $a_{k_i}^{(i)}$, and

(iv) $(x_1, \ldots, x_n) : P \in \mathcal{A}'$.

Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$, there exist domain elements $d_j^{(i)} \in \Delta_{\mathcal{I}}$ with $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and $z_1, \ldots, z_n \in \Delta_{\mathcal{D}}$ such that, for $1 \leq i \leq n$, we have

- $(a^{\mathcal{I}}, d_1^{(i)}) \in (f_1^{(i)})^{\mathcal{I}}$,
- $(d_{j-1}^{(i)}, d_j^{(i)}) \in (f_j^{(i)})^{\mathcal{I}}$ for $1 < j \leq k_i$,
- $g_i^{\mathcal{I}}(d_{k_i}^{(i)}) = z_i$, and
- $(z_1, \ldots, z_n) \in P^{\mathcal{D}}$.

Define $\mathcal{I}'$ as the interpretation obtained from $\mathcal{I}$ by setting

$$(a_j^{(i)})^{\mathcal{I}'} := d_j^{(i)} \text{ for } 1 \leq i \leq n \text{ and } 1 < j \leq k_i$$

and

$$x_i^{\mathcal{I}'} := z_i \text{ for all } i \text{ with } 1 \leq i \leq n.$$

It is straightforward to check that $\mathcal{I}'$ is a model of $\mathcal{A}'$.

- Applications of the R$\downarrow$ rule are treated similar to the previous case.
- Applications of the R$\uparrow$ rule are also treated similar to the Rc case.
- The Rfe rule is applied to assertions $\{(a, b) : f, (a, c) : f\} \in \mathcal{A}$ and replaces c with b. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Due to the presence of the above two assertions and since features are interpreted as partial functions, we have $b^{\mathcal{I}} = c^{\mathcal{I}}$. It is readily checked that this implies that $\mathcal{I}$ is a model of $\mathcal{A}'$.

(2) Based on $\mathcal{A}$, an interpretation $\mathcal{I}$ can be defined as follows. Fix a solution δ for $\zeta_{\mathcal{A}}$ which exists since $\mathcal{A}$ is clash-free.

1. $\Delta_{\mathcal{I}}$ consists of all abstract objects that occur in $\mathcal{A}$,
2. $A^{\mathcal{I}} := \{a \in \mathsf{O_a} \mid a : A \in \mathcal{A}\}$ for all $A \in \mathsf{N_C}$,
3. $R^{\mathcal{I}} := \{(a, b) \in \mathsf{O_a} \times \mathsf{O_a} \mid (a, b) : R \in \mathcal{A}\}$ for all $R \in \mathsf{N_R}$,
4. $g^{\mathcal{I}} := \{(a, \delta(x)) \in \mathsf{O_a} \times \Delta_{\mathcal{D}} \mid (a, x) : g \in \mathcal{A}\}$ for all $g \in \mathsf{N_{cF}}$,
5. $a^{\mathcal{I}} := a$ for all $a \in \mathsf{O_a}$, and
6. $x^{\mathcal{I}} := \delta(x)$ for all $x \in \mathsf{O_c}$.

Note that $\mathcal{I}$ is well-defined: Since the Rfe rule is not applicable, $f^{\mathcal{I}}$ and $g^{\mathcal{I}}$ are functional for all $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. We prove that $\mathcal{I}$ is a model of $\mathcal{A}$, i.e., that all assertions in $\mathcal{A}$ are satisfied by $\mathcal{I}$. It is an immediate consequence of the definition of $\mathcal{I}$ that $(a,b) : R \in \mathcal{A}$ implies $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $(a,x) : g \in \mathcal{A}$ implies $g^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}}$. Moreover, if $(a \neq b) \in \mathcal{A}$, then $a \neq b$ since $\mathcal{A}$ is clash-free. Hence, $(a \neq b) \in \mathcal{A}$ implies $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Since δ is a solution for $\zeta_{\mathcal{A}}$, $(x_1, \ldots, x_n) : P \in \mathcal{A}$ implies $(x_1^{\mathcal{I}}, \ldots, x_n^{\mathcal{I}}) \in P^{\mathcal{D}}$. It thus remains to show that $a : C \in \mathcal{A}$ implies $a \in C^{\mathcal{I}}$. This is done by induction on the structure of C. For the induction start, we make a case distinction according to the form of C:

- If $C \in \mathsf{N_C}$, then the above claim is an immediate consequence of the definition of C.
- $C = \neg E$. Since we assume all concepts to be in negation normal form, E is a concept name. Since $\mathcal{A}$ is clash-free, $a : E \notin \mathcal{A}$ and, by definition of $\mathcal{I}$, $a \notin E^{\mathcal{I}}$. Hence, $a \in (\neg E)^{\mathcal{I}}$.
- $C = \exists u_1, \ldots, u_n.P$. Since the Rc rule is not applicable to $\mathcal{A}$, there exist $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is u_i-successor of a in $\mathcal{A}$ for $1 < i \leq n$. By definition of $\mathcal{I}$, we have $u_i^{\mathcal{I}}(a) = \delta(x_i)$ for $1 < i \leq n$. Furthermore, we have $(x_1, \ldots, x_n) : P \in \mathcal{A}$ and, since δ is a solution for $\zeta_{\mathcal{P}}$, $(\delta(x_1), \ldots, \delta(x_n)) \in P^{\mathcal{D}}$. Summing up, $a \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$.
- $C = p_1{\downarrow}p_2$. Since the $\mathsf{R}{\downarrow}$ rule is not applicable to $\mathcal{A}$, there exists an object $b \in \mathsf{O_a}$ which is both a p_1-successor and a p_2-successor of a in $\mathcal{A}$. By definition of $\mathcal{I}$, we have $p_1^{\mathcal{I}}(a) = p_2^{\mathcal{I}}(a) = b$ and, hence, $a \in (p_1{\downarrow}p_2)^{\mathcal{I}}$.
- $C = p_1{\uparrow}p_2$. Since the $\mathsf{R}{\uparrow}$ rule is not applicable to $\mathcal{A}$, there exist $b_1, b_2 \in \mathsf{O_a}$ such that b_1 is a p_1-successor of a in $\mathcal{A}$, b_2 is a p_2-successor of a in $\mathcal{A}$, and $b_1 \neq b_2 \in \mathcal{A}$. Since $\mathcal{A}$ is clash-free, we have $b_1 \neq b_2$. By definition of $\mathcal{I}$, we have $p_1^{\mathcal{I}}(a) = b_1$ and $p_2^{\mathcal{I}}(a) = b_2$ and, hence, $a \in (p_1{\uparrow}p_2)^{\mathcal{I}}$.
- $C = g{\uparrow}$. Since $\mathcal{A}$ is clash-free, a has no g-successor x in $\mathcal{A}$. By definition of $\mathcal{I}$, $g^{\mathcal{I}}(a)$ is undefined and hence $a \in (g{\uparrow})^{\mathcal{I}}$.

For the induction step, we make a case analysis according to the topmost constructor in C.

- $C = C_1 \sqcap C_2$. Since the $\mathsf{R}\sqcap$ rule is not applicable to $\mathcal{A}$, we have $\{C_1, C_2\} \subseteq \mathcal{A}(a)$. By induction, $a \in C_1^{\mathcal{I}}$ and $a \in C_2^{\mathcal{I}}$, which implies $a \in (C_1 \sqcap C_2)^{\mathcal{I}}$.
- $C = C_1 \sqcup C_2$. Similar to the previous case.
- $C = \exists R.E$. Since neither the $\mathsf{R}\exists\mathsf{f}$ nor the $\mathsf{R}\exists\mathsf{r}$ rule is applicable to $\mathcal{A}$, there exists an object $b \in \mathsf{O_a}$ such that b is an R-successor of a in $\mathcal{A}$ and $E \in \mathcal{A}(b)$. By definition of $\mathcal{I}$, b being an R-successor of a implies $(a,b) \in R^{\mathcal{I}}$. By induction, we have $b \in E^{\mathcal{I}}$ and may hence conclude $a \in (\exists R.E)^{\mathcal{I}}$.

- $C = \forall R.E$. Let $b \in \Delta_\mathcal{I}$ such that $(a,b) \in R^\mathcal{I}$. By definition of $\mathcal{I}$, b is an R-successor of a in $\mathcal{A}$. Since neither the R∀f not the R∀r rule is applicable to $\mathcal{A}$, we have $E \in \mathcal{A}(b)$. By induction, it follows that $b \in E^\mathcal{I}$. Since this holds for all b, we can conclude $a \in (\forall R.E)^\mathcal{I}$. ❑

In the following, the *i-th recursion step* denotes the recursion step of the extended completion algorithm in which the counter sc has value i.

Proposition 3.10 (Soundness). *If the completion algorithm returns* satisfiable, *then the input concept is satisfiable.*

Proof. Assume that the completion algorithm is started on an input concept D and there exists a way to make the non-deterministic decisions such that the algorithm returns satisfiable. Moreover assume that the extended algorithm constructs the ABox $\mathcal{A}^n_\cup$ if the non-deterministic decisions are made in precisely the same way, i.e., the counter rc has value n upon termination. We first establish the following claim:

Claim: $\mathcal{A}^n_\cup$ is complete and clash-free.

First for completeness. We distinguish several cases. First assume that a rule

$$\mathsf{R} \in \{\mathsf{R}\sqcap, \mathsf{R}\sqcup, \mathsf{R}\exists\mathsf{f}, \mathsf{Rc}, \mathsf{R}\!\downarrow, \mathsf{R}\!\uparrow, \mathsf{R}\exists\mathsf{r}\}$$

is applicable to $\mathcal{A}^n_\cup$. This is due to the presence of an assertion $a_i : C$ in $\mathcal{A}^n_\cup$. If, e.g., $\mathsf{R} = \mathsf{R}\sqcap$, then C has the form $C_1 \sqcap C_2$. By construction of $\mathcal{A}^n_\cup$, this implies that $a : C$ is in $\mathcal{A}$ in the i-th recursion step. Hence, if $\mathsf{R} \neq \mathsf{R}\exists\mathsf{r}$, the rule R has been applied to $a : C$ by the fcompl function which, again by construction of $\mathcal{A}^n_\cup$, implies that R is not applicable to $a_i : C$ in $\mathcal{A}^n_\cup$ yielding a contradiction. If $\mathsf{R} = \mathsf{R}\exists\mathsf{r}$, then $C = \exists R.E$. Clearly, $(a_i, b_j) : R$ and $b_j : C$ (for some $j > i$) is added to $\mathcal{A}^n_\cup$ due to a subsequent recursion call and we obtain a contradiction to the applicability of R∃r to $a_i : C$ in $\mathcal{A}^n_\cup$.

Now assume that the R∀f rule is applicable to $\mathcal{A}^n_\cup$. This is due to the presence of assertions $a_i : \forall f.C$ and $(a_i, b_j) : f$ in $\mathcal{A}^n_\cup$. Since assertions $(a_i, b_j) : f$ are only added to $\mathcal{A}^n_\cup$ because of applications of the rules R∃f, Rc, R↓, and R↑ performed by the fcompl function, we have $i = j$. It follows that $a : \forall f.C$ and $(a, b) : f$ are in $\mathcal{A}$ in the i-th recursion step. Hence, the R∀f rule is applied by fcompl to these assertions. This implies that $b : C$ is in $\mathcal{A}$ in the i-th recursion step which allows us to conclude $b_i : C \in \mathcal{A}^n_\cup$, a contradiction.

Assume that R∀r is applicable to $\mathcal{A}^n_\cup$ due to the presence of assertions $a_i : \forall R.C$ and $(a_i, b_j) : R$. By construction of $\mathcal{A}^n_\cup$, $a_i : \forall R.C$ in is $\mathcal{A}$ in the i-th recursion step and $(a_i, b_j) : R$ has been added to $\mathcal{A}^n_\cup$ due to a recursion call made during the i-th recursion step. By definition of the annotated algorithm, these two facts imply that $b_j : C$ has also been added to $\mathcal{A}^n_\cup$ in the i-th recursion step. Again a contradiction.

To finish the proof that $\mathcal{A}^n_\cup$ is complete, assume that Rfe is applicable to $\mathcal{A}^n_\cup$ due to the presence of assertions $(a_i, b_j) : f$ and $(a_i, c_\ell) : f$. Since assertions $(a_i, b_j) : f$ are only added to $\mathcal{A}^n_\cup$ because of applications of the rules R∃f, Rc, R↓, and R↑ performed by the fcompl function, we have $i = j = \ell$. It follows that $(a, b) : f$ and $(a, c) : f$ are in $\mathcal{A}$ in the i-th recursion step. Hence, the Rfe rule is applied by fcompl. This, however, implies that either $(a_i, b_j) : f$ or $(a_i, c_\ell) : f$ is not in $\mathcal{A}^n_\cup$.

We now prove that $\mathcal{A}$ is clash-free. Assume $\{A, \neg A\} \subseteq \mathcal{A}^n_\cup(a_i)$. Then $\{A, \neg A\} \subseteq \mathcal{A}(a)$ in the i-th recursion step. Since $\mathcal{A}$ is clash-free in every recursion step (the algorithm returned satisfiable), we obtain a contradiction. Clashes of the form $a_i \neq a_i \in \mathcal{A}^n_\cup$ are treated analogously. Now assume $a_i : g\uparrow$ and $(a_i, x_j) : g$ are in $\mathcal{A}^n_\cup$. Since assertions $(a_i, x_j) : g$ are only added due to applications of the Rc rule by fcompl, we have $i = j$. It is again straightforward to derive a contradiction.

It remains to show that $\mathcal{A}^n_\cup$ is concrete domain satisfiable. For every $i \leq n$, let $\mathcal{A}_i$ be the ABox $\mathcal{A}$ in the i-th recursion step after the application of fcompl and let δ_i be a solution for $\zeta_{\mathcal{A}_i}$, which exists since $\mathcal{A}_i$ is clash-free. Define $\delta(x_i) := \delta_i(x)$ for all x_i occurring in $\mathcal{A}^n_\cup$. It is readily checked that δ is a solution for $\zeta_{\mathcal{A}^n_\cup}$: fix an assertion $((x_1)_{i_1}, \ldots, (x_k)_{i_k}) : P \in \mathcal{A}^n_\cup$. Since such assertions are only added due to applications of the Rc rule by fcompl, there exists an $i \leq n$ such that $i_j = i$ for all j with $1 \leq j \leq k$. Hence, $(x_1, \ldots, x_k) : P \in \mathcal{A}_i$ and $(\delta_i(x_1), \ldots, \delta_i(x_k)) \in P^{\mathcal{D}}$. By definition of δ, it follows that $(\delta((x_1)_{i_1}), \ldots, \delta((x_k)_{i_k})) \in P^{\mathcal{D}}$, as was to be shown.

The proof of the claim is now finished and we return to the proof of soundness. By Lemma 3.9, Point 2, the claim implies that $\mathcal{A}^n_\cup$ is consistent. By construction, we have $a_0 : D \in \mathcal{A}^n_\cup$. It immediately follows that D is satisfiable. ❑

Proposition 3.11 (Completeness). *If the completion algorithm is started on a satisfiable input concept, then it returns* satisfiable.

Proof. Let the extended completion algorithm be started on an input concept D that is satisfiable. Then, the initial ABox $\mathcal{A}_D = \{a : D\}$ is obviously consistent. By Lemma 3.9, Point 1 and due to the fact that performing a recursion step corresponds to the application of rules from Figure 3.5, we can make the non-deterministic decisions of the extended algorithm such that every ABox in the sequence $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$ is consistent.

Now assume that the extended algorithm returns unsatisfiable. This means that an ABox $\mathcal{A}$ is encountered that contains a clash. Let, upon this event, the counter rc have the value n. By definition of the extended algorithm, we clearly have $\mathcal{A} \subseteq \mathcal{A}^n_\cup$ up to variable renaming, and, thus, $\mathcal{A}^n_\cup$ also contains a clash. Since it is straightforward to check that an ABox containing a clash is inconsistent, we obtain a contradiction to the consistency of $\mathcal{A}^n_\cup$. Thus, the extended algorithm does not return unsatisfiable. By Proposition 3.7, this implies that it returns satisfiable. It remains to note that the original algorithm returns satisfiable iff the extended algorithm returns satisfiable.

❑

It may be viewed as a byproduct of the soundness and completeness proof that $\mathcal{ALCF}(\mathcal{D})$ has the generalized tree model property defined in Section 3.1.1: assume that the extended algorithm is started with initial ABox $\mathcal{A}_D = \{a : D\}$ and that D is satisfiable. By Proposition 3.11 and the correspondence of the original and the extended algorithm, the extended algorithm returns satisfiable. From the proof of Proposition 3.10, we learn that in this case the ABox $\mathcal{A}^n_\cup$ (where n is the value of the counter sc upon termination) is complete and clash-free. In the proof of Lemma 3.9 Point 2, a canonical model $\mathcal{I}$ of $\mathcal{A}^n_\cup$ is constructed where $\Delta_\mathcal{I}$ is the set of abstract objects used in $\mathcal{A}^n_\cup$. It is straightforward to check that this model is a generalized tree model for D since

1. $a_0 : D$ is in $\mathcal{A}^n_\cup$,

2. the sets $X_i := \{a_i \mid a_i \in \Delta_\mathcal{I}\}$ for $0 \leq i \leq n$ are equivalence classes w.r.t. $\mathcal{I}$ and $\sim$ as in Definition 3.1, and

3. due to the recursive nature of the completion algorithm, the graph $(V_\mathcal{I}, E_\mathcal{I})$ (see Definition 3.1) is a tree.

We now analyze the time and space requirements of our algorithm.

Proposition 3.12.

1. *If $\mathcal{D}$-satisfiability is in* PSPACE, *then the completion algorithm can be executed in polynomial space.*

2. *If $\mathcal{D}$-satisfiability is in* NEXPTIME, *then the completion algorithm can be executed in nondeterministic exponential time.*

3. *If $\mathcal{D}$-satisfiability is in* EXPSPACE, *then the completion algorithm can be executed in exponential space.*

Proof. By Point 1 of Lemma 3.6 and Lemma 3.5, the maximum size of ABoxes $\mathcal{A}$ encountered in recursion steps is bounded by $p(|D|)$, where p is a polynomial. Since, by Point 3 of Lemma 3.6, the recursion depth is bounded by $|D|$, sat can be executed in polynomial space if the check for concrete domain satisfiability is not taken into account.

Assume that $\mathcal{D}$-satisfiability is in PSPACE. Since the maximum size of ABoxes $\mathcal{A}$ encountered in recursion steps is bounded by $p(|D|)$, the maximum number of conjuncts in predicate conjunctions $\zeta_\mathcal{A}$ checked for concrete domain satisfiability is also bounded by $p(|D|)$. Together with the fact that the complexity class PSPACE is oblivious for polynomial blowups of the input, it follows that the completion algorithm can be executed in polynomial space. Along the same lines, it can be shown that the algorithm can be executed in exponential space if $\mathcal{D}$-satisfiability is in EXPSPACE.

Now assume that $\mathcal{D}$-satisfiability is in NEXPTIME. From Lemma 3.5, we know that fcompl terminates after at most $|\mathcal{A}|^3$ rule applications if started on input $\mathcal{A}$. Since, by Point 1 of Lemma 3.6, the size of its input is bounded by $|D|^2$, it terminates after at most $|D|^6$ rule applications. Since the recursion depth is bounded by $|D|$, and, by Point 2 of Lemma 3.6, at most $q(|D|)$ recursion calls are made per recursion step for some polynomial q, sat can be executed in nondeterministic exponential time if the check for concrete domain satisfiability is not taken into account. By the bounds on the recursion depth and the number of recursion calls per recursion steps, the number of concrete domain satisfiability checks performed is at most exponential in $|D|$. Since the size of predicate conjunctions passed in each step is bounded by $p(D)$ and $\mathcal{D}$-satisfiability is in NEXPTIME, we can perform each check in (non-deterministic) time exponential in $|D|$. Summing up, the sat algorithm an be executed in nondeterministic exponential time. ❑

Combining this result with the PSPACE lower bound of $\mathcal{ALC}$-concept satisfiability [Schmidt-Schauß & Smolka 1991] and using Savitch's Theorem which implies that PSPACE = NPSPACE and EXPSPACE = NEXPSPACE [Savitch 1970], we obtain the following theorem.

Theorem 3.13. *Let $\mathcal{D}$ be an admissible concrete domain.*

1. *If $\mathcal{D}$-satisfiability is in* PSPACE, *then $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability is* PSPACE-*complete.*

2. *If $\mathcal{D}$-satisfiability is in* $\mathrm{C} \in \{\text{NExpTime}, \text{ExpSpace}\}$, *then $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability is also in* C.

Since lower complexity bounds transfer from $\mathcal{D}$-satisfiability to $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability, this theorem yields tight complexity bounds if $\mathcal{D}$-satisfiability is NEXPTIME-complete or EXPSPACE-complete. Moreover, since subsumption can be reduced to (un)satisfiability and vice versa (see Section 2.1.1), the obtained complexity bounds also apply to subsumption.[2]

3.2 ABox Consistency

We extend the complexity results obtained in the previous section from concept satisfiability to ABox consistency by devising a *precompletion algorithm* in the style of [Hollunder 1996]. In particular, this algorithm yields a tight PSPACE complexity bound for $\mathcal{ALCF}(\mathcal{D})$-ABox consistency if $\mathcal{D}$-satisfiability is in PSPACE.

3.2.1 The Algorithm

The algorithm works by reducing ABox consistency to concept satisfiability. First, a set of precompletion rules is exhaustively applied to the input ABox $\mathcal{A}$ yielding a *precompletion* of $\mathcal{A}$. Intuitively, rule application makes all implicit knowledge in the ABox explicit except that it does *not* generate new R-successors for roles $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. Then, several *reduction concepts* are generated from the precompletion and passed to the concept satisfiability algorithm devised in the previous section. The input ABox is satisfiable iff the precompletion contains no obvious contradiction and all reduction concepts are satisfiable.

The precise formulation of the algorithm can be found in Figure 3.7. We assume all concepts in the input ABox to be in NNF. As already mentioned in Section 3.1.3, the $\mathsf{precompl}$ function is identical to the fcompl function in Figure 3.3 except that it additionally applies the $\mathsf{R\forall r}$ rule. This is necessary since, in contrast to ABoxes processed by the sat algorithm, the input ABox to cons may contain assertions of the form $(a, b) : R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. Although not generating new R-successors for roles $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, the precompletion algorithm *does* generate new f-successors and new g-successors for features $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. Intuitively, the input ABox induces a

[2] More precisely, this holds for all cases except one: if $\mathcal{D}$-satisfiability is in NEXPTIME, then $\mathcal{ALCF}(\mathcal{D})$-concept subsumption is obviously in co-NEXPTIME.

define procedure $\mathsf{cons}(\mathcal{A})$
  $\mathcal{A} := \mathsf{precompl}(\mathcal{A})$
  if $\mathcal{A}$ contains a clash **then**
    return inconsistent
  forall objects a in $\mathsf{O_a}$ and $\exists R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ **do**
    Fix $b \in \mathsf{O_a}$
    if $\mathsf{sat}(\{b : C \sqcap \bigsqcap_{\forall R.E \in \mathcal{A}(a)} b : E) = \mathsf{unsatisfiable}$ **then**
      return inconsistent
  return consistent

define procedure $\mathsf{precompl}(\mathcal{A})$
  while a rule from $\{\mathsf{R}\sqcap, \mathsf{R}\sqcup, \mathsf{R}\forall\mathsf{r}, \mathsf{R}\forall\mathsf{f}, \mathsf{R}\exists\mathsf{f}, \mathsf{Rc}, \mathsf{R}\downarrow, \mathsf{R}\uparrow, \mathsf{Rfe}\}$
      is applicable to $\mathcal{A}$ **do**
    Choose an applicable rule R s.t. $\mathsf{R} = \mathsf{Rfe}$ if Rfe is applicable
    Apply R to $\mathcal{A}$
  return $\mathcal{A}$

Figure 3.7: The $\mathcal{ALCF}(\mathcal{D})$-ABox consistency algorithm.

set of clusters of objects as discussed in Section 3.1.1 and this cluster is constructed by the precompl function.

Note that the construction of a reduction concept corresponds to a single application of the $\mathsf{R}\exists\mathsf{r}$ rule together with exhaustive application of the $\mathsf{R}\forall\mathsf{r}$ rule very similar to recursion calls of the sat functions in Figure 3.3.

3.2.2 Correctness and Complexity

Termination of the precompletion algorithm is easily obtained.

Proposition 3.14. *The precompletion algorithm terminates on any input.*

Proof. By Lemma 3.5, the precompl function terminates, and, by Proposition 3.7, the sat function also terminates. ❑

We now prove soundness and completeness. In the following, the ABox $\mathcal{A}'$ is called a *precompletion* of the ABox $\mathcal{A}$ iff $\mathcal{A}'$ can be obtained by applying the precompl function to $\mathcal{A}$. Note that precompl is non-deterministic and hence there may exist more than a single precompletion for a given ABox $\mathcal{A}$.

Proposition 3.15 (Soundness). *If the precompletion algorithm returns* consistent*, then the input ABox is consistent.*

Proof. If the algorithm is started on input ABox $\mathcal{A}$ returning consistent, then there exists a precompletion $\mathcal{A}_p$ for $\mathcal{A}$ that does not contain a clash and all reduction concepts $C_1, \ldots, C_n$ of $\mathcal{A}_p$ that are passed as arguments to the sat algorithm are satisfiable. We show that this implies that $\mathcal{A}_p$ has a model, which, by Lemma 3.9 Point 1 and the definition of precompletion, proves the proposition.

Let $\mathcal{I}_1, \ldots, \mathcal{I}_n$ be the models of the reduction concepts $C_1, \ldots, C_n$ and $a_i : \exists R_i.E_i$ be the assertion in $\mathcal{A}_p$ that triggered the construction of the reduction concept C_i. W.l.o.g., we assume that

- $\Delta_{\mathcal{I}_i} \cap \Delta_{\mathcal{I}_j}$ for $1 \leq i < j \leq n$ and
- $\Delta_{\mathcal{I}_i} \cap \mathsf{O_a} = \emptyset$ for $1 \leq i \leq n$.

For each i with $1 \leq i \leq n$, we fix an element $d_i \in \Delta_{\mathcal{I}_i}$ with $d_i \in C_i^{\mathcal{I}_i}$. Moreover, we fix a solution δ for $\zeta_{\mathcal{A}_p}$, which exists since $\mathcal{A}_p$ is clash-free. Define an interpretation $\mathcal{I}$ as follows:

1. $\Delta_{\mathcal{I}} := \{a \in \mathsf{O_a} \mid a \text{ used in } \mathcal{A}_p\} \uplus \Delta_{\mathcal{I}_1} \uplus \cdots \uplus \Delta_{\mathcal{I}_n}$,
2. $A^{\mathcal{I}} := \{a \in \mathsf{O_a} \mid a : A \in \mathcal{A}_p\} \cup \bigcup_{1 \leq i \leq n} A^{\mathcal{I}_i}$ for all $A \in \mathsf{N_C}$,
3. $R^{\mathcal{I}} := \{(a, b) \in \mathsf{O_a} \times \mathsf{O_a} \mid (a, b) : R \in \mathcal{A}\} \cup \{(a_i, d_i) \mid 1 \leq i \leq n \text{ and } R = R_i\}$
   $\cup \bigcup_{1 \leq i \leq n} R^{\mathcal{I}_i}$ for all $R \in \mathsf{N_R}$,
4. $g^{\mathcal{I}} := \{(a, \delta(x)) \in \mathsf{O_a} \times \Delta_{\mathcal{D}} \mid (a, x) : g \in \mathcal{A}\} \cup \bigcup_{1 \leq i \leq n} g^{\mathcal{I}_i}$ for all $g \in \mathsf{N_{cF}}$,
5. $a^{\mathcal{I}} := a$ for all $a \in \mathsf{O_a}$, and
6. $x^{\mathcal{I}} := \delta(x)$ for all $x \in \mathsf{O_c}$.

$\mathcal{I}$ is well-defined: due to the non-applicability of the Rfe rule to $\mathcal{A}_p$, $f^{\mathcal{I}}$ and $g^{\mathcal{I}}$ are functional for all $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. The following claim is an easy consequence of the construction of $\mathcal{I}$:

Claim: Let $1 \leq i \leq n$. For all $d \in \Delta_{\mathcal{I}_i}$ and $C \in \mathsf{sub}(\mathcal{A}_p)$, $d \in C^{\mathcal{I}_i}$ implies $d \in C^{\mathcal{I}}$.

It remains to show that $\mathcal{I}$ is a model of $\mathcal{A}_p$, i.e., that all assertions in $\mathcal{A}_p$ are satisfied by $\mathcal{I}$. For assertions of the form $(a, b) : R$ and $(a, x) : g$, this is an immediate consequence of the definition of $\mathcal{I}$. Assertions $a \neq b$ are satisfied since $\mathcal{A}_p$ is clash-free and assertions $(x_1, \ldots, x_n) : P$ since δ is a solution for $\zeta_{\mathcal{A}_p}$. It thus remains to show that $a : C \in \mathcal{A}_p$ implies $a \in C^{\mathcal{I}}$. This is done by induction over the structure of C as in the proof of Lemma 3.9, Point 2. The only differences are in the following cases of the induction step:

- $a : \exists R.E \in \mathcal{A}_p$. Then there is an i with $1 \leq i \leq n$ such that $a = a_i$, $R = R_i$, and $E = E_i$ appears as a conjunct in the reduction concept C_i. By definition of $\mathcal{I}$, we have $(a, d_i) \in R^{\mathcal{I}}$. By the above claim together with $d_i \in C_i^{\mathcal{I}_i}$, we have $d_i \in C_i^{\mathcal{I}}$. Since E is a conjunct in C_i, this clearly implies $d_i \in E^{\mathcal{I}}$ and thus $a \in (\exists R.E)^{\mathcal{I}}$.
- $a : \forall R.E \in \mathcal{A}_p$. Fix a $b \in \Delta_{\mathcal{I}}$ such that $(a, b) \in R^{\mathcal{I}}$. Then either b is an R-successor of a in $\mathcal{A}_p$ or $a = a_i$, $R = R_i$, and $b = d_i$ for some $1 \leq i \leq n$. The first case was already treated in the proof of Lemma 3.9, Point 2. Hence, let us stick to the second case. By construction of C_i, E appears as a conjunct in C_i. By the claim, we have $d_i \in C_i^{\mathcal{I}}$ and hence $d_i \in E^{\mathcal{I}}$.

❑

Proposition 3.16 (Completeness). *If the precompletion algorithm is started on a consistent input ABox, then it returns* consistent.

Proof. Suppose that the algorithm is started on a consistent ABox $\mathcal{A}$. By Lemma 3.9 Point 1, the precompl function can apply the completion rules such that only consistent ABoxes are obtained. Hence, by Lemma 3.5, the precompl function generates a consistent precompletion $\mathcal{A}_p$ of $\mathcal{A}$. Consistency of $\mathcal{A}_p$ clearly implies that the reduction concepts constructed from $\mathcal{A}_p$ are satisfiable. Since, by Proposition 3.7, the sat function terminates, the precompletion algorithm also terminates and returns consistent. ❏

It remains to analyze the time and space requirements of our algorithm.

Proposition 3.17.

1. *If $\mathcal{D}$-satisfiability is in* PSPACE, *then the precompletion algorithm can be executed in polynomial space.*
2. *If $\mathcal{D}$-satisfiability is in* NEXPTIME, *then the precompletion algorithm can be executed in nondeterministic exponential time.*
3. *If $\mathcal{D}$-satisfiability is in* EXPSPACE, *then the precompletion algorithm can be executed in exponential space.*

Proof. Let $\mathcal{A}$ be the input ABox to the precompletion algorithm. By Lemma 3.5, the precompl function terminates after at most $|\mathcal{A}|^3$ steps generating an ABox $\mathcal{A}'$ of size at most $|\mathcal{A}|^5$. Since all complexity classes mentioned in the proposition are oblivious for polynomial blowups of the input, the concrete domain satisfiability check does not spoil the upper bound on the time/space requirements. Concerning the calls to the sat function, it is enough to refer to Proposition 3.12. ❏

As in the previous section, we use the PSPACE lower bound of $\mathcal{ALC}$-concept satisfiability and the fact that PSPACE =NPSPACE and EXPSPACE =NEXPSPACE to obtain the following theorem.

Theorem 3.18. *Let $\mathcal{D}$ be an admissible concrete domain.*

1. *If $\mathcal{D}$-satisfiability is in* PSPACE, *then $\mathcal{ALCF}(\mathcal{D})$-ABox consistency is* PSPACE-*complete.*
2. *If $\mathcal{D}$-satisfiability is in* C $\in$ {NEXPTIME, EXPSPACE}, *then $\mathcal{ALCF}(\mathcal{D})$-ABox consistency is also in* C.

3.3 Discussion

In this chapter, we have established tight upper bounds for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability and $\mathcal{ALCF}(\mathcal{D})$-ABox consistency for a large class of concrete domains. The upper bound for concept satisfiability has been obtained by a completion algorithm that uses the tracing technique while the upper bound for ABox consistency has been established by a precompletion-style reduction to concept satisfiability. We have strictly separated the algorithms for these two inference problems since this allows for a clearer presentation and makes more precise the additional means necessary for dealing with ABoxes instead of with concepts. We will keep up this separation between concept satisfiability algorithms and ABox consistency algorithms throughout this thesis. However, for the implementation of DL reasoners that can decide ABox consistency, it may in some cases be more appropriate to use a "direct" ABox consistency algorithm instead of reducing this reasoning task to concept satisfiability. Considering the description of the two algorithms given in this chapter, it should be straightforward to devise such an algorithm.

The established upper bounds are readily applicable to a wide range of interesting concrete domains. For example, Theorem 3.18 implies that $\mathcal{ALCF}(\mathsf{R})$-ABox consistency, where $\mathcal{ALCF}(\mathsf{R})$ is $\mathcal{ALCF}(\mathcal{D})$ instantiated with the expressive concrete domain R introduced in Section 2.4.2, is ExpSpace-complete; Theorem 3.13 implies that $\mathcal{ALCF}(\mathtt{I})$-concept satisfiability, where $\mathtt{I}$ is the interval-based temporal concrete domain from Section 2.4.3, is PSpace-complete. This latter result has already found applications in knowledge representation. In [Artale & Lutz 1999], it is used to obtain a PSpace upper bound for concept satisfiability in the interval-based temporal Description Logic $\mathcal{TL}$-$\mathcal{ALCF}$, which was first described in [Artale & Franconi 1998]. In [Kullmann *et al.* 2000], the logic $\mathcal{ALCF}(\mathtt{I})$ has been used for temporal reasoning in the application domain of disaster management.

Chapter 4

Acyclic TBoxes and Complexity

In the following chapters, we will investigate the complexity of reasoning with various extensions of $\mathcal{ALC}(\mathcal{D})$, an important one being by acyclic TBoxes. To allow a proper evaluation of the obtained results, it is convenient to relate them to known results for other Description Logics that are in some sense comparable. In contrast to Description Logics that are fragments of $\mathcal{ALC}$ [Donini *et al.* 1997; Calvanese 1996a; Buchheit *et al.* 1998], the complexity of reasoning with Description Logics that contain $\mathcal{ALC}$ as a fragment *and* are equipped with acyclic TBoxes is not well explored. Therefore, in this chapter we perform a general analysis of the impact of acyclic TBoxes on the complexity of reasoning with Description Logics "above" $\mathcal{ALC}$. In doing so, we concentrate on logics for which concept satisfiability is PSPACE-complete.

As already mentioned in Section 2.2.1, Nebel [1990] was the first to investigate the complexity of Description Logics with acyclic TBoxes. He showed that, for the very simple Description Logic $\mathcal{TL}$ whose concept language offers only conjunction and universal value restriction, concept subsumption is in PTIME if no TBoxes are present and co-NP-complete if acyclic TBoxes are admitted. Hence, there exist Description Logics for which the presence of acyclic TBoxes makes the complexity of reasoning harder. On the other hand, it is common knowledge in the Description Logic community that, if concept satisfiability without reference to TBoxes is PSPACE-complete for a Description Logic $\mathcal{L}$, then admitting acyclic TBoxes does "usually" not increase the complexity of $\mathcal{L}$-concept satisfiability. However, to the best of our knowledge, this conviction has never been used to obtain formal complexity results. One reason for this may be that researchers concentrated on the more expressive cyclic and general TBoxes. We argue that there are good reasons to investigate the complexity of reasoning with acyclic TBoxes as well: first, the complexity of reasoning with acyclic TBoxes is in many cases lower than the complexity of reasoning with general TBoxes. Hence, acyclic TBoxes should be preferred over general TBoxes if their expressive power is sufficient for the application at hand. Second, there exist Description Logics for which reasoning with general TBoxes is undecidable but reasoning with acyclic TBoxes is not. In many cases, it is more desirable to sacrifice expressivity instead of losing decidability.

In this chapter, we establish formal grounds for the above mentioned common knowledge: we show that, for many "standard" PSPACE-complete Description Logics such as $\mathcal{ALC}$ and $\mathcal{ALCN}^{\sqcap}$, admitting acyclic TBoxes does indeed not increase the complexity of reasoning. However, we also show that there exist counterexamples to this general observation, i.e., logics for which the complexity of reasoning *does* get significantly harder if acyclic TBoxes are admitted. More precisely, this chapter is organized as follows: we start with devising a modification technique allowing to extend existing completion algorithms that establish a PSPACE upper bound by using the tracing technique to take into account acyclic TBoxes. We apply this modification technique to the standard completion algorithm with tracing for $\mathcal{ALC}$ and argue that it can also be applied to many other such algorithms. Hence, we prove $\mathcal{ALC}$-concept satisfiability w.r.t. acyclic TBoxes to be PSPACE-complete. We then extend the modification technique to *pre*completion algorithms and the PSPACE-completeness result to $\mathcal{ALC}$-ABox consistency. Finally, we show that the modification technique cannot succeed in all cases: $\mathcal{ALCF}$-concept satisfiability without reference to TBoxes is known to be PSPACE-complete, as first proved in [Hollunder & Nutt 1990] and reproved in Chapter 3 using a completion algorithm with tracing. Surprisingly, we are able to show that $\mathcal{ALCF}$-concept satisfiability w.r.t. acyclic TBoxes is NEXPTIME-complete. Thus, for this logic, admitting acyclic TBoxes results in a dramatic increase of complexity. It should be noted that $\mathcal{ALCF}$-concept satisfiability w.r.t. general TBoxes is undecidable as a consequence of Theorem 6.3 in [Baader *et al.* 1993]. Thus, it is rather natural to consider $\mathcal{ALCF}$ with acyclic TBoxes.

4.1 PSPACE Upper Bounds

We present the standard completion algorithm for $\mathcal{ALC}$ that uses the tracing technique, as first described in [Schmidt-Schauß & Smolka 1991], and demonstrate how it can be modified to take into account acyclic TBoxes.[1] We then argue that the presented modification strategy can also be applied to tracing-style algorithms for several extensions of $\mathcal{ALC}$.

4.1.1 $\mathcal{ALC}$ with Acyclic TBoxes

The completion algorithm for $\mathcal{ALC}$ is a restricted version of the completion algorithm for $\mathcal{ALCF}(\mathcal{D})$ presented in Chapter 3: the two algorithms are identical except that some completion rules and several clash conditions are dropped. For the sake of completeness, we nevertheless present a detailed description. As with the $\mathcal{ALCF}(\mathcal{D})$-algorithm, we assume that the input concept D is in negation normal form. If the satisfiability of an input concept D is to be decided, the completion algorithm is started with the *initial ABox* $\mathcal{A}_D = \{a : D\}$. The completion algorithm repeatedly applies completion rules to the initial ABox until either a contradiction is found or no more completion rule is applicable. The completion rules are already known from Figures 3.2 and 3.5 and can be found in Figure 4.1. The rule set includes the non-deterministic rule R$\sqcup$ and, thus, the described algorithm is a non-deterministic decision procedure. The $\mathcal{ALC}$-completion algorithm uses only a single clash condition:

[1] In the form presented here, the algorithm first appeared in [Baader & Hollunder 1991b].

R$\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{A}(a)$ then $\mathcal{A} := \mathcal{A} \cup \{a : C_1, a : C_2\}$
R$\sqcup$	if $C_1 \sqcup C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \cap \mathcal{A}(a) = \emptyset$ then $\mathcal{A} := \mathcal{A} \cup \{a : C\}$ for some $C \in \{C_1, C_2\}$
R$\exists$	if $\exists R.C \in \mathcal{A}(a)$ and there is no R-successor b of a with $C \in \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{(a, b) : R, b : C\}$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R$\forall$	if $\forall R.C \in \mathcal{A}(a)$, b is an R-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$

Figure 4.1: Completion rules for $\mathcal{ALC}$.

```
define procedure sat(𝒜)
  while a rule R from {R⊓, R⊔} is applicable to 𝒜 do
    Apply R to 𝒜
    if 𝒜 contains a clash then
      return unsatisfiable
  forall ∃R.C ∈ 𝒜(a) do
    if sat({a : C} ∪ {a : E | ∀R.E ∈ 𝒜(a)}) = unsatisfiable then
      return unsatisfiable
  return satisfiable
```

Figure 4.2: The $\mathcal{ALC}$-concept satisfiability algorithm.

Definition 4.1 (Clash). An ABox $\mathcal{A}$ contains a *clash* iff $\{A, \neg A\} \subseteq \mathcal{A}(a)$ for some concept name A and some object $a \in \mathsf{O_a}$. If $\mathcal{A}$ does not contain a clash, then $\mathcal{A}$ is called *clash-free*. $\diamond$

The algorithm itself can be found in Figure 4.2. It tries to construct a tree model for the input concept by performing depth-first search over role successors without keeping the entire model in memory. As already noted in Section 3.1.1, it is sufficient to consider only tree models since it is well-known that $\mathcal{ALC}$ has the tree model property [Halpern & Moses 1992]. Similar to the R$\exists$r and R$\forall$r rules of the $\mathcal{ALCF}(\mathcal{D})$ algorithm, the R$\exists$ and R$\forall$ rules are not applied explicitly by the algorithm but only implicitly through recursion calls. In contrast to the $\mathcal{ALCF}(\mathcal{D})$-algorithm, each recursion step of the $\mathcal{ALC}$-completion algorithm does only concern the single object "a" instead of a cluster of objects connected by features.[2] We shall later see that this makes a considerable difference if acyclic TBoxes are admitted.

Soundness and completeness of the $\mathcal{ALC}$-completion algorithm are an immediate consequence of the facts that

1. every $\mathcal{ALC}$-concept is an $\mathcal{ALCF}(\mathcal{D})$-concept,

[2] This also means that using ABoxes as the underlying data structure for the $\mathcal{ALC}$-completion algorithm is in fact overkill; see the K-world algorithm from Modal Logic [Ladner 1977].

2. the $\mathcal{ALC}$-completion algorithm is the restriction of the $\mathcal{ALCF}(\mathcal{D})$-completion algorithm to $\mathcal{ALC}$-concepts: if started on an $\mathcal{ALC}$-concept, the $\mathcal{ALCF}(\mathcal{D})$-completion algorithm will obviously never apply the rules $\mathsf{R}\exists\mathsf{f}$, $\mathsf{R}\forall\mathsf{f}$, Rc, $\mathsf{R}\downarrow$, $\mathsf{R}\uparrow$, and Rfe; similarly, the clash conditions 2 to 4 from Definition 3.4 will also never apply.

3. the $\mathcal{ALCF}(\mathcal{D})$-completion algorithm terminates and is sound and complete.

Along the same lines, one can show that the $\mathcal{ALC}$-completion algorithm needs polynomial space in the worst case.

How can the $\mathcal{ALC}$-completion algorithm be generalized to take into account acyclic TBoxes? Using the unfolding technique described in Section 2.2.1 as a preprocessing step yields decidability of concept satisfiability w.r.t. acyclic TBoxes, but this is not an adequate means to obtain a PSPACE upper bound: since unfolding may result in an exponential blowup in concept length [Nebel 1990], the result of unfolding the input concept cannot be stored in polynomial space. Hence, we have to pursue a different approach. Firstly, the input TBox is converted into a certain normal form and then the completion algorithm is modified to operate solely with concept names that are defined in the TBox instead of with complex concepts.

Definition 4.2 (Simple TBox). A TBox $\mathcal{T}$ is called *simple* iff it is acyclic and, additionally, satisfies the following conditions:

- the right-hand side of each concept definition in $\mathcal{T}$ is of the form $\neg A$, $A_1 \sqcap A_2$, $A_1 \sqcup A_2$, $\exists R.A_1$, or $\forall R.A_1$ where $A_1, A_2 \in \mathsf{N_C}$,
- if the right hand side of a concept definition in $\mathcal{T}$ is $\neg A$, then A does not occur on the left hand side of any concept definition in $\mathcal{T}$.

$\diamond$

See [Calvanese 1996a; Buchheit *et al.* 1998] for similar normal forms. Let $\mathcal{T}$ and $\mathcal{T}'$ be TBoxes. We say that a model $\mathcal{I}$ of $\mathcal{T}$ *can be extended* to a model of $\mathcal{T}'$ if there exists a model $\mathcal{I}'$ of $\mathcal{T}'$ that agrees with $\mathcal{I}$ on the interpretation of all concept and role names used in $\mathcal{T}$. The following lemma shows that restricting ourselves to simple TBoxes is not a limitation.

Lemma 4.3. *Any acyclic TBox $\mathcal{T}$ can be converted into a simple one $\mathcal{T}'$ in polynomial time such that $\mathcal{T}'$ is equivalent to $\mathcal{T}$ in the following sense: any model of $\mathcal{T}'$ can be extended to a model of $\mathcal{T}$ and vice versa.*

Proof. Let $\mathcal{T}$ be an acyclic TBox. The conversion can be done in three steps:

1. Eliminate non-atomic negation. Firstly, convert the right-hand sides of all concept definitions in $\mathcal{T}$ to NNF. Then add a new concept definition $\overline{A} \doteq \mathsf{nnf}(\neg C)$ for each definition $A \doteq C$ in $\mathcal{T}$ where $\overline{A}$ is a concept name not appearing in $\mathcal{T}$. Finally, replace every occurrence of $\neg A$ in $\mathcal{T}$ with $\overline{A}$ if A occurs on the left-hand side of a concept definition in $\mathcal{T}$.

2. Break up concepts. Exhaustively apply the following rewrite rules, where C denotes a concept such that $C \notin \mathsf{N_C}$ and D denotes an arbitrary concept.

$$\begin{array}{lll} A \doteq C \sqcap D & \rightsquigarrow & A \doteq A' \sqcap D,\ A' = C \quad \text{(and analogous for } \sqcup) \\ A \doteq D \sqcap C & \rightsquigarrow & A \doteq D \sqcap A',\ A' = C \quad \text{(and analogous for } \sqcup) \\ A \doteq \exists R.C & \rightsquigarrow & A \doteq \exists R.A',\ \ A' = C \quad \text{(and analogous for } \forall) \end{array}$$

   In all cases, A' is a concept name not yet used in $\mathcal{T}$.

3. Eliminate redundant names. For each concept definition $A \doteq A'$ with $A, A' \in \mathsf{N_C}$, replace every occurrence of A' in $\mathcal{T}$ with A. Remove the definition from $\mathcal{T}$.

It is straightforward to show that the TBox resulting from the above procedure is acyclic and satisfies the two conditions from Definition 4.2. The loosened form of equivalence is necessary since $\mathcal{T}'$ may contain additional concept names introduced in Steps 1 and 2, and, moreover, some "redundant" concept names from $\mathcal{T}$ may have been deleted in Step 3. Assume that the above procedure is applied to a TBox $\mathcal{T}$. The first step can be performed in time polynomial in $|\mathcal{T}|$ since NNF conversion needs at most linear time (c.f. Section 2.1.1) and $|\mathcal{T}|$ such conversions are performed. Since the number of rewrite rule applications in the second step is bounded by the number of constructors in $\mathcal{T}$, this step can clearly also be performed in polynomial time. This obviously also holds for the last step. ❑

Since the conversion of a TBox $\mathcal{T}$ into a simple one $\mathcal{T}'$ as in Lemma 4.3 can be done in polynomial time, the size of $\mathcal{T}'$ is obviously polynomial in the size of the original TBox $\mathcal{T}$. Hence, we may use conversion of the input TBox into simple form as a preprocessing step for the completion algorithm without running into the same problems as with unfolding.

We now modify the $\mathcal{ALC}$-completion algorithm to decide the satisfiability of concept names w.r.t. simple TBoxes. It is easily seen that the resulting algorithm also allows to decide the satisfiability of possibly complex concepts D w.r.t. acyclic TBoxes $\mathcal{T}$: just add a definition $A \doteq D$ to $\mathcal{T}$, where A was not previously used in $\mathcal{T}$, convert the resulting TBox into an equivalent one $\mathcal{T}'$ in simple form according to Lemma 4.3 and start the modified algorithm with $(A, \mathcal{T}')$. This yields the desired result since, as is easily seen, A is satisfiable w.r.t. $\mathcal{T}'$ iff D is satisfiable w.r.t. $\mathcal{T}$.

The modified algorithm uses ABoxes of a restricted form as the underlying data structure: in assertions of the form $a : C$, we require C to be a concept name. In the following, such ABoxes are called *simple*.

Definition 4.4 (Modified Completion Algorithm). The *modified $\mathcal{ALC}$-completion algorithm* tboxsat is obtained from the $\mathcal{ALC}$-completion algorithm sat in Figure 4.2 by the following modifications:

1. To decide the satisfiability of a concept name A w.r.t. a simple TBox $\mathcal{T}$, the tboxsat algorithm starts with the initial ABox $\mathcal{A}_A := \{a : A\}$ where $a \in \mathsf{O_a}$.

2. The completion rules are modified as follows: in the premise of each completion rule, substitute

$$\text{“}C \in \mathcal{A}(a)\text{”} \quad \text{with} \quad \text{“}B \in \mathcal{A}(a) \text{ and } B \doteq C \in \mathcal{T}\text{”}.$$

For example, in the conjunction rule, "$C_1 \sqcap C_2 \in \mathcal{A}(a)$" is replaced by "$B \in \mathcal{A}(a)$ and $(B \doteq C_1 \sqcap C_2) \in \mathcal{T}$";

◇

Note that the second modification also concerns implicit applications of the R∃ and R∀ rules in recursion calls. More precisely, the **forall** condition now reads

$$\textbf{forall } A \in \mathcal{A}(a) \text{ such that } A \doteq (\exists R.C) \in \mathcal{T}$$

and the argument to recursion calls is

$$\{b : C\} \cup \{b : E \mid A \in \mathcal{A}(a) \text{ and } (A \doteq \forall R.E) \in \mathcal{T}\}.$$

Intuitively, the modified algorithm is sound, complete, and terminates since the original algorithm has these properties and there exists a one-to-one correspondence between runs of the modified algorithm on input $A, \mathcal{T}$ and runs of the original algorithm on input C, where C is the result of unfolding A w.r.t. $\mathcal{T}$. In the following, we make this correspondence more precise. We use $\mathsf{unfold}(C, \mathcal{T})$ to denote the result of unfolding the concept C w.r.t. the TBox $\mathcal{T}$. The following notion captures the relationship between ABoxes $\mathcal{A}$ in runs of the modified algorithm on input $A, \mathcal{T}$ and corresponding ABoxes $\mathcal{A}'$ in runs of the original algorithm on input $\mathsf{unfold}(A, \mathcal{T})$.

Definition 4.5. A simple ABox $\mathcal{A}$ is a *variant* of an ABox $\mathcal{A}'$ w.r.t. a TBox $\mathcal{T}$ iff the following conditions hold:

1. $a : A \in \mathcal{A}$ and $\mathsf{unfold}(A, \mathcal{T}) = C$ implies $a : C \in \mathcal{A}'$,
2. $a : C \in \mathcal{A}'$ implies the existence of an $A \in \mathsf{N_C}$ such that $a : A \in \mathcal{A}$ and $\mathsf{unfold}(A, \mathcal{T}) = C$, and
3. $(a, b) : R \in \mathcal{A}$ iff $(a, b) : R \in \mathcal{A}'$ for all assertions of this form.

◇

The following lemma establishes the correspondence between runs of the two algorithms.

Lemma 4.6. *Let $\mathcal{A}_1$ be a simple ABox. Moreover, let $\mathcal{A}_1$ be a variant of an ABox $\mathcal{A}'_1$ w.r.t. the simple TBox $\mathcal{T}$.*

- *If the modified completion algorithm can apply a completion rule* R *to $\mathcal{A}_1$ yielding an ABox $\mathcal{A}_2$, then the original completion algorithm can apply* R *to $\mathcal{A}'_1$ yielding an ABox $\mathcal{A}'_2$ such that $\mathcal{A}_2$ is a variant of $\mathcal{A}'_2$ w.r.t. $\mathcal{T}$.*
- *Conversely, if the original completion algorithm can apply a completion rule* R *to $\mathcal{A}'_1$ yielding an ABox $\mathcal{A}'_2$, then the modified completion algorithm can apply* R *to $\mathcal{A}_1$ yielding a variant $\mathcal{A}_2$ of $\mathcal{A}'_2$ w.r.t. $\mathcal{T}$.*

Proof. The proof is by a straightforward case distinction according to the type of R. We only treat one case exemplarily, namely $\mathsf{R} = \mathsf{R}\sqcap$.

First assume that $\mathsf{R}\sqcap$ was applied by the modified completion algorithm to an assertion $a : A$ in $\mathcal{A}_1$ with $(A \doteq A_1 \sqcap A_2) \in \mathcal{T}$. Then $\mathcal{A}_2 = \mathcal{A}_1 \cup \{a : A_1, a : A_2\}$. Since $\mathcal{A}_1$ is a variant of $\mathcal{A}_1'$ w.r.t. $\mathcal{T}$, $a : C$ is in $\mathcal{A}_1'$ with $C = \mathsf{unfold}(A, \mathcal{T})$. By definition of unfolding, we have $C = C_1 \sqcap C_2$ with $C_1 = \mathsf{unfold}(A_1, \mathcal{T})$ and $C_2 = \mathsf{unfold}(A_2, \mathcal{T})$. Hence, the original algorithm may apply $\mathsf{R}\sqcap$ to $\mathcal{A}_1'$ yielding $\mathcal{A}_2' = \mathcal{A}_1' \cup \{a : C_1, a : C_2\}$. It is readily checked that $\mathcal{A}_2$ is a variant of $\mathcal{A}_2'$ w.r.t. $\mathcal{T}$.

Now assume that $\mathsf{R}\sqcap$ was applied by the original completion algorithm to an assertion $a : C_1 \sqcap C_2$ in $\mathcal{A}_1'$. Then $\mathcal{A}_2' = \mathcal{A}_1' \cup \{a : C_1, a : C_2\}$. Since $\mathcal{A}_1$ is a variant of $\mathcal{A}_1'$ w.r.t. $\mathcal{T}$, there exists an A such that $a : A \in \mathcal{A}_1$ and $\mathsf{unfold}(A, \mathcal{T}) = C_1 \sqcap C_2$. By definition of unfolding and since $\mathcal{T}$ is simple, this implies the existence of A_1, A_2 such that $A \doteq A_1 \sqcap A_2 \in \mathcal{T}$, $\mathsf{unfold}(A_1, \mathcal{T}) = C_1$, and $\mathsf{unfold}(A_2, \mathcal{T}) = C_2$. Hence, the modified algorithm can apply $\mathsf{R}\sqcap$ to $\mathcal{A}_1$ yielding $\mathcal{A}_2 = \mathcal{A}_1 \cup \{a : A_1, a : A_2\}$ which is obviously a variant of $\mathcal{A}_2'$ w.r.t. $\mathcal{T}$. ❑

We now establish correctness and termination and investigate the space requirements of the modified algorithm.

Proposition 4.7. *The* tboxsat *algorithm terminates on any input, is sound and complete, and can be executed in polynomial space.*

Proof. Termination, soundness, and completeness are an immediate consequence of Lemma 4.6 and the facts that

1. the original algorithm is sound, complete, and terminating;
2. a concept name A and a simple TBox $\mathcal{T}$ have a model iff the concept $C = \mathsf{unfold}(A, \mathcal{T})$ has a model; and
3. recursion calls can be viewed as rule applications.

Assume that the tboxsat algorithm is started on a concept name A and a TBox $\mathcal{T}$. It is an immediate consequence of the following facts that tboxsat needs at most polynomial space.

- The recursion depth is bounded by $|\mathcal{T}|$. This can also be shown using the correspondence between the original and the modified algorithm. However, we choose a more direct approach since, later on, this will help us to characterize the class of completion algorithms to which the described modification technique is applicable.

  Let C be the result of unfolding A w.r.t. $\mathcal{T}$. For an ABox $\mathcal{A}$ constructed during a recursion step of the algorithm, set

  $$\mathsf{rd}_{\mathcal{T}}(\mathcal{A}) := \max\{\mathsf{rd}(D) \mid \text{there exists } a : A \in \mathcal{A} \text{ such that } D = \mathsf{unfold}(A, \mathcal{T})\}$$

  where $\mathsf{rd}(D)$ denotes the role depth of the concept D. It is not hard to see that the value $\mathsf{rd}_{\mathcal{T}}(\mathcal{A})$ strictly decreases with recursion depth: if $\mathcal{A}'$ is the result of applying the $\mathsf{R}\sqcap$ or $\mathsf{R}\sqcup$ rule to an ABox $\mathcal{A}$, then $\mathsf{rd}_\mathcal{T}(\mathcal{A}') = \mathsf{rd}_\mathcal{T}(\mathcal{A})$; moreover, if $\mathcal{A}$ is an ABox considered during a recursion step and $\mathcal{A}'$ is constructed from $\mathcal{A}$ as an argument to a recursion call, then $\mathsf{rd}_\mathcal{T}(\mathcal{A}') < \mathsf{rd}_\mathcal{T}(\mathcal{A})$. Since, for the initial ABox $\mathcal{A}_A$, we clearly have $\mathsf{rd}_\mathcal{T}(\mathcal{A}_A) = \mathsf{rd}(C)$, it remains to show that $\mathsf{rd}(C) \leq |\mathcal{T}|$. Assume to the contrary that the role depth of C exceeds $|\mathcal{T}|$. This means that the right- hand side of some concept definition $A' \doteq \exists R.D$ or $A' \doteq \forall R.D$ in $\mathcal{T}$ contributes to the role depth of C more than once. From this, however, it follows that unfolding D w.r.t. $\mathcal{T}$ yields a concept containing A' which is a contradiction to the acyclicity of $\mathcal{T}$.

- The size of ABoxes $\mathcal{A}$ constructed in recursion steps is linear in $|\mathcal{T}|$. This is the case since each such ABox contains at most a single object a and, if $a : A$ occurs in $\mathcal{A}$, then A is a concept name occurring in $\mathcal{T}$.

❑

Together with the known PSpace-hardness of $\mathcal{ALC}$-concept satisfiability without reference to TBoxes [Schmidt-Schauß & Smolka 1991] and the fact that NPSpace = PSpace [Savitch 1970], we obtain the following result.

Theorem 4.8. *$\mathcal{ALC}$-concept satisfiability w.r.t. acyclic TBoxes is* PSpace*-complete.*

Since subsumption can be reduced to (un)satisfiability, we have that $\mathcal{ALC}$-concept subsumption w.r.t. acyclic TBoxes is also PSpace-complete.

4.1.2 A Rule of Thumb

We argue that the use of the presented modification scheme is not limited to $\mathcal{ALC}$. In order to give an intuition for when the described modification technique can be applied yielding a PSpace algorithm, let us summarize why the approach is successful in the case of $\mathcal{ALC}$.

We started with a completion algorithm for $\mathcal{ALC}$-concept satisfiability (without reference to TBoxes) that uses the tracing technique, i.e., that constructs a model by performing depth-first search over role successors without keeping the entire model in memory. This original algorithm can be executed in polynomial space because

1. its recursion depth is bounded by the role depth of the input concept D and
2. in each recursion step, the size of the constructed ABox $\mathcal{A}$ is bounded by $|\mathsf{sub}(D)|$ (since each such ABox contains only a single object).

The modified completion algorithm `tboxsat` has very similar properties. Assume that it is started on a concept name A and a simple TBox $\mathcal{T}$ and that $C = \mathsf{unfold}(A, \mathcal{T})$.

1. as shown in the proof of Proposition 4.7, the recursion depth of the modified algorithm is bounded by the role depth of C. We obtain a linear bound for the recursion depth of the modified algorithm since the role depth is "polynomial under unfolding", i.e., the role depth of C is linear in $|\mathcal{T}|$.

2. Similarly, the size of ABoxes $\mathcal{A}$ constructed in recursion steps of the modified algorithm is bounded by $|\mathsf{sub}(C)|$. The function $|\mathsf{sub}(C)|$ is also polynomial under unfolding in the above sense, i.e., $|\mathsf{sub}(C)|$ is linear in $|\mathcal{T}|$. A formal proof of the latter claim is ommitted here and will be given later (Lemma 5.56).

We may generalize the above observations into a rule of thumb that characterizes a class of completion algorithms to which the modification technique may be applied successfully. To do this, we first formalize what it means for a function to be preserved by unfolding.

Definition 4.9. A function f mapping concepts to natural numbers is called *polynomial under unfolding* iff there exists a polynomial p such that, for all concept names A and simple TBoxes $\mathcal{T}$, the result C of unfolding A w.r.t. $\mathcal{T}$ satisfies $f(C) \leq p(|\mathcal{T}|)$. $\diamond$

As was shown in the proof of Proposition 4.7, the role-depth of concepts is an example for a function that is polynomial under unfolding. An example for a function which is not polynomial under unfolding is the length of concepts. This is implied by Nebel's result that unfolding may cause an exponential blowup in concept length [Nebel 1990]).

Rule of Thumb. *The described modification technique can be applied to all completion algorithms that decide $\mathcal{L}$-concept satisfiability for some Description Logic $\mathcal{L}$. Assume that the algorithm performs depth-first search over role-successors and can be executed in polynomial space. If there exist functions f and g that are polynomial under unfolding such that the size of ABoxes constructed during recursion steps is bounded by $f(C)$ and the recursion depth is bounded by $g(C)$, then the modified algorithm can also be expected to be executable in polynomial space.*

The key to understand polynomiality under unfolding and the rule of thumb is the one-to-one-correspondence between runs of the modified algorithm on input $A, \mathcal{T}$ and runs of the original algorithm on the result C of unfolding A w.r.t. $\mathcal{T}$. Since the functions f and g are polynomial under unfolding, we can carry over the polynomial space requirement of the original algorithm started on C to the modified algorithm started on A and $\mathcal{T}$.

Let us discuss an example application of the rule of thumb. In [Donini *et al.* 1997], the Description Logic $\mathcal{ALCN}^{\sqcap}$ is introduced, which extends $\mathcal{ALC}$ by role conjunction and unqualifying number restrictions of the form $\geq nR$ and $\leq nR$ where $n \in \mathbb{N}$. These constructors are a restricted form of the qualifying number restrictions introduced in Section 2.1.2 and their semantics is defined as

$$(\geq nR)^{\mathcal{I}} := (\geq nR.\top)^{\mathcal{I}} \qquad \text{and} \qquad (\leq nR)^{\mathcal{I}} := (\leq nR.\top)^{\mathcal{I}}.$$

Donini et al. describe an algorithm for deciding $\mathcal{ALCN}^{\sqcap}$-concept satisfiability without reference to TBoxes that performs depth-first search over role successors. The authors assume that numbers inside number restrictions are coded unarily and show that, under this assumption, their algorithm can be executed using only polynomial space. More precisely, the algorithm's recursion depth is bounded by the role depth of the input concept D. The ABoxes considered in each recursion step contain at most $\mathsf{ex}(D) + \mathsf{nsum}(D) + 1$ objects, at most $|\mathsf{sub}(D)|$ assertions of the form $a : C$ per object a, and at most $|\mathsf{sub}(D)|$ assertions of the form $(a, b) : R$ per pair of objects a, b. Here, $\mathsf{ex}(D)$ is the number of distinct subconcepts of D that are of the form $\exists R.E$ and $\mathsf{nsum}(D) = \sum_{\geq nR \,\in\, \mathsf{sub}(D)} n$.

Since Donini et al. assume unary coding of numbers, $\mathsf{nsum}()$ is obviously polynomial under unfolding. It is easy to prove that this also holds for $\mathsf{ex}()$: assume that C is the result of unfolding a concept name A w.r.t. a simple TBox $\mathcal{T}$ and that $\mathsf{ex}(C) > |\mathcal{T}|$. For each $\exists R.E \in \mathsf{sub}(C)$, we fix a concept equation $B \doteq \exists R.B'$ in $\mathcal{T}$ that was "used for generating" $\exists R.E$ during unfolding, i.e., the concept name B was replaced by $\exists R.B'$ and then B' was unfolded to E. This concept equation is denoted by $\mathcal{E}(\exists R.E)$. By definition of acyclic TBoxes, $\mathcal{E}(\exists R.E) = \mathcal{E}(\exists R'.E')$ implies $\exists R.E = \exists R'.E'$. Since the number of distinct subconcepts of the form $\exists R.E$ in $\mathsf{sub}(C)$ exceeds $|\mathcal{T}|$, there exist *distinct* concepts $\exists R.E$, $\exists R'.E' \in \mathsf{sub}(C)$ such that $\mathcal{E}(\exists R.E) = \mathcal{E}(\exists R'.E')$: a contradiction.

Hence, there exist functions f and g as required and we may apply the rule of thumb to the $\mathcal{ALCN}^{\sqcap}$-completion algorithm obtaining the following conjecture.

Conjecture. *$\mathcal{ALCN}^{\sqcap}$-concept satisfiability w.r.t. acyclic TBoxes is* PSPACE*-complete.*

This conjecture illustrates the usefulness of the devised rule of thumb. It is, however, out of the scope of this thesis to prove the conjectured result which can be done by modifying Donini et al.'s algorithm in the same way as we modified the $\mathcal{ALC}$ completion algorithm in Section 4.1.1.

We claim that the rule of thumb covers a large class of completion algorithms that use the tracing technique. However, there exist notable exceptions. Consider for example the $\mathcal{ALCF}(\mathcal{D})$-completion algorithm described in Chapter 3. Its recursion depth is bounded by the role depth of its input and thus we have found a function g that is polynomial under unfolding and as required by the rule of thumb. But it is not at all obvious how to find an function f that is polynomial under unfolding and describes the size of ABoxes generated in each recursion step. In fact, such a function does not exist: in Section 4.2, we will see that $\mathcal{ALCF}$-concept satisfiability w.r.t. acyclic TBoxes is already NEXPTIME-hard.

4.1.3 $\mathcal{ALC}$-ABox Consistency

The strategy for modifying completion algorithms to take into account acyclic TBoxes can also be applied to precompletion algorithms that decide ABox consistency. We illustrate this by showing how the standard precompletion algorithm for $\mathcal{ALC}$-ABox consistency can be modified to take into account acyclic TBoxes.

The $\mathcal{ALC}$-precompletion algorithm is presented in Figure 4.3. It exhaustively applies the completion rules R$\sqcap$, R$\sqcup$, and R$\forall$ to the input ABox and then constructs a number of reduction concepts that are passed to the $\mathcal{ALC}$ completion algorithm for satisfiability checking. The construction of the reduction concepts can be viewed as an (implicit) application of the R$\exists$ rule together with multiple applications of the R$\forall$ rule. Obviously, the $\mathcal{ALC}$-precompletion algorithm is nothing but the restric-

define procedure $\mathsf{cons}(\mathcal{A})$
  while a rule $\mathsf{R} \in \{\mathsf{R}\sqcap, \mathsf{R}\sqcup, \mathsf{R}\forall\}$ is applicable to $\mathcal{A}$ **do**
    Apply R to $\mathcal{A}$
  if $\mathcal{A}$ contains a clash **then**
    return inconsistent
  forall objects a in $\mathsf{O_a}$ and $\exists R.C \in \mathcal{A}(a)$ **do**
    Fix $b \in \mathsf{O_a}$
    if $\mathsf{sat}(\{b : C \sqcap \bigsqcap_{\forall R.E \in \mathcal{A}(a)} b : E) = \mathsf{unsatisfiable}$ **then**
      return inconsistent
  return consistent

Figure 4.3: The $\mathcal{ALC}$-ABox consistency algorithm.

tion of the $\mathcal{ALCF}(\mathcal{D})$-precompletion algorithm to $\mathcal{ALC}$-concepts. This implies that termination, soundness, and completeness of the $\mathcal{ALC}$-precompletion algorithm is an immediate consequence of termination, soundness, and completeness of the $\mathcal{ALCF}(\mathcal{D})$-precompletion algorithm.

We now modify the $\mathcal{ALC}$-ABox consistency algorithm to take into account acyclic TBoxes. The modified algorithm decides consistency of simple ABoxes w.r.t. simple TBoxes. It can obviously also be used to decide consistency of regular ABoxes $\mathcal{A}$ w.r.t. acyclic TBoxes $\mathcal{T}$: extend the TBox $\mathcal{T}$ by a new concept definition $A_C \doteq C$ for each concept C appearing in $\mathcal{A}$, where A_C is a concept name not yet appearing in $\mathcal{T}$; then convert $\mathcal{T}$ into simple form according to Lemma 4.3 and replace C by A_C in $\mathcal{A}$ for all concepts C appearing in $\mathcal{A}$. Call the resulting ABox $\mathcal{A}'$ and the resulting TBox $\mathcal{T}'$. It is easily seen that $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ iff $\mathcal{A}'$ is consistent w.r.t. $\mathcal{T}'$.

Definition 4.10 (Modified Precompletion Algorithm). The *modified $\mathcal{ALC}$-precompletion algorithm* $\mathsf{tboxcons}$ is obtained from the $\mathcal{ALC}$-precompletion algorithm cons in Figure 4.3 by the following modification:

In the premise of each completion rule, substitute

$$\text{“}C \in \mathcal{A}(a)\text{”} \quad \text{by} \quad \text{“}A \in \mathcal{A}(a) \text{ and } A \doteq C \in \mathcal{T}\text{”}$$

◇

As in Definition 4.4, the modification also concerns the implicit applications of the $\mathsf{R}\exists$ and $\mathsf{R}\forall$ rules, i.e., the **forall** condition and the construction of the reduction conceptsa. We may now establish the correctness of the modified algorithm. In the following, the result of exhaustively applying the modified precompletion rules $\mathsf{R}\sqcap$, $\mathsf{R}\sqcup$, and $\mathsf{R}\forall$ to the input ABox $\mathcal{A}$ using the input TBox $\mathcal{T}$ is called a *precompletion* of $\mathcal{A}$ w.r.t. $\mathcal{T}$.

Proposition 4.11. *The* $\mathsf{tboxcons}$ *algorithm terminates on any input, is sound and complete, and can be executed in polynomial space.*

Proof. Termination, soundness, and completeness can be proved similarly to the corresponding properties of the modified $\mathcal{ALC}$-concept satisfiability algorithm (using Lemma 4.6).

Let $\mathcal{A}$ and $\mathcal{T}$ be the input to the tboxcons algorithm and $\mathcal{A}'$ a precompletion computed by the algorithm. To prove that tboxsat can be executed in polynomial space, we show that, for every precompletion $\mathcal{A}'$ of $\mathcal{A}$ w.r.t. $\mathcal{T}$, $|\mathcal{A}'|$ is polynomial in $|\mathcal{A}| + |\mathcal{T}|$. First consider the number of new assertions of the form $a : C$ that are present in $\mathcal{A}'$ but not in $\mathcal{A}$. Since the precompletion rules do not introduce new object names and both $\mathcal{A}$ and $\mathcal{A}'$ are simple, the number of such new assertions is clearly bounded by $|\mathcal{A}| \cdot |\mathcal{T}|$. Similarly, the number of new assertions $(a, b) : R$ in $\mathcal{A}'$ is bounded by $|\mathcal{A}|^2 \cdot |\mathcal{T}|$. It remains to recall that, by Theorem 4.8, $\mathcal{ALC}$-concept satisifiability w.r.t. acyclic TBoxes can be decided using polynomial space. ❑

Together with the lower bound for $\mathcal{ALC}$-concept satisfiability and the well-known fact that NPSpace = PSpace, we thus obtain the following result.

Theorem 4.12. *$\mathcal{ALC}$-ABox consistency w.r.t. acyclic TBoxes is* PSpace*-complete.*

As in the case of the completion algorithm, the application of the presented modification technique is not limited to the logic $\mathcal{ALC}$. Again, we can straightforwardly generalize the obtained results into a rule of thumb.

Rule of Thumb. *The described modification technique can be applied to all precompletion algorithms that reduce $\mathcal{L}$-ABox consistency to $\mathcal{L}$-concept satisfiability for some Description Logic $\mathcal{L}$. If there exists an function f that is polynomial under unfolding such that the size of the constructed precompletions is bounded by $f(C)$ and $\mathcal{L}$-concept satisfiability w.r.t. acyclic TBoxes is* PSpace*-complete, then the modified algorithm can also be expected to be executable in polynomial space.*

It seems that, if $\mathcal{L}$-concept satisfiability w.r.t. acyclic TBoxes is PSpace-complete for some Description Logic $\mathcal{L}$, then, in most cases, $\mathcal{L}$-ABox consistency w.r.t. acyclic TBoxes is also PSpace-complete. At least there is—to the best of our knowledge—no known counterexample to this claim.

4.2 A Counterexample: $\mathcal{ALCF}$

Given the modification scheme for completion algorithms and the rule of thumb developed in the previous section, it is a natural question to ask whether there are any relevant Description Logics for which reasoning without reference to TBoxes is in PSpace but reasoning w.r.t. acyclic TBoxes is not. In the following, we will answer this question to the affirmative by showing that the complexity of $\mathcal{ALCF}$-concept satisfiability moves from PSpace-complete to NExpTime-complete if TBoxes are admitted.

The PSpace upper bound for $\mathcal{ALCF}$-concept satisfiabiliy without reference to TBoxes has already been established in Chapter 3. We prove the lower bound for $\mathcal{ALCF}$-concept satisfiability w.r.t. acyclic TBoxes by a reduction of a NExpTime-hard variant of the well-known undecidable domino problem [Berger 1966; Knuth 1968]. A domino problem is given by a finite set of *tile types*. All tile types are of the same size, each type has a quadratic shape and colored edges. Of each type, an unlimited number of tiles is available. The problem in the original domino problem is to arrange these tiles to cover the plane without holes or overlapping, such that adjacent tiles have identical colors on their touching edges (rotation of the tiles is not allowed). In the NExpTime-hard variant of the domino problem that we use, the task is not to tile the whole plane, but to tile a $2^{n+1} \times 2^{n+1}$-torus, i.e., a $2^{n+1} \times 2^{n+1}$-rectangle whose edges are "glued" together. See, e.g., [Berger 1966; Knuth 1968] for undecidable versions of the domino problem and [Börger *et al.* 1997] for bounded variants.

Definition 4.13. Let $\mathcal{D} = (T, H, V)$ be a *domino system*, where T is a finite set of *tile types* and $H, V \subseteq T \times T$ represent the horizontal and vertical matching conditions. For $s, t \in \mathbb{N}$, let $U(s,t)$ be the torus $\mathbb{Z}_s \times \mathbb{Z}_t$, where $\mathbb{Z}_n$ denotes the set $\{0, \ldots, n-1\}$. Let $a = a_0, \ldots, a_{n-1}$ be an n-tuple of tiles (with $n \leq s$). We say that $\mathcal{D}$ *tiles* $U(s,t)$ *with initial condition* a iff there exists a mapping $\tau : U(s,t) \rightarrow T$ such that, for all $(x, y) \in U(s,t)$:

- if $\tau(x,y) = t$ and $\tau(x \oplus_s 1, y) = t'$, then $(t, t') \in H$
- if $\tau(x,y) = t$ and $\tau(x, y \oplus_t 1) = t'$, then $(t, t') \in V$
- $\tau(i, 0) = a_i$ for $0 \leq i < n$.

where $\oplus_i$ denotes addition modulo i. Such a mapping τ is called a *solution* for $\mathcal{D}$ w.r.t. a. $\diamond$

These bounded domino systems are capable of expressing the computational behaviour of restricted, so-called simple, Turing machines (TMs).[3] The restriction is non-essential in the following sense: every language accepted in time $T(n)$ and space $S(n)$ by some one-tape TM is accepted within the same time and space bounds by a simple TM, provided that $S(n), T(n) \geq 2n$ [Börger *et al.* 1997].

Theorem 4.14 ([Börger *et al.* 1997], Theorem 6.1.2). *Let M be a simple TM with input alphabet Γ. Then there exists a domino system $\mathcal{D} = (T, H, V)$ and a linear time reduction that takes any input $x \in \Gamma^*$ to an n-tuple a of tiles with $|x| = n$ such that*

- *If M accepts x in time t_0 with space s_0, then $\mathcal{D}$ tiles $U(s,t)$ with initial condition a for all $s \geq s_0 + 2, t \geq t_0 + 2$;*
- *if M does not accept x, then $\mathcal{D}$ does not tile $U(s,t)$ with initial condition a for any $s, t \geq 2$.*

This theorem implies the existence of NExpTime-complete domino problems:

Corollary 4.15. *There exists a domino system $\mathcal{D}$ such that the following is a* NExpTime*-hard problem: given an initial condition $a = a_0 \cdots a_{n-1}$ of length n, does $\mathcal{D}$ tile the torus $U(2^{n+1}, 2^{n+1})$ with initial condition a?*

[3]We introduce Turing machines in more detail in Section 5.1.

Proof. It is not hard to see that there exists a non-deterministic TM M with time- (and hence also space-) bound 2^n that decides a NExpTime-hard language over some alphabet Γ. We only sketch the proof: take a non-deterministic TM M' that decides an arbitrary NExpTime-hard language L with time- and space-bound 2^{n^d} for some integer constant d. For any word $x \in \Gamma^*$ and every integer constant $e > 1$, we define $\mathsf{bu}_e(x)$ as the "blowup" of x to size $|x|^e$ by padding with a fixed symbol $s \notin \Gamma$. For each such e, we define a Turing machine M_e as follows:

1. the input alphabet of M_e is $\Gamma \cup \{s\}$,
2. if started on an input x, M_e first checks whether x is in the range of $\mathsf{bu}_e(x)$. If this is not the case, it rejects the input. Otherwise, it behaves just as M' with the only difference that the s symbol is treated in the same way as M' treats the blank symbol.

Obviously, for every fixed e, the language accepted by the Turing machine M_e is also NExpTime-hard. It is easy to check that we can choose e such that M_e accepts any input with time- and space-bound 2^n: let w be an input to M_e. Then we have $w = xv$, where $|v| = |x|^e - |x|$. The initial "checking" of the input can obviously be done in time $p(|w|)$ for some polynomial p and the succeeding computation can be done in time $2^{|x|^d}$. Hence, we need to choose e such that

$$2^{|x|^e} \geq p(|x|^e) + 2^{|x|^d}.$$

Now let M be a TM with time- and space-bound 2^n accepting a NExpTime-hard language L and w.l.o.g. assume that M is simple. Let $\mathcal{D}$ be the corresponding domino system and tran the reduction from Theorem 4.14. The function tran is a linear reduction of L to the problem formulated in the corollary: for $x \in \Gamma^*$ with $|x| = n$, we have $x \in L$ iff M accepts x in time and space $2^{|x|}$ iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition $\mathsf{tran}(x)$. ❏

In the following, the NExpTime-hard domino problem from Corollary 4.15 is reduced to $\mathcal{ALCF}$-concept satisfiability w.r.t. acyclic TBoxes. Given a domino system $\mathcal{D} = (T, H, V)$ with initial condition a, we define a concept (name) $C_{\mathcal{D},a}$ and an acyclic TBox $\mathcal{T}_{\mathcal{D},a}$ of size polynomial in $|T| + |a|$ such that $C_{\mathcal{D},a}$ is satisfiable w.r.t. $\mathcal{T}_{\mathcal{D},a}$ iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition a (where $n = |a|$). The reduction TBox $\mathcal{T}_{\mathcal{D},a}$ is split into the two Figures 4.4 and 4.6, where the concept definitions in Figure 4.4 are needed twice: once with α, β, γ replaced by ℓ_1, r_1, y and once with α, β, γ replaced by ℓ_2, r_2, x. With ℓ_i, r_i, x, y, and z_i, we denote features (there is no distinction between abstract and concrete features since $\mathcal{ALCF}$ provides for abstract features, only). We first informally describe the reduction and then prove its correctness.

Models of $C_{\mathcal{D},a}$ w.r.t. $\mathcal{T}_{\mathcal{D},a}$ describe a grid of size 2^{n+1} which has the form of a torus and is properly tiled by $\mathcal{D}$ with initial condition a. The nodes of the grid are represented by domain elements, horizontal edges are represented by the feature x and vertical edges by the feature y . How is the grid enforced? The first task is to define two cyclic "feature chains" with 2^{n+1} nodes each. One chain will be "row 0" of the grid while the other one will be "column 0". The chains are established by defining a

$$
\begin{array}{rcl}
\mathsf{Tree}_0[\alpha,\beta,\gamma] & \doteq & \beta^{n+1}\gamma{\downarrow}\alpha^{n+1} \\
& & \sqcap\ \exists\alpha.\mathsf{Tree}_1[\alpha,\beta,\gamma] \sqcap \exists\beta.\mathsf{Tree}_1[\alpha,\beta,\gamma] \\
& & \sqcap\ \alpha\beta^n\gamma{\downarrow}\beta\alpha^n \\
\mathsf{Tree}_1[\alpha,\beta,\gamma] & \doteq & \exists\alpha.\mathsf{Tree}_2[\alpha,\beta,\gamma] \sqcap \exists\beta.\mathsf{Tree}_2[\alpha,\beta,\gamma] \\
& & \sqcap\ \alpha\beta^{n-1}\gamma{\downarrow}\beta\alpha^{n-1} \\
& & \vdots \\
\mathsf{Tree}_n & \doteq & \exists\alpha.\mathsf{Tree}_{n+1}[\alpha,\beta,\gamma] \sqcap \exists\beta.\mathsf{Tree}_{n+1} \\
& & \sqcap\ \alpha\gamma{\downarrow}\beta \\
\mathsf{Tree}_{n+1} & \doteq & \mathsf{Grid}_{n+1}
\end{array}
$$

Figure 4.4: The $\mathcal{ALCF}$-reduction TBox $\mathcal{T}_{\mathcal{D},a}$ with $|a| = n$: tree definition. Substitute α, β, γ with ℓ_1, r_1, y and ℓ_2, r_2, x.

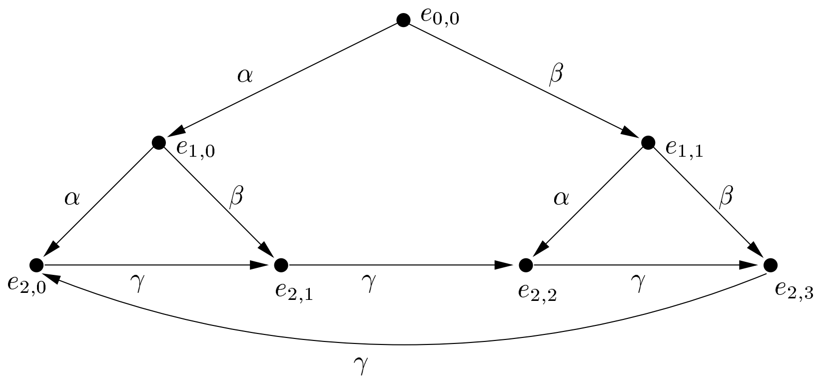

Figure 4.5: An example model of $\mathsf{Tree}_0[\alpha, \beta, \gamma]$ with $n = 1$.

binary tree of depth $n + 1$ whose leaf nodes are connected by a cyclic feature chain. The relevant concept Tree_0 can be found in Figure 4.4. Since two trees are needed, the TBox in the Figure has to be instantiated twice as described above. The first instantiation yields a cyclic y-chain with 2^{n+1} nodes and the second one a cyclic x-chain of the same length. An example model of the Tree_0 concept can be found in Figure 4.5.

To see how the rest of the grid is established, consider the concept $C_{\mathcal{D},a}$ in Figure 4.6, which glues together all the necessary building parts. It refers to the Tree_0 concept to build up the cyclic feature chains and enforces the identification of their "start" nodes. The remainig grid is built by the Grid_i concepts. The features $z_0, \ldots, z_n$ are diagonals in the grid (each z_i spans 2^i "grid cells") and play a central role in the grid definition: the use of these diagonals allows the definition of the exponentially sized grid by an (acyclic) TBox of polynomial size. First observe that, by definition of the Tree_{n+1} concept, each domain element on the two cyclic feature chains (row 0 and column 0 of the torus to be defined) is in the extension of Grid_{n+1} and hence also of Grid_0. Because of this, each element on the chains has coinciding z_0-, xy-, and

$$
\begin{array}{rcl}
\mathsf{Grid}_0 & \doteq & \mathsf{Tile} \sqcap xy{\downarrow}yx \sqcap xy{\downarrow}z_0 \\
\mathsf{Grid}_1 & \doteq & \mathsf{Grid}_0 \sqcap z_0z_0{\downarrow}z_1 \sqcap \exists z_0.\mathsf{Grid}_0 \\
& & \vdots \\
\mathsf{Grid}_n & \doteq & \mathsf{Grid}_{n-1} \sqcap z_{n-1}z_{n-1}{\downarrow}z_n \sqcap \exists z_{n-1}.\mathsf{Grid}_{n-1} \\
\mathsf{Grid}_{n+1} & \doteq & \mathsf{Grid}_n \sqcap \exists z_n.\mathsf{Grid}_n \\
\mathsf{Tile} & \doteq & \bigsqcup_{t \in T} A_t \sqcap \bigsqcap_{t \in T} \bigsqcap_{t' \in T \setminus \{t\}} \neg(A_t \sqcap A_{t'}) \\
& & \sqcap \bigsqcap_{t \in T} (A_t \rightarrow \exists x. \bigsqcup_{(t,t') \in H} A_{t'}) \\
& & \sqcap \bigsqcap_{t \in T} (A_t \rightarrow \exists y. \bigsqcup_{(t,t') \in V} A_{t'}) \\
\mathsf{Init} & \doteq & \exists \ell_2^{n+1}.(A_{a_0} \sqcap \exists x.(A_{a_1} \sqcap \cdots \sqcap \exists x.(A_{a_{n-2}} \sqcap \exists x.A_{a_{n-1}}) \ldots)) \\
C_{\mathcal{D},a} & \doteq & \mathsf{Tree}_0[\ell_1, r_1, y] \sqcap \mathsf{Tree}_0[\ell_2, r_2, x] \sqcap \ell_1^{n+1}{\downarrow}\ell_2^{n+1} \sqcap \mathsf{Init}
\end{array}
$$

Figure 4.6: The $\mathcal{ALCF}$ reduction TBox $\mathcal{T}_{\mathcal{D},a}$ with $n = |a|$: grid definition and tiling.

yx-successors . Together with the cyclicity of the initial feature chains, this properly defines row 1 and column 1 of the torus. Since the elements on the initial chains are in the extension of Grid_1, the elements on row 1 and column 1, which are z_0-successors of elements on the initial chains, are in the extension of Grid_0. Hence, we can repeat the argument for row/column 1 and conclude the proper definition of row/column 2. Now observe that the elements on row/column 2 are z_1-successors of elements on the initial chains. This implies that they are in the extension of both the Grid_0 and Grid_1 concept and we can repeat the entire argument to derive the existence of rows/columns 3 and 4. This "doubling" can be repeated $n+1$ times due the existence of the features $z_0, \ldots, z_n$ and yields rows/columns $0, \ldots, 2^{n+1}$ of the torus. The cyclicity of the initial feature chains ensures that the edges of the grid are properly "glued" to form a torus, i.e., that row/column 2^{n+1} coincides with row/column 0. Figure 4.7 shows a clipping of a model of $\mathcal{T}_{\mathcal{D},a}$.

The grid represents the structure to be tiled. The final task is to define the tiling itself. Tile types are represented by concept names A_t. Due to the definition of Grid_0, each node in the grid is in the extension of the concept Tile. This ensures that, horizontally as well as vertically, the tiling condition is satisfied. The Init concept enforces the initial condition $a = a_0, \ldots, a_{n-1}$.

Lemma 4.16. *$C_{\mathcal{D},a}$ is satisfiable w.r.t. $\mathcal{T}_{\mathcal{D},a}$ iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition a where $n = |a|$.*

Proof. First for the "only if" direction. Let $\mathcal{I}$ be a model of $C_{\mathcal{D},a}$ and $\mathcal{T}_{\mathcal{D},a}$. To prove that $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition a, we show that there is a mapping τ as in Definition 4.13.

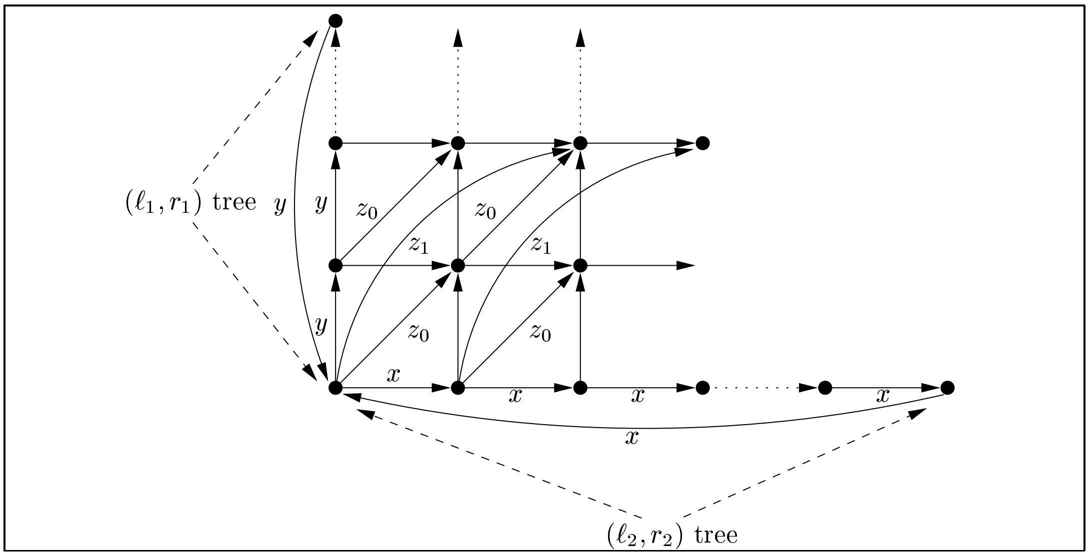

Figure 4.7: Clipping of a model of the reduction concept C.

Using induction on n and the definitions of the Tree_i concepts and the $C_{\mathcal{D},a}$ concept, it is easy to show that there exist domain elements $e_{i,j}$ for $0 \leq i \leq n+1$ and $0 \leq j < 2^i$ such that $e_{0,0} \in C^{\mathcal{I}}_{\mathcal{D},a}$, $e_{0,0} \in (r_1^{n+1} y{\downarrow}\ell_1^{n+1})^{\mathcal{I}}$ and the following holds:

1. $e_{i,j} \in \mathsf{Tree}_i[\ell_1, r_1, y]^{\mathcal{I}}$ for $i \leq n+1$ and $j < 2^i$,
2. $\ell_1^{\mathcal{I}}(e_{i,j}) = e_{(i+1),2j}$ and $r_1^{\mathcal{I}}(e_{i,j}) = e_{(i+1),(2j+1)}$ for $i \leq n$ and $j < 2^i$, and
3. $e_{i,j} \in (\ell_1 r_1^{n-i} y{\downarrow} r_1 \ell_1^{n-i})^{\mathcal{I}}$ for $i \leq n$ and $j < 2^i$.

Intuitively, Property 2 states that the $d_{i,j}$ form a binary tree in which edges connecting left successors are labeled with ℓ_1 and edges connecting right successors are labeled with r_1.[4] Property 3 states that the leaves of the tree are connected by a "cyclic chain" w.r.t. the feature y. The naming scheme for nodes is as indicated in Figure 4.5.

Similarly, we may show the existence of domain elements $e'_{i,j}$ for $0 \leq i \leq n+1$ and $0 \leq j < 2^i$ such that $e'_{0,0} = e_{0,0}$, $e_{(n+1),0} = e'_{(n+1),0}$, $e'_{0,0} \in (r_2^{n+1} y{\downarrow}\ell_2^{n+1})^{\mathcal{I}}$, and the following holds:

1. $e'_{i,j} \in \mathsf{Tree}_i[\ell_2, r_2, x]^{\mathcal{I}}$ for $i \leq n+1$ and $j < 2^i$,
2. $\ell_1^{\mathcal{I}}(e'_{i,j}) = e'_{(i+1),2j}$ and $r_1^{\mathcal{I}}(e'_{i,j}) = e'_{(i+1),(2j+1)}$ for $i \leq n$ and $j < 2^i$, and
3. $e'_{i,j} \in (\ell_2 r_2^{n-i} x{\downarrow} r_2 \ell_2^{n-i})^{\mathcal{I}}$ for $i \leq n$ and $j < 2^i$.

Intuitively, this means that there exists another binary tree sharing the root and the "leftmost" leaf with the previous one.

We now show that the model $\mathcal{I}$ describes a torus of size $2^{n+1} \times 2^{n+1}$. Observe that the elements $e_{(n+1),0}, \ldots, e_{(n+1),(2^{n+1}-1)}$ and the elements $e'_{(n+1),0}, \ldots, e'_{(n+1),(2^{n+1}-1)}$

[4]Note that nodes in the tree are not necessarily distinct.

are in the extension of the concepts $\mathsf{Grid}_i^{\mathcal{I}}$ for $0 \leq i \leq n+1$. The former elements will be column 0 of the torus while the latter will be row 0 of the torus. The following claim can be proved using induction on k, the definition of the Grid_i concepts, and the "diagonal" features $z_0, \ldots, z_n$.

Claim: If there exist domain elements $d_{i,0}$ with $0 \leq i < 2^{n+1}$, domain elements $d_{0,j}$ with $0 \leq j < 2^{n+1}$, and a $k \leq n$ such that

(i) $y^{\mathcal{I}}(d_{i,0}) = d_{(i \oplus_{2^{n+1}} 1),0}$ for $i < 2^{n+1}$,

(ii) $x^{\mathcal{I}}(d_{0,j}) = d_{0,(j \oplus_{2^{n+1}} 1)}$ for $j < 2^{n+1}$, and

(iii) $d_{0,0}, \ldots, d_{(2^{n+1}-1),0} \in \mathsf{Grid}_k^{\mathcal{I}}$ and $d_{0,0}, \ldots, d_{0,(2^{n+1}-1)} \in \mathsf{Grid}_k^{\mathcal{I}}$,

then there exist domain elements $d_{i,j}$ with $0 \leq i < 2^{n+1}$ and $0 \leq j \leq 2^k$ and domain elements $d_{i,j}$ with $0 \leq i \leq 2^k$ and $0 \leq j < 2^{n+1}$ such that[5]

(a) $y^{\mathcal{I}}(d_{i,j}) = d_{(i \oplus_{2^{n+1}} 1),j}$ for $i < 2^{n+1}$ and $j \leq 2^k$,

(b) $x^{\mathcal{I}}(d_{i,j}) = d_{i,(j+1)}$ for $i < 2^{n+1}$ and $j < 2^k$,

(c) $y^{\mathcal{I}}(d_{i,j}) = d_{(i+1),j}$ for $i < 2^k$ and $j < 2^{n+1}$,

(d) $x^{\mathcal{I}}(d_{i,j}) = d_{i,(j \oplus_{2^{n+1}} 1)}$ for $i \leq 2^k$ and $j < 2^{n+1}$,

(e) $z_k^{\mathcal{I}}(d_{i,0}) = d_{(i \oplus_{2^{n+1}} 2^k),2^k}$ for $i < 2^{n+1}$, and

(f) $z_k^{\mathcal{I}}(d_{0,j}) = d_{2^k,(j \oplus_{2^{n+1}} 2^k)}$ for $j < 2^{n+1}$.

(g) for all above elements $d_{i,j}$, we have $d_{i,j} \in \mathsf{Grid_0}^{\mathcal{I}}$.

Intuitively, if the claim is applied to row 0 and column 0 of the torus, then the elements in the consequent describe the leftmost 2^k+1 columns and the lower 2^k+1 rows of the torus. Apply the claim to the domain elements from above $e_{(n+1),0}, \ldots, e_{(n+1),(2^{n+1}-1)}$ (these are the $d_{i,0}$) and $e'_{(n+1),0}, \ldots, e'_{(n+1),(2^{n+1}-1)}$ (these are the $d_{0,j}$) with $k = n$. We obtain elements $d_{i,j}$ with $i < 2^{n+1}$ and $j \leq 2^n$ and $d_{i,j}$ with $i \leq 2^n$ and $j < 2^{n+1}$ as in the claim. Since $y^{\mathcal{I}}(d_{2^{n+1}-1,0}) = d_{0,0}$ and $x^{\mathcal{I}}(d_{0,2^{n+1}-1}) = d_{0,0}$ and due to Point (g), it is readily checked that

- $y^{\mathcal{I}}(d_{2^{n+1}-1,j}) = d_{0,j}$ for $1 \leq j \leq 2^k$ and
- $x^{\mathcal{I}}(d_{i,2^{n+1}-1}) = d_{i,0}$ for $1 \leq i \leq 2^k$.

Hence, we have obtained the leftmost half and the upper half of the torus. Due to Points (e) and (f) from above and since the elements $e_{(n+1),0}, \ldots, e_{(n+1),(2^{n+1}-1)}$ and $e'_{(n+1),0}, \ldots, e'_{(n+1),(2^{n+1}-1)}$ are in Grid_{n+1}, we have

$$d_{0,2^k}, \ldots, d_{(2^{n+1}-1),2^k}, d_{2^k,0}, \ldots, d_{2^k,(2^{n+1}-1)} \in \mathsf{Grid}_n^{\mathcal{I}}.$$

[5]Note that these two sets of elements overlap each other and also contain the initial elements $d_{i,0}$ and $d_{0,i}$ from the antecedent.

Hence, we may again apply the claim with $k = n$, the elements $d_{i,0}$ set to

$$d_{2^k,2^k}, \ldots, d_{(2^{n+1}-1),2^k}, d_{0,2^k}, \ldots, d_{(2^k-1),2^k}$$

and the elements $d_{0,j}$ set to

$$d_{2^k,2^k}, \ldots, d_{2^k,(2^{n+1}-1)}, d_{2^k,0}, \ldots, d_{2^k,(2^k-1)}.$$

By doing this, we obtain the rightmost half and the upper half of the torus. The entire torus is comprised of domain elements $d_{i,j}$ with $0 \leq i < 2^{n+1}$ and $0 \leq j < 2^{n+1}$ that are connected by x and y features as expected, and, by Property (g) from above, in the extension of Grid_0. Hence, all $d_{i,j}$ are also in the extension of Tile. Define the mapping τ as follows:

$$\text{set } \tau(i,j) = t \text{ if } d_{i,j} \in A_t.$$

Since all $d_{i,j}$ are in the extension of Tile, this mapping is total and well-defined and the horizontal and vertical tiling conditions are satisfied. Moreover, since obviously $e_{0,0} \in \mathsf{Init}^{\mathcal{I}}$, the initial condition a is also met.

Now for the "if" direction. Assume that the domino system $\mathcal{D}$ tiles the torus $U(2^{n+1}, 2^{n+1})$ with initial condition a of length n. This means that there exists a mapping τ as in Definition 4.13. We use this mapping to define a model $\mathcal{I}$ of $C_{\mathcal{D},a}$ and $\mathcal{T}_{\mathcal{D},a}$, details can be found in Figure 4.8. It is straightforward to check that $\mathcal{I}$ is a model of $C_{\mathcal{D},a}$ and $\mathcal{T}_{\mathcal{D},a}$: the $e_{i,j}$ domain elements form a tree of depth $n+1$ where edges are labeled with ℓ_1 and r_1. The $n+1$'st level of the tree consists of the elements $d_{0,0}, \ldots, d_{0,(2^{n+1}-1)}$. Similarly, the $e'_{i,j}$ elements form an ℓ_2, r_2-tree where the root is the element $e_{0,0}$ (i.e., shared with the (ℓ_1, r_1)-tree) and the $n+1$'st level consists of the elements $d_{0,0}, \ldots, d_{(2^{n+1}-1),0}$. The $d_{i,j}$ elements are arranged in a grid w.r.t. the features x and y (and diagonals z_i) which satisfies the Tile and Init concept since the extension of the A_t concepts is defined through the tiling τ respecting the initial condition a. We conclude that $\mathcal{I}$ is a model of $C_{\mathcal{D},a}$ and $\mathcal{T}_{\mathcal{D},a}$. ❑

It is easy to verify that the size of $\mathcal{T}_{\mathcal{D},a}$ is polynomial in $|T| + |a|$ and can be computed in polynomial time. Hence, we have obtained the following result:

Theorem 4.17. *Satisfiability of $\mathcal{ALCF}$ concepts w.r.t. acyclic TBoxes is* NExpTime-*hard.*

This theorem yields a co-NExpTime lower bound for $\mathcal{ALCF}$-concept subsumption since concept unsatisfiability w.r.t. acyclic TBoxes can be reduced to concept subsumption w.r.t. acyclic TBoxes.

In contrast to (dis)agreements on *roles*, called "role value maps" [Schmidt-Schauß 1989], (dis)agreements on features are usually believed to be "harmless" w.r.t. decidability and complexity. The above result indicates that this is not always the case. If general TBoxes are admitted, things get even worse: the given reduction can easily be extended to an undecidability proof. Consider the following TBox:

$$\{\top \doteq xy{\downarrow}yx, \quad \top \doteq \mathsf{Tile}\}$$

$$\Delta_{\mathcal{I}} = \{d_{i,j} \mid 0 \leq i,j < 2^{n+1}\} \cup \{e_{i,j},\ e'_{i,j} \mid 0 \leq i \leq n,\ 0 \leq j < 2^i\}$$

for $i \leq n$, set
$\mathsf{Tree}_i[\ell_1, r_1, y]^{\mathcal{I}} := \{e_{i,j} \mid 0 \leq j < 2^i\}$
$\mathsf{Tree}_i[\ell_2, r_2, x]^{\mathcal{I}} := \{e'_{i,j} \mid 0 \leq j < 2^i\}$

$\ell_1^{\mathcal{I}}(e_{0,0}) := e_{1,0}, \quad r_1^{\mathcal{I}}(e_{0,0}) := e_{1,1}, \quad \ell_2^{\mathcal{I}}(e_{0,0}) := e'_{1,0}, \quad r_2^{\mathcal{I}}(e_{0,0}) := e'_{1,1}$

for $i < n$ and $j < 2^i$, set
$\ell_1^{\mathcal{I}}(e_{i,j}) := e_{(i+1),(2j)}, \quad r_1^{\mathcal{I}}(e_{i,j}) := e_{(i+1),(2j+1)}$
$\ell_2^{\mathcal{I}}(e'_{i,j}) := e'_{(i+1),(2j)}, \quad r_2^{\mathcal{I}}(e'_{i,j}) := e'_{(i+1),(2j+1)}$

$\mathsf{Tree}_{n+1}^{\mathcal{I}} := \mathsf{Grid}_{n+1}^{\mathcal{I}} := \{d_{0,j},\ d_{j,0} \mid 1 \leq j < 2^{n+1}\}$

for $i < 2^n$, set
$\ell_1^{\mathcal{I}}(e_{n,i}) := d_{0,(2i)}, \quad r_1^{\mathcal{I}}(e_{n,i}) := d_{0,(2i+1)},$
$\ell_2^{\mathcal{I}}(e'_{n,i}) := d_{(2i),0}, \quad r_2^{\mathcal{I}}(e'_{n,i}) := d_{(2i+1),0}$

for $i,j < 2^{n+1}$, set $x^{\mathcal{I}}(d_{i,j}) := d_{(i \oplus_{2^{n+1}} 1),j}, \quad y^{\mathcal{I}}(d_{i,j}) := d_{i,(j \oplus_{2^{n+1}} 1)}$

for $i,j < 2^{n+1}$ and $k < n$, set $z_k^{\mathcal{I}}(d_{i,j}) := d_{(i \oplus_{2^{n+1}} 2^k),(j \oplus_{2^{n+1}} 2^k)}$

for $i < 2^{n+1}$, set $z_n^{\mathcal{I}}(d_{i,0}) := d_{(i \oplus_{2^{n+1}} 2^n),2^n}, \quad z_n^{\mathcal{I}}(d_{0,i}) := d_{2^n,(i \oplus_{2^{n+1}} 2^n)}$

$\mathsf{Grid}_0^{\mathcal{I}} := \cdots := \mathsf{Grid}_n^{\mathcal{I}} := \mathsf{Tile}^{\mathcal{I}} := \{d_{i,j} \mid 0 \leq i,j < 2^{n+1}\}$

for $t \in T$, set $A_t^{\mathcal{I}} := \{d_{i,j} \mid \tau(i,j) = t\}$

$\mathsf{Init}^{\mathcal{I}} := C_{\mathcal{D},a}^{\mathcal{I}} := \{e_{0,0}\}$

Figure 4.8: Constructing a model $\mathcal{I}$ from a function τ.

where Tile is defined as in Figure 4.6. Models of this TBox have the form of an infinite grid and it is not hard to see that satisfiability of the concept $\top$ w.r.t. the above TBox implies the existence of a complete tiling of the first quadrant of the plane and vice versa. As shown in [Knuth 1968], the variant of the domino problem that requires tilings of the first quadrant of the plane is already undecidable. Hence, we have proved undecidability of $\mathcal{ALCF}$-concept satisfiability w.r.t. general TBoxes. This result is already known from feature logic [Baader *et al.* 1993, Theorem 6.3], where it was proved using a more complicated reduction of the word problem for finitely presented groups.

4.3 The Upper Bound

Our main goal in this section is to prove a NExpTime upper bound for $\mathcal{ALCF}$-concept satisfiability w.r.t. acyclic TBoxes matching the lower bound established in the previous section. However, in Section 5.5.1, we will need a NExpTime upper bound for

$\mathcal{ALCF}^{\sqcap}$-concept satisfiability without reference to TBoxes—where $\mathcal{ALCF}^{\sqcap}$ is $\mathcal{ALCF}$ extended with the role conjunction constructor introduced in Section 2.1.2. To avoid double work, we combine both logics and, in this section, prove a NExpTime upper bound for $\mathcal{ALCF}^{\sqcap}$-concept satisfiability w.r.t. acyclic TBoxes. This upper bound is established by proving a *bounded model property (BMP)*, i.e., by showing that, if an $\mathcal{ALCF}^{\sqcap}$-concept C is satisfiable w.r.t. an acyclic TBox $\mathcal{T}$, then there exists a model of C and $\mathcal{T}$ of size exponential in the size of C and $\mathcal{T}$. The BMP induces the following simple decision procedure: to decide the satisfiability of a concept C w.r.t. an acyclic TBox $\mathcal{T}$, a non-deterministic Turing machine may "guess" an interpretation $\mathcal{I}$ of exponential size and check whether it is a model of C and $\mathcal{T}$. Due to the bounded model property and that, as we shall see, checking whether an interpretation is a model of a given concept and TBox can be done in linear time, this yields a NExpTime decision procedure.

Some introductory remarks concerning the Description Logic $\mathcal{ALCF}^{\sqcap}$ are in order: $\mathcal{ALCF}^{\sqcap}$ allows the application of the role conjunction constructor introduced in Section 2.1.2 to both roles and features (even intermixed). However, this constructor may only be used inside existential and universal value restrictions and not inside feature (dis)agreements. Hence, $\exists(R \sqcap f \sqcap f').A$ is an $\mathcal{ALCF}^{\sqcap}$-concept, but $f \downarrow (f \sqcap R)$ and $f_1 \downarrow (f_2 \sqcap f_3)$ are not.

For developing a decision procedure for $\mathcal{ALCF}^{\sqcap}$-concepts w.r.t. acyclic TBoxes, we restrict ourselves to *simple* $\mathcal{ALCF}^{\sqcap}$-TBoxes which are defined as in Definition 4.2 with the only difference that right-hand sides may also be of the form $\exists(R_1 \sqcap \cdots \sqcap R_n).C$, $\forall(R_1 \sqcap \cdots \sqcap R_n).C$, $p_1{\downarrow}p_2$, or $p_1{\uparrow}p_2$. Since Lemma 4.3 obviously generalizes from simple $\mathcal{ALC}$-TBoxes to simple $\mathcal{ALCF}^{\sqcap}$-TBoxes, this can be done without loss of generality. Moreover, as argued in Section 4.1.1, it is sufficient to decide satisfiability of concept *names* w.r.t. simple TBoxes since it is straightforward to reduce satisfiability of (possibly complex) concepts C w.r.t. acyclic TBoxes to this task.

We now establish the bounded model property for concept names and simple TBoxes using a technique known as *selective filtration* in Modal Logic [Chagrov & Zakharyaschev 1996]. We first show how models of $\mathcal{ALCF}^{\sqcap}$-concepts can be filtrated into smaller models (without considering TBoxes at all). Intuitively, the (selective) filtration of a model $\mathcal{I}$ of a concept D is obtained by "selecting" the part of $\mathcal{I}$ that is relevant for D. We then use filtrated models to establish the bounded model property for concept names and TBoxes and describe the decision procedure induced by the BMP in more detail. In what follows, the *role depth* of an $\mathcal{ALCF}^{\sqcap}$-concept C is denoted by $\mathsf{rd}(C)$ and defined analogously to the role depth of $\mathcal{ALCF}(\mathcal{D})$-concepts as defined in Section 3.1.3. For example, the role depth of $\exists R_1 \sqcap R_2.(f_1 f_2 \downarrow f_1)$ is 3.

Definition 4.18 (Filtration). Let D be an $\mathcal{ALCF}^{\sqcap}$-concept. Assume that $\mathsf{sub}(D)$ contains k existential concepts, i.e. concepts of the form $\exists(R_1 \sqcap \cdots \sqcap R_n).C$ for some $n \geq 1$, where $R_1, \ldots, R_n \in \mathsf{N_R}$. We assume that this set of concepts is linearly ordered and that $\mathcal{E}(i)$ (with $0 \leq i \leq k$) yields the i-th such concept. A model $\mathcal{J}$ is called a *filtration* of $\mathcal{I}$ w.r.t. D if it can be constructed from $\mathcal{I}$ in the following way.

Fix a domain element $d_0 \in D^{\mathcal{I}}$. Moreover, for each $d \in \Delta_{\mathcal{I}}$ and each $i \leq k$ with $\mathcal{E}(i) = \exists R_1 \sqcap \cdots \sqcap R_n.C$ and $d \in (\exists R_1 \sqcap \cdots \sqcap R_n.C)^{\mathcal{I}}$, fix a witness $\tau(d,i) \in \Delta_{\mathcal{I}}$ such that $(d, \tau(d,i)) \in (R_1 \sqcap \cdots \sqcap R_n)^\mathcal{I}$ and $\tau(d,i) \in C^\mathcal{I}$. An *edge* in $\mathcal{I}$ is a triple (d, e, R) such that $(d,e) \in R^\mathcal{I}$. Determine a set of *selected domain elements* and a set of *selected edges* in $\mathsf{rd}(D) + 1$ rounds as follows:

- In round 0, select d_0 (no edges are selected).
- In round i with $0 < i \leq \mathsf{rd}(D)$, do the following for all elements d that have been selected in round $i - 1$:
  - for all $e \in \Delta_\mathcal{I}$ and features f used in D, if $(d,e) \in f^\mathcal{I}$, then select the object e and the edge (d,e,f).
  - for $j \leq k$, if $d \in (\exists(R_1 \sqcap \cdots \sqcap R_n).C)^\mathcal{I}$ and $\exists(R_1 \sqcap \cdots \sqcap R_n).C = \mathcal{E}(j)$, then select the element $\tau(d,j)$ and the edges $(d, \tau(d,j), R_1), \ldots, (d, \tau(d,j), R_n)$.

Define the interpretation $\mathcal{J}$ as the restriction of $\mathcal{I}$ to the set of selected nodes and edges, i.e.,

- $\Delta_\mathcal{J} := \{d \in \Delta_\mathcal{I} \mid d \text{ is selected}\}$,
- $A^\mathcal{J} := A^\mathcal{I} \cap \Delta_\mathcal{J}$ for all $A \in \mathsf{N_C}$, and
- $R^\mathcal{J} := \{(d,e) \in R^\mathcal{I} \mid (d,e,R) \text{ selected}\}$ for all $R \in \mathsf{N_R}$.

The object $d_0 \in \Delta_\mathcal{J}$ selected in round 0 is called the *root* of $\mathcal{J}$. $\diamond$

Note that, given a model $\mathcal{I}$ of a concept D, there may exist more than one filtration of $\mathcal{I}$ w.r.t. D. Also note that, during the selection procedure, objects may be selected multiple times, since e.g., for distinct (d,i) and (e,j), we may have $\tau(d,i) = \tau(e,j)$.

Our next subgoal is to prove that, if $\mathcal{I}$ is a model of D and $\mathcal{J}$ a filtration of $\mathcal{I}$ w.r.t. D, then $\mathcal{J}$ is also a model of D. To this end, we first establish a new notion and a technical lemma. Let $\mathcal{I}$ be an interpretation and $d_0 \in \Delta_\mathcal{I}$. The mapping $\mathsf{depth}^\mathcal{I}_{d_0}()$ from the domain $\Delta_\mathcal{I}$ into the natural numbers measures the "distance" between d_0 and other domain elements:

- set $\mathsf{depth}^\mathcal{I}_{d_0}(d_0) := 0$;
- for any $d \in \Delta_\mathcal{J}$ with $d \neq d_0$, define $\mathsf{depth}^\mathcal{I}_{d_0}(d)$ to be the length of the shortest sequence $R_1, \ldots, R_k$ of roles such that $(d_0, d) \in R_1^\mathcal{J} \circ \cdots \circ R_k^\mathcal{J}$ and ∞ if no such sequence exists.

The following lemma shows the connection between the selection procedure and the $\mathsf{depth}()$ function.

Lemma 4.19. *Let $\mathcal{I}$ be a model of D, $\mathcal{J}$ a filtration of $\mathcal{I}$ w.r.t. D, and d_0 the root of $\mathcal{J}$. For all $d \in \Delta_\mathcal{J}$, $\mathsf{depth}^\mathcal{J}_{d_0}(d) = \ell$ implies that d was selected in round ℓ during the construction of $\mathcal{J}$.*

Proof. Let $\mathcal{E}()$ and $\tau()$ be as in the construction of $\mathcal{J}$ (see Definition 4.18). The proof is by induction on ℓ. For the induction start, we have $\ell = 0$. By definition of $\mathsf{depth}()$, this implies $d = d_0$. Hence, d was obviously selected in round 0. For the induction step, assume $0 < \ell \leq \mathsf{rd}(D)$. Then there exists a sequence of roles $R_1, \ldots, R_\ell$ such that $(d_0, d) \in R_1^{\mathcal{J}} \circ \cdots \circ R_\ell^{\mathcal{J}}$ which is the shortest sequence with this property. Hence, there exists a domain element e such that $(e, d) \in R_\ell^{\mathcal{J}}$ and $\mathsf{depth}_{d_0}^{\mathcal{J}}(e) = \ell - 1$. By the induction hypothesis, e was selected in round $\ell - 1$. Since $(e, d) \in R_\ell^{\mathcal{J}}$, the edge (e, d, R_ℓ) is selected in $\mathcal{I}$. By definition of the selection procedure, this means that either (i) $R_\ell = f$ for some $f \in \mathsf{N_{aF}}$, and $(e, d) \in f^{\mathcal{I}}$ or (ii) there exists an i such that $\exists(R'_1 \sqcap \cdots \sqcap R'_n).C = \mathcal{E}(i)$, $R_\ell = R'_j$ for some $1 \leq j \leq n$, $e \in \mathcal{E}(i)^{\mathcal{I}}$, and $\tau(e, i) = d$. In both cases, d is selected in round ℓ. ❏

Note that, if D, $\mathcal{J}$, and d_0 are defined as in Lemma 4.19, then $\mathsf{depth}_{d_0}^{\mathcal{J}}(d) \leq \mathsf{rd}(D)$ for all $d \in \Delta_{\mathcal{J}}$. $\mathcal{ALCF}^{\sqcap}$-concepts can be converted into equivalent ones in NNF in precisely the same way as $\mathcal{ALCF}(\mathcal{D})$-concepts, c.f. Section 3.1.2. The following lemma implies that each filtration of a model of a concept D is also a model of D. Without loss of generality, we assume the concept D to be in NNF.

Lemma 4.20. *Let D be a concept in NNF, $\mathcal{I}$ a model of D, $\mathcal{J}$ a filtration of $\mathcal{I}$ w.r.t. D, and d_0 the root of $\mathcal{J}$. For all $d \in \Delta_{\mathcal{J}}$ and $C \in \mathsf{sub}(D)$, we have that $d \in C^{\mathcal{I}}$ and $\mathsf{rd}(C) \leq \mathsf{rd}(D) - \mathsf{depth}_{d_0}^{\mathcal{J}}(d)$ implies $d \in C^{\mathcal{J}}$.*

Proof. Let $\mathcal{E}()$ and $\tau()$ be as in the construction of $\mathcal{J}$ (see Definition 4.18). The proof is by induction on the structure of C. The induction start consists of the following cases:

- C is a concept name. Straightforward by definition of $\mathcal{J}$.
- $C = \neg C_1$. Since D is in NNF and $C \in \mathsf{sub}(D)$, C is also in NNF and hence $C_1 = A$ for some $A \in \mathsf{N_C}$. By definition of $\mathcal{J}$, $d \notin A^{\mathcal{I}}$ implies $d \notin A^{\mathcal{J}}$ and we are done.
- $C = p_1{\uparrow}p_2$ with $p_1 = f_1 \cdots f_n$ and $p_2 = f'_1 \cdots f'_m$. Since $d \in C^{\mathcal{I}}$, there exist domain elements e_1 and e_2 such that $p_1^{\mathcal{I}}(d) = e_1$, $p_2^{\mathcal{I}}(d) = e_2$, and $e_1 \neq e_2$. Since $\mathsf{rd}(C) \leq \mathsf{rd}(D) - \mathsf{depth}_{d_0}^{\mathcal{J}}(d)$ and $\mathsf{rd}(C) = \max\{n, m\}$, we have $\mathsf{depth}_{d_0}^{\mathcal{J}}(d) \leq \mathsf{rd}(D) - \max\{n, m\}$. Hence, by Lemma 4.19, d has been selected in round $j \leq \mathsf{rd}(D) - \max\{n, m\}$. By definition of the selection procedure, this implies that e_1 and e_2 and all the domain elements and edges "between" d and e_1 and "between" d and e_2 are also selected. Thus $d \in C^{\mathcal{J}}$ by definition of $\mathcal{J}$.
- $C = p_1{\downarrow}p_2$. Similar to the previous case.

For the induction step, we make a case distinction according to the topmost constructor in C:

- $C = C_1 \sqcap C_2$. $d \in C^{\mathcal{I}}$ implies $d \in C_1^{\mathcal{I}}$ and $d \in C_2^{\mathcal{I}}$. By induction, we have $d \in C_1^{\mathcal{J}}$ and $d \in C_2^{\mathcal{J}}$ and thus $d \in C^{\mathcal{J}}$.
- $C = C_1 \sqcup C_2$. Similar to the previous case.

- $C = \exists(R_1 \sqcap \cdots \sqcap R_n).C_1$. Then there exists an i such that $\mathcal{E}(i) = C$. Since $\mathsf{rd}(C) \leq \mathsf{rd}(D) - \mathsf{depth}^{\mathcal{J}}_{d_0}(d)$ and $\mathsf{rd}(C) \geq 1$, we have $\mathsf{depth}^{\mathcal{J}}_{d_0}(d) < \mathsf{rd}(D)$. By Lemma 4.19, d has been selected in round $j < \mathsf{rd}(D)$. Because of this and since $d \in \mathcal{E}(i)^{\mathcal{I}}$, $\tau(d,i)$ is selected in round $j+1$. By definition of the selection procedure, we have $(d, \tau(d,i)) \in (R_1 \sqcap \cdots \sqcap R_n)^{\mathcal{J}}$. Moreover, since $\tau(d,i) \in C_1^{\mathcal{I}}$, we have $\tau(d,i) \in C_1^{\mathcal{J}}$ by induction. Hence, $d \in C^{\mathcal{J}}$.

- $C = \forall(R_1 \sqcap \cdots \sqcap R_n).C_1$. Assume $(d,e) \in (R_1 \sqcap \cdots \sqcap R_n)^{\mathcal{J}}$. By definition of $\mathcal{J}$, this implies $(d,e) \in (R_1 \sqcap \cdots \sqcap R_n)^{\mathcal{I}}$. Moreover, we clearly have $\mathsf{depth}^{\mathcal{J}}_{d_0}(e) \leq \mathsf{depth}^{\mathcal{J}}_{d_0}(d) + 1$. Since $\mathsf{rd}(C) \leq \mathsf{rd}(D) - \mathsf{depth}^{\mathcal{J}}_{d_0}(d)$ and, obviously, $\mathsf{rd}(C_1) = \mathsf{rd}(C) - 1$, we have $\mathsf{rd}(C_1) \leq \mathsf{rd}(D) - \mathsf{depth}^{\mathcal{J}}_{d_0}(e)$. Since $d \in C^{\mathcal{I}}$, we have $e \in C_1^{\mathcal{I}}$. Thus, we obtain $e \in C_1^{\mathcal{J}}$ by induction. Since this holds independently of the choice of e, we conclude $d \in C^{\mathcal{J}}$. ❏

We now investigate the size of filtrations. For this purpose, the *size* of an interpretation $\mathcal{I}$ is denoted by $|\mathcal{I}|$ and defined as the cardinality of $\Delta_{\mathcal{I}}$. Let C be an $\mathcal{ALCF}^{\sqcap}$-concept. We use $\mathsf{feat}(C)$ to denote the number of distinct features occurring in C (c.f. Section 3.1.3) and $\mathsf{ex}(D)$ to denote the number of distinct subconcepts of D that are of the form $\exists(R_1 \sqcap \cdots \sqcap R_n).E$ (c.f. Section 4.1.2).

Lemma 4.21. *Let $\mathcal{I}$ be a model of D and $\mathcal{J}$ a filtration of $\mathcal{I}$ w.r.t. D. Then $|\mathcal{J}| \leq (\mathsf{feat}(D) + \mathsf{ex}(D))^{\mathsf{rd}(D)+1} - 1$.*

Proof. The selection procedure consists of $\mathsf{rd}(D)$ rounds. Using induction on i, it is easily proved that, for each $i \leq \mathsf{rd}(D)$, at most $(\mathsf{feat}(D) + \mathsf{ex}(D))^i$ objects are selected in round i. This clearly implies the claim. ❏

We now have all necessary building blocks to prove the bounded model property.

Theorem 4.22 (Bounded Model Property). *If a concept name B is satisfiable w.r.t. a simple TBox $\mathcal{T}$, then there exists a model $\mathcal{I}$ of B and $\mathcal{T}$ such that $|\mathcal{I}| \leq 2^{|\mathcal{T}|^2}$.*

Proof. Assume that B is satisfiable w.r.t. $\mathcal{T}$ and let $\mathcal{I}$ be a model of B and $\mathcal{T}$. Moreover, let D be the result of unfolding B w.r.t. $\mathcal{T}$. By induction over the number of unfolding steps, it is straightforward to prove that $\mathcal{I}$ is a model of D. Let $\mathcal{J}$ be the filtration of $\mathcal{I}$ w.r.t. D with root d_0. By definition of simple TBoxes, it is easily seen that D is in NNF. Hence, by Lemma 4.20 and since $d_0 \in D^{\mathcal{I}}$, we have $d_0 \in D^{\mathcal{J}}$, i.e., $\mathcal{J}$ is a model of D. It is now easy to convert $\mathcal{J}$ into a model $\mathcal{J}'$ of B and $\mathcal{T}$: w.l.o.g. assume that $A^{\mathcal{J}} = \emptyset$ for all $A \in \mathsf{N_C}$ not appearing in D. Let $B_1 \doteq C_1, \ldots, B_k \doteq C_k$ be a total ordering of the concept definitions in $\mathcal{T}$ such that B_j does not "use" B_i in $\mathcal{T}$ (see Definition 2.5) for $1 \leq i < j \leq k$. Such an ordering obviously exists due to the definition of acyclic TBoxes. We define a sequence $\mathcal{J}_0, \ldots, \mathcal{J}_k$ of interpretations by setting $\mathcal{J}_0 := \mathcal{J}$ and, for $0 < i \leq k$, defining $\mathcal{J}_i$ as the interpretation obtained from $\mathcal{J}_{i-1}$ by setting

$$B_i^{\mathcal{J}_i} := B_i^{\mathcal{J}_{i-1}} \cup C_i^{\mathcal{J}_{i-1}}.$$

By induction on i, it is straightforward to prove that $\mathcal{J}_i$ is a model of the TBox $\{B_1 \doteq C_1, \ldots, B_i \doteq C_i\}$. Hence, $\mathcal{J}' := \mathcal{J}_k$ is a model of $\mathcal{T}$. By construction, $\mathcal{J}'$ is also a model of B.

Since clearly $|\mathcal{J}'| = |\mathcal{J}|$, it remains to show that $|\mathcal{J}| \leq 2^{|\mathcal{T}|^2}$ to prove the lemma. By Lemma 4.21, we have $|\mathcal{J}| \leq (\mathsf{feat}(D) + \mathsf{ex}(D))^{\mathsf{rd}(D)+1} - 1$. It is not hard to see that $\mathsf{feat}(D) \leq |\mathcal{T}|$ (in other words, $\mathsf{feat}()$ is polynomial under unfolding as defined in Section 4.1.2). Moreover, in Sections 4.1.1 and 4.1.2 we have shown that, if A is a concept name and $\widehat{\mathcal{T}}$ is an $\mathcal{ALC}$-TBox, then $\mathsf{ex}(\mathsf{unfold}(A, \widehat{\mathcal{T}})) \leq |\widehat{\mathcal{T}}|$ and $\mathsf{rd}(\mathsf{unfold}(A, \widehat{\mathcal{T}})) \leq |\widehat{\mathcal{T}}|$. Since these results do obviously generalize to $\mathcal{ALCF}^{\sqcap}$, we have $\mathsf{ex}(D) \leq |\mathcal{T}|$ and $\mathsf{rd}(D) \leq |\mathcal{T}|$. It follows that $|\mathcal{J}| \leq (2 \cdot |\mathcal{T}|)^{|\mathcal{T}|+1} - 1$ and aa function plotter reveals that $|\mathcal{J}| \leq 2^{|\mathcal{T}|^2}$ if $|\mathcal{T}| \geq 4$. Furthermore, if $|\mathcal{T}| < 4$, then $\mathcal{T}$ is of the form $\{A \doteq A'\}$ or $\{A \doteq \neg A'\}$ and clearly has a model of size 1. ❑

The bounded model property just established gives rise to a decision procedure for the satisfiability of $\mathcal{ALCF}^{\sqcap}$-concepts w.r.t. acyclic TBoxes.

Corollary 4.23. *$\mathcal{ALCF}^{\sqcap}$-concept satisfiability w.r.t. acyclic TBoxes is* NExpTime-*complete.*

Proof. The lower bound stems from Theorem 4.17. Hence we concentrate on the upper bound. As already argued, we may restrict ourselves to the satisfiability of concept names w.r.t. simple TBoxes. Hence, let B be a concept name and $\mathcal{T}$ be a simple TBox whose satisfiability is to be decided. A non-deterministic Turing machine may "guess" an interpretation of size at most $2^{|\mathcal{T}|^2}$, check whether it is a model of B and $\mathcal{T}$, and return satisfiable if this check succeeds and unsatisfiable otherwise. In the light of Theorem 4.22, the correctness of this procedure is easily seen. The Turing machine can be designed to run in non-deterministic exponential time due to the following observations:

1. In models $\mathcal{I}$ of B and $\mathcal{T}$, the interpretation of concept names $A \neq B$ not used in $\mathcal{T}$ and role names R not used in $\mathcal{T}$ is clearly irrelevant and can thus be assumed to be empty. Hence, we can represent any model $\mathcal{I}$ of B and $\mathcal{T}$ of size at most $2^{|\mathcal{T}|^2}$ by using a sequence of symbols of length $2^{p(|\mathcal{T}|)}$, where $p()$ is a polynomial.

2. As is easily seen, checking whether an interpretation $\mathcal{I}$ is a model of a concept name B and a simple TBox $\mathcal{T}$ can be done using model checking algorithms for First Order Logic: the interpretation $\mathcal{I}$ can naturally be viewed as a first order structure and the corresponding formula is

$$\varphi_{B,\mathcal{T}} := P_B(x) \wedge \bigwedge_{A \doteq C \in \mathcal{T}} \forall x (P_A(x) \leftrightarrow C^*(x)).$$

Here, $\cdot^*$ is the translation of $\mathcal{ALC}$-concepts into FO-formulas given in Section 2.1.1 extended with the following cases:

$$\begin{array}{lcl}
(\exists (R_1 \sqcap \cdots \sqcap R_k).C)^* & := & (\exists y (P_{R_1}(x,y) \wedge \cdots \wedge P_{R_k}(x,y) \wedge \exists x (x = y \wedge C^*))) \\
(\forall (R_1 \sqcap \cdots \sqcap R_k).C)^* & := & (\forall y ((P_{R_1}(x,y) \wedge \cdots \wedge P_{R_k}(x,y)) \rightarrow \forall x (x = y \rightarrow C^*))) \\
(f_1 \cdots f_k \downarrow f'_1 \cdots f'_{k'})^* & := & \exists x_1, \ldots, x_k, y_1, \ldots, y_k \big(P_{f_1}(x, x_1) \wedge P_{f'_1}(x, y_1) \\
 & & \wedge\, x_k = y_{k'} \wedge \bigwedge\limits_{1 \leq i < k} P_{f_i}(x_i, x_{i+1}) \wedge \bigwedge\limits_{1 \leq i < k'} P_{f'_i}(y_i, y_{i+1}) \big)
\end{array}$$

It is well-known that the combined space complexity of model checking in FO is $\mathcal{O}(|\varphi| \cdot \log(|\mathfrak{A}|))$, where φ is the input formula and $\mathfrak{A}$ the input structure [Vardi 1982]. Hence, the time complexity of this problem is $\mathcal{O}(2^{|\varphi|} \cdot |\mathfrak{A}|)$. Since the models to be checked have a representation of size at most $2^{p(|\mathcal{T}|)}$, and, obviously, the length of $\varphi_{B,\mathcal{T}}$ is linear in $|\mathcal{T}|$, the model checking performed by our algorithm can be done in time exponential in $|\mathcal{T}|$.

❑

Since subsumption w.r.t. acyclic TBoxes can be reduced to (un)satisfiability w.r.t. acyclic TBoxes, subsumption of $\mathcal{ALCF}^{\sqcap}$-concepts w.r.t. acyclic TBoxes is co-NExpTime-complete. We refrain from extending the above result to ABox consistency since $\mathcal{ALCF}$ is just a side-track in this thesis. However, we conjecture that $\mathcal{ALCF}^{\sqcap}$-ABox consistency is also NExpTime-complete and that this can be shown by a precompletion algorithm similar to the one presented in Section 3.2.

4.4 Discussion

We have developed a rule of thumb that characterizes a class of PSpace-complete Description Logics for which admitting acyclic TBoxes does not increase the complexity of reasoning. As we have seen, this class includes many "standard" DLs such as $\mathcal{ALC}$ and $\mathcal{ALCN}^{\sqcap}$. Moreover, we proved that there also exist standard DLs to which the rule of thumb is not applicable and the addition of acyclic TBoxes *does* significantly increase the complexity of reasoning: for the logic $\mathcal{ALCF}$, adding acyclic TBoxes raises the complexity from PSpace-complete to NExpTime-complete. In the following chapter, we shall see that $\mathcal{ALC}(\mathcal{D})$ is another example of a DL for which the addition of acyclic TBoxes does make a difference. In fact, we will see that, complexity-wise, extensions of $\mathcal{ALC}(\mathcal{D})$ "behave" very similar to extensions of $\mathcal{ALCF}$. Hence, feature (dis)agreements are not the only reason why the modification technique from Section 4.1.1 cannot be applied to the $\mathcal{ALCF}(\mathcal{D})$ completion algorithm presented in Chapter 3. Dropping feature (dis)agreement from $\mathcal{ALCF}(\mathcal{D})$ would not help in this respect.

As noted in Section 2.2.1, there exists a variant of acyclic TBoxes in which concept definitions have the form $A \sqsubseteq C$ and are satisfied by an interpretation $\mathcal{I}$ iff $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. In the following, we call such TBoxes *primitive*. It is not hard to see that acyclic TBoxes are stronger than primitive ones in the sense that primitive concept definitions $A \sqsubseteq C$ can be rephrased as $A \doteq C \sqcap A'$, where A' is a concept name not yet appearing in the TBox, but concept definitions $A \doteq C$ cannot be "simulated" by primitive TBoxes. There exist Description Logics for which the complexity of reasoning with primitive TBoxes does not coincide with the complexity of reasoning with acyclic TBoxes: for example, $\mathcal{TL}$-concept subsumption w.r.t. primitive TBoxes is in PTime [Calvanese 1996a] while $\mathcal{TL}$-concept subsumption w.r.t. acyclic TBoxes is co-NP-complete [Nebel 1990].[6] However, as is easily verified, all results obtained in this chapter do also apply to primitive TBoxes. This holds for the modification technique presented in

[6] Recall that $\mathcal{TL}$ is the Description Logic whose only constructors are conjunction and universal value restriction.

Section 4.1.1, which can in exactly the described form be used for primitive TBoxes, as well as for the $\mathcal{ALCF}$ lower bound and the $\mathcal{ALCF}^{\sqcap}$ upper bound from Section 4.2—for the upper bound this is even trivial since acyclic TBoxes can "simulate" primitive ones as argued above. Hence, it seems that, for Description Logics containing $\mathcal{ALC}$ as a fragment, the complexity of reasoning with acyclic TBoxes "usually" coincides with the the complexity of reasoning with primitive TBoxes.

Chapter 5

Extensions of $\mathcal{ALC}(\mathcal{D})$

In Sections 2.1.2 and 2.2.1, we have introduced several extensions of the basic Description Logic $\mathcal{ALC}$ that have been proposed since the expressive power of $\mathcal{ALC}$ itself is not sufficient for many applications. Although $\mathcal{ALC}(\mathcal{D})$ extends $\mathcal{ALC}$ by the capability of representing knowledge about "concrete qualities" of real-world entities, many of $\mathcal{ALC}$'s shortcomings are still present in $\mathcal{ALC}(\mathcal{D})$. For example, using $\mathcal{ALC}(\mathcal{D})$, we cannot talk about the inverses of roles or about the number of role successors a domain element may have. Moreover, until now we only considered $\mathcal{ALC}(\mathcal{D})$ without TBoxes and hence are lacking a means to represent terminological knowledge and background knowledge from application domains.

In this chapter, we extend $\mathcal{ALC}(\mathcal{D})$ with various means of expressivity and analyze the complexity of the resulting formalisms. More precisely, we consider the extension by (i) acyclic TBoxes, (ii) a role conjunction constructor, (iii) an inverse role constructor, (iv) generalized concrete domain constructors, and (v) the concrete domain role constructor (the latter two have been introduced in Section 2.3.2). Most of these extensions are seemingly "harmless" w.r.t. complexity: in Section 4.1, we saw that the extension of many PSpace Description Logics with acyclic TBoxes does not increase the complexity of reasoning. Moreover, it is well-known that the extension of a PSpace DL with role conjunction and inverse roles usually does not change the complexity of reasoning. For example, $\mathcal{ALC}$ extended with role conjunction is still in PSpace [Donini *et al.* 1997] as is $\mathcal{ALC}$ extended with inverse roles [Horrocks *et al.* 2000a; Spaan 1993b] or even with both [Tobies 2001b]. Surprisingly, we find that—despite their harmlessness in the context of $\mathcal{ALC}$—each of the above mentioned extensions of $\mathcal{ALC}(\mathcal{D})$ yields a logic for which reasoning is significantly harder than with $\mathcal{ALC}(\mathcal{D})$ itself.

More precisely, we will see that there exists a rather simple concrete domain W such that concept satisfiability in each of the five extensions of $\mathcal{ALC}(\mathsf{W})$ is NExpTime-hard. With "simple", we refer both to the nature of the predicates offered by W as well as to the complexity of W-satisfiability, which is in PTime. To demonstrate the relevance of the obtained hardness results, we show that they also apply to many other concrete domains proposed in the literature. As a corresponding upper bound, we prove that, if reasoning with a concrete domain $\mathcal{D}$ is in NP, then reasoning with $\mathcal{ALC}(\mathcal{D})$ and all five extensions (simultaneously) is in NExpTime. The established lower bounds show that the complexity of $\mathcal{ALC}(\mathcal{D})$ is rather sensitive w.r.t. extensions of the logic and hence the PSpace upper bound for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability established in Chapter 3 cannot be considered robust. However, it should be noted that there also exist interesting extensions of $\mathcal{ALC}(\mathcal{D})$ that (most probably) have no impact on the complexity of reasoning. We shall return to this issue in Section 5.6.

This chapter is organized as follows: we start with introducing a NExpTime-complete variant of the well-known undecidable Post Correspondence Problem [Post 1946]. Then, we introduce the concrete domain W that is designed for encoding the NExpTime-complete PCP in Description Logics with concrete domains and show that W-satisfiability is in PTime. Using this concrete domain, we establish the five lower bounds and then prove the corresponding upper bound by devising a completion algorithm. Finally, we compare the extensions of $\mathcal{ALC}(\mathcal{D})$ with the corresponding extensions of $\mathcal{ALCF}$, obtaining the result that, in most cases, the complexity of (extensions of) $\mathcal{ALCF}$ parallels the complexity of (extensions of) $\mathcal{ALC}(\mathcal{D})$.

5.1 A NExpTime-complete Variant of the PCP

Post's Correspondence Problem, which was introduced by Emil Post as early as 1946 [Post 1946], is a valuable tool for undecidability proofs. In the following, we tailor a NExpTime-complete variant of Post's original undecidable problem to allow for convenient NExpTime-hardness proofs in subsequent sections.

Definition 5.1 (PCP). An *instance* P of the *Post Correspondence Problem* is given by a finite, non-empty list $(\ell_1, r_1), \ldots, (\ell_k, r_k)$ of pairs of words over some alphabet Σ. A sequence of integers $i_1, \ldots, i_m$, with $m \geq 1$, is called a *solution* to P iff

$$\ell_{i_1} \cdots \ell_{i_m} = r_{i_1} \cdots r_{i_m}.$$

The *Post Correspondence Problem (PCP)* is to decide, for a given instance P, whether P has a solution. Let $f(n)$ be a mapping from $\mathbb{N}$ to $\mathbb{N}$ and let $|P|$ denote the sum of the lengths of all words in the PCP P, i.e.,

$$|P| = \sum_{1 \leq i \leq k} |\ell_i| + |r_i|.$$

A solution $i_1, \ldots, i_m$ to a PCP-instance P is called $f(n)$*-solution* iff $m \leq f(|P|)$. By $f(n)$*-PCP*, we denote the version of the PCP that admits only $f(n)$-solutions. $\diamond$

Undecidability of the general PCP was first proved in [Post 1946] and later reproved by Hopcroft and Ullman in [1979]. In [Garey & Johnson 1979], a variant of the PCP is listed as an NP-complete problem (problem number [SR11]). In this variant, a PCP-instance is given by a finite list of word pairs $(\ell_1, r_1), \ldots, (\ell_k, r_k)$ and a positive integer $K \leq k$ in unary coding. As solutions, only sequences of length at most K are admitted. Inspired by this result, we prove NExpTime-completeness of the $2^n + 1$-PCP. For proving NExpTime-hardness, which is much more difficult than establishing the upper bound, we first introduce another variant of the PCP.

Definition 5.2 (MPCP). Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP-instance. A solution $i_1, \ldots, i_m$ to P is called an *MPCP-solution* iff $i_1 = 1$. By *MPCP*, we denote the version of the PCP that admits only MPCP-solutions. ◇

For functions $f(n)$ from $\mathbb{N}$ to $\mathbb{N}$, we define *$f(n)$-MPCP-solutions* and the *$f(n)$-MPCP* in the obvious way. In the following, we will sometimes call a PCP-instance P an *MPCP-instance* if we are only interested in MPCP solutions. The next lemma shows that, in order to prove NExpTime-hardness of the $2^n + 1$-PCP, it suffices to prove NExpTime-hardness of certain MPCPs.

Lemma 5.3. *Let $p(n)$ be a polynomial such that $p(n) \geq 8n$ for all $n \in \mathbb{N}$. If the $2^{p(n)}$-MPCP is* NExpTime*-hard, then the $2^n + 1$-PCP is* NExpTime*-hard.*

Proof. We reduce the $2^{p(n)}$-MPCP to the $2^n + 1$-PCP. The reduction is largely identical to Hopcroft and Ullman's reduction of the general MPCP to the general PCP [Hopcroft & Ullman 1979]. However, we need to do some additional work to deal with the lengths of solutions.

Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be an MPCP-instance with $|P| = s$ and let ¢, \$, and $\sharp$ be symbols not occurring in P. Our goal is to define a corresponding PCP-instance P' such that P has a $2^{p(n)}$-MPCP-solution iff P' has a $2^n + 1$-solution. W.l.o.g., we assume that P contains no pairs of the form (ϵ, ϵ). Moreover, we can safely assume $s > 1$ since otherwise P has no solution which means that we can set $P' := P$ and are done.

Define a PCP-instance $P' = (\ell'_0, r'_0), \ldots, (\ell'_{k+2}, r'_{k+2})$ as follows:

1. for $1 \leq i \leq k$, ℓ'_i is obtained from ℓ_i by inserting the symbol ¢ after each symbol in ℓ_i, and r'_i is obtained from r_i by inserting the symbol ¢ ahead of each symbol in r_i;
2. $(\ell'_0, r'_0) := (¢\ell'_1, r'_1)$ and $(\ell'_{k+1}, r'_{k+1}) := (\$, ¢\$)$;
3. $\ell'_{k+2} = \epsilon$ and $r'_{k+2} = \sharp^x$ where $x = p(s) - (2s + |\ell'_1| + |r'_1| + 4)$.

Note that $x \geq 0$: since $|\ell'_1| \leq 2s$, $|r'_1| \leq 2s$, and $4 \leq 2s$ (recall that $s > 1$), we have

$$(2s + |\ell'_1| + |r'_1| + 4) \leq 8s.$$

Moreover, we require $p(s) \geq 8s$ and hence $x \geq 0$. It is not hard to check that $|P'| = p(s)$: the size of the list $(\ell'_1, r'_1), \ldots, (\ell'_k, r'_k)$ is $2s$, the size of (ℓ'_0, r'_0) is $|\ell'_1| + |r'_1| + 1$, the size of (ℓ'_{k+1}, r'_{k+1}) is 3, and the size of (ℓ'_{k+2}, r'_{k+2}) is x. Hence, by definition of x, we have $|P'| = p(s)$.

We claim that every $2^{p(n)}$-MPCP solution to P can be translated into a $2^n + 1$-solution to P' and vice versa. Intuitively, the ¢ and \$ symbols are introduced to ensure that (i) (ℓ'_0, r'_0) is the only pair in which the left and the right word start with the same symbol and (ii) MPCP-solutions to P can be translated into solutions to P' and vice versa (this part of the reduction is identical to Hopcroft and Ullman's). The purpose of the $\sharp$ symbol and the pair (ℓ'_{k+2}, r'_{k+2}) is to "blow up" the size of P' which is necessary to ensure that solutions have correct lengths. Note that the index $k + 2$

can appear in solutions to P' only if $x = 0$. In this case, however, it can es well be left away since the corresponding pair degenerates to (ϵ, ϵ).

Assume that $1, i_1, \ldots, i_r$ is a $2^{p(n)}$-MPCP-solution to P (i.e., $r < 2^{p(s)}$). Using the definition of P', it is readily checked that this implies that $0, i_1, \ldots, i_r, k+1$ is a solution to P' (c.f. [Hopcroft & Ullman 1979] where more details and an example are presented). Since $|P'| = p(s)$, it is readily checked that $0, i_1, \ldots, i_r, k+1$ is even a $2^n + 1$-solution to P'.

Now assume that $i_1, \ldots, i_r$ is a $2^n + 1$-solution to P'. We assume w.l.o.g. that $i_j \neq k+2$ for $1 \leq j \leq r$ (see above). We have $i_1 = 0$ due to the use of the ¢ symbol. Moreover, it is readily checked that $i_1, \ldots, i_j$ is a solution to P, where j is the smallest integer such that $i_{j+1} = k+1$. Such a j always exists due to the use of the ¢ and \$ symbols (see [Hopcroft & Ullman 1979]). Since $|P'| = p(s)$ and (obviously) $j < r$, it is readily checked that $i_1, \ldots, i_j$ is a $2^{p(n)}$-MPCP-solution to P. ❑

In the following, we show that there in fact exists a polynomial $p(n)$ with $p(n) \geq 8n$ for all $n \in \mathbb{N}$ such that the $2^{p(n)}$-MPCP is NExpTime-hard. This is done using a reduction of the acceptance problem of Turing machines.

Definition 5.4 (Turing machine). A *non-deterministic Turing machine* is given by a tuple $M := (Q, \Gamma, \delta, q_0, Q_f)$ where

- Q is a finite set of *states*,
- Γ is a finite set of *symbols* with $\Gamma \cap Q = \emptyset$, which contains the special symbol B called the *blank symbol*,
- δ is a *transition function* which maps $Q \times \Gamma$ to the power set of $Q \times \Gamma \times \{\mathsf{left}, \mathsf{right}, \mathsf{stay}\}$,
- $q_0 \in Q$ is the *initial state*, and
- $Q_f \subseteq Q$ is a set of *final states.*

Let $M = (Q, \Gamma, \delta, q_0, Q_f)$ be a (non-deterministic) Turing machine. An *ID* uqv of M is a word in $\Gamma^* Q \Gamma^*$. Such an ID has the usual interpretation, i.e., it describes the inscription of the infinite tape (all tape cells "before u" and "behind v" are labeled with B), the current state q, and the head position of M, which is on the leftmost symbol of v.
The usual *transition relation* on IDs is denoted by $\vdash$. Intuitively. $uqv \vdash u'q'v'$ if a single step of M in ID uqv may result in ID $u'q'v'$. An exact definition is omitted and can be found in any book on recursion theory, e.g. [Hopcroft & Ullman 1979]. By $\vdash^*$, we denote the reflexive transitive closure of $\vdash$.
M accepts an input w (given as an initial tape inscription) iff there exists a $q_f \in Q_f$ such that $q_0 w \vdash^* u q_f v$ for some $u, v \in \Gamma^*$. Let f be a function from $\mathbb{N}$ to $\mathbb{N}$. M is an *$f(n)$-Turing machine* iff, for each input w, every sequence

$$q_0 w \vdash u_1 q_1 v_1 \vdash u_2 q_2 v_2 \vdash \cdots$$

is of length at most $f(|w|)$. ◇

Group I

Left word	Right word	
X	X	for each $X \in \Gamma$
$\sharp$	$\sharp$	

Group II. For each $q \in Q \setminus Q_f$, $p \in Q$, and $X, Y, Z \in \Gamma$:

Left word	Right word	
qX	Yp	if $\delta(q, X) = (p, Y, R)$
ZqX	pZY	if $\delta(q, X) = (p, Y, L)$
$q\sharp$	$Yp\sharp$	if $\delta(q, B) = (p, Y, R)$
$Zq\sharp$	$pZY\sharp$	if $\delta(q, B) = (p, Y, L)$

Group III. For each $q \in Q_f$ and $X, Y \in \Gamma$:

Left word	Right word
XqY	q
Xq	q
qY	q

Group IV.

Left word	Right word	
$q\sharp\sharp$	$\sharp$	for each $q \in Q_f$

Figure 5.1: The MPCP translation.

We now give a translation of Turing machines and their inputs to MPCP-instances, which is crucial for proving the central result of this section. The translation is identical to the one used by Hopcroft and Ullman to prove undecidability of the general PCP [Hopcroft & Ullman 1979]. We repeat it for the sake of completeness. Let $M = (Q, \Gamma, \delta, q_0, Q_f)$ be a Turing machine and w an input for M. We define a corresponding MPCP-instance $P_w^M = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ as follows. The first pair (ℓ_1, r_1) is defined as

$$\ell_1 := \sharp \qquad r_1 := \sharp q_0 w \sharp.$$

The set of remaining pairs (ℓ_i, r_i) is partitioned into 4 groups and can be found in Figure 5.1. Intuitively, if P_w^M has an MPCP-solution $I = i_1, \ldots, i_r$, then the word $w_I = \ell_{i_1} \cdots \ell_{i_r} = r_{i_1} \cdots r_{i_r}$ starts with $\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_n q_n v_n$, where subwords between successive $\sharp$'s are successive IDs in a computation of M with input w and q_n is a final state.

We now fix a Turing machine M and a word w and establish some properties of the MPCP-instance P_w^M. We call a pair of words (x, y) a *partial solution* iff x is a prefix of y and there exists a sequence of integers $i_1, \ldots, i_m$ such that $x = \ell_{i_1} \cdots \ell_{i_m}$ and $y = r_{i_1} \cdots r_{i_m}$. Hopcroft and Ullman prove the following lemma:

Lemma 5.5. *If there exists a sequence of IDs* $q_0w \vdash\!\!\!- u_1q_1v_1 \vdash\!\!\!- \cdots \vdash\!\!\!- u_nq_nv_n$*, then there exists a partial solution*

$$\begin{aligned}(x,y) = (&\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp,\\ &\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp).\end{aligned}$$

The proof is by induction on n where the induction step straightforwardly translates Turing machine transitions into sequences of word pairs as given in Figure 5.1. We now show that the existence of a partial solution describing an accepting computation of M on w implies the existence of a (partial) solution (x, y) such that $x = y$ and certain length bounds are satisfied.

Lemma 5.6. *If there exists a partial solution*

$$\begin{aligned}(x,y) = (&\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp,\\ &\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp),\end{aligned}$$

with $q_n \in Q_f$*, and* $|u_iv_i| \leq n + |w|$ *for* $1 \leq i \leq n$*, then there exists a (partial) solution*

$$\begin{aligned}(x',y') = (&\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp\cdots\sharp u_rq_rv_r\sharp\sharp,\\ &\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp\cdots\sharp u_rq_rv_r\sharp\sharp)\end{aligned}$$

with $r \leq 2n + |w|$ *and* $|u_iv_i| \leq n + |w|$ *for all* $1 \leq i \leq r$.

Proof. If $|u_nv_n| = 0$, it remains to concatenate a single pair from Group IV (we then have $n = r$ in (x', y')). Hence assume $|u_nv_n| > 0$. It is not hard to see that each partial solution of the form (x, y) can be extended to a partial solution

$$\begin{aligned}(&\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp,\\ &\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp u_{n+1}q_nv_{n+1}\sharp)\end{aligned}$$

where $|u_{n+1}| < |u_n|$ or $|v_{n+1}| < |v_n|$: just use (at most) $|u_n| + |v_n|$ concatenations of pairs from Group I and a single concatenation of a pair from Group III. Repeating this extension step, we will finally arrive at a partial solution

$$\begin{aligned}(x'',y'') = (&\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp\cdots\sharp u_{r-1}q_nv_{r-1}\sharp,\\ &\sharp q_0w\sharp u_1q_1v_1\sharp\cdots\sharp u_{n-1}q_{n-1}v_{n-1}\sharp u_nq_nv_n\sharp\cdots\sharp u_{r-1}q_nv_{r-1}\sharp q_n\sharp)\end{aligned}$$

where $r \leq n + |u_nv_n|$. Since $|u_nv_n| \leq n + |w|$, we have $r \leq 2n + |w|$. By construction of (x'', y''), we have $|u_iv_i| \leq n + |w|$ for all $1 \leq i < r$. A single concatenation of a pair from Group IV yields the desired solution (x', y'). ❏

We may now establish the lower bound.

Proposition 5.7. *There exists a polynomial* $p(n)$ *with* $p(n) \geq 8n$ *for all* $n \in \mathbb{N}$ *such that the* $2^{p(n)}$*-MPCP is* NExpTime*-hard.*

Proof. Let M be a 2^{n^d}-Turing machine (for some integer $d \geq 1$) deciding an arbitrary NEXPTIME-hard language. W.l.o.g., we assume that M makes at least $|w|$ steps on w before stopping. The reason for this will become clear later. We show that

$$M \text{ accepts } w \text{ iff } P_w^M \text{ has a } 2^{8n^d}\text{-MPCP-solution,} \tag{$*$}$$

i.e., we reduce the NEXPTIME-hard problem solved by the Turing machine M to the 2^{8n^d}-MPCP. It is easy to check that the polynomial $8n^d$ is as required.

First for the "only if" direction. Let w be an input to M and assume that M accepts w in n steps, where $n \leq 2^{|w|^d}$. Then there exists a sequence of IDs

$$q_0 w \longmapsto u_1 q_1 v_1 \longmapsto \cdots \longmapsto u_n q_n v_n$$

such that $q_n \in Q_f$. By Lemma 5.5, this implies the existence of a partial solution

$$\begin{aligned}(x, y) = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp, \\ &\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp),\end{aligned}$$

to P_w^M. Since a Turing machine writes at most one symbol per step, we obviously have $|u_i v_i| \leq n + |w|$ for $1 \leq i \leq n$. By Lemma 5.6, there exists a solution $I = i_1, \ldots, i_m$ to P_w^M corresponding to a word

$$w_I = \ell_{i_1} \cdots \ell_{i_m} = r_{i_1} \cdots r_{i_m} = \sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_r q_r v_r \sharp\sharp$$

with $r \leq 2n + |w|$ and $|u_i v_i| \leq n + |w|$ for all $1 \leq i \leq r$. Since, by assumption, M makes at least $|w|$ steps if started on w, it follows that $r \leq 3n$ and $|u_i v_i| \leq 2n$ for all $1 \leq i \leq r$. We need an estimation for the length m of the solution $i_1, \ldots, i_m$. Obviously, we have $m \leq r \cdot (2n + 2) + 2$ since m is clearly bounded by the number of symbols in w_I, and the length of each subword of w_I of the form $\sharp u_1 q_i v_i$ is bounded by $2n + 2$. It follows that $m \leq 6n^2 + 6n + 2$ and hence also $m \leq n^5 + 13$. Together with $n \leq 2^{|w|^d}$, this implies $m \leq 2^{5 \cdot |w|^d} + 13$, and, since $|w| \leq |P_w^M|$ by definition of P_w^M, we have $m \leq 2^{5 \cdot |P_w^M|^d} + 13$. Since $|P_w^M| \geq 2$ (also by definition), we have $2^{5 \cdot |P_w^M|^d} + 13 \leq 2^{8 \cdot |P_w^M|^d}$.

Now for the "if" direction. We show the contrapositive. Hence, assume that M does not accept w, i.e., no computation of M on w reaches a final state. Assume to the contrary of what is to be shown that P_w^M has a 2^{8n^d}-solution. Following Hopcroft and Ullmann, we claim that, if P_w^M has a solution, then it has a solution $I = i_1, \ldots, i_m$ such that $w_I = \sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_k q_k v_k \sharp$ where strings between successive $\sharp' s$ are successive IDs in a computation of M with input w, and q_k is a final state. This implies that M accepts w (the length of the above mentioned solution I and the corresponding TM computation is irrelevant since we know that *every* computation of M has length at most $2^{|w|^d}$). Hence, the existence of a 2^{8n^d}-solution to P_w^M is a contradiction to the assumption that M does not accept w. ❑

The main result of this section is now easily obtained.

Theorem 5.8. *It is* NEXPTIME*-complete to decide whether a* $2^n + 1$*-PCP has a solution.*

Proof. NExpTime-hardness is an immediate consequence of Proposition 5.7 and Lemma 5.3. To decide whether a PCP-instance has a $2^n + 1$-solution, a non-deterministic Turing machine may simply "guess" such a solution and then check its validity. Since it is not hard to see that this can be done using a 2^{n^d}-Turing machine (for some constant d), the $2^n + 1$-PCP is in NExpTime. ❏

5.2 A Concrete Domain for Encoding the PCP

In this section, we introduce a concrete domain W that will allow to reduce the $2^n + 1$-PCP to concept satisfiability in Description Logics providing for this concrete domain. More specifically, W is based on the set of words over some sufficiently large alphabet Σ and a set of rather simple predicates that, most importantly, allow to express the concatenation of words. Here, "sufficiently large" means that 2^n+1-PCP's based on Σ must be capable of encoding the computations of some NExpTime-hard Turing machine (see the proof of Proposition 5.7). In the following, we assume that an appropriate Σ is used without specifying it explicitly.

Definition 5.9 (Concrete Domain W). Let Σ be an alphabet. The concrete domain W is defined by setting $\Delta_\mathsf{W} := \Sigma^*$ and defining Φ_W as the smallest set containing the following predicates:

- unary predicates word and nword with $\mathsf{word}^\mathsf{W} = \Delta_\mathsf{W}$ and $\mathsf{nword}^\mathsf{W} = \emptyset$,
- unary predicates $=_\epsilon$ and $\neq_\epsilon$ with $=_\epsilon^\mathsf{W} = \{\epsilon\}$ and $\neq_\epsilon^\mathsf{W} = \Sigma^+$,
- a binary equality predicate $=$ and a binary inequality predicate $\neq$ with the obvious interpretation, and
- for each $w \in \Sigma^+$, two binary predicates conc_w and nconc_w with

$$\mathsf{conc}_w^\mathsf{W} = \{(u, v) \mid v = uw\} \text{ and } \mathsf{nconc}_w^\mathsf{W} = \{(u, v) \mid v \neq uw\}.$$

◇

Note that Φ_W is closed under negation. To show that W is admissible, it hence remains to prove that the satisfiability of finite predicate conjunctions (see Definition 2.8) is decidable. We do this by developing an appropriate algorithm.

The algorithm works on predicate conjunctions that contain only the predicates nword, $=_\epsilon$, $\neq$, and conc_w (for any $w \in \Delta_\mathsf{W}$). Such predicate conjunctions are said to be in *normal form (NF)*. We show that assuming predicate conjunctions to be in NF does not sacrifice generality: by applying the following normalization steps, every predicate conjunction c can be converted into a predicate conjunction c' that is in NF and satisfiable iff c is satisfiable.

1. Eliminate all occurrences of the word predicate from c and call the result c_1.
2. Let x be a variable not appearing in c_1. Augment c_1 by the conjunct $=_\epsilon(x)$ and then replace every occurrence of $\neq_\epsilon(y)$ in c_1 with $\neq(x, y)$ calling the result c_2.

3. Let $\beta_1, \ldots, \beta_k$ be all conjuncts in c_2 which are of the form $\mathsf{nconc}_w(x, y)$ and let $z_1, \ldots, z_k$ be variables not appearing in c_2. For each i with $1 \leq i \leq k$ and $\beta_i = \mathsf{nconc}_w(x, y)$, augment c_2 by the conjuncts $\mathsf{conc}_w(x, z_i)$ and $\neq(z_i, y)$. Then delete the conjunct β_i from c_2. Call the result c_3.

4. Remove occurrences of the $=$ predicate from c_3 by "filtration": Let $\sim$ be the equivalence relation induced by occurrences of the $=$ predicate in c_3 and use $[x]_\sim$ to denote the equivalence class of $\sim$ containing x. For each equivalence class $\mathcal{C}$ of $\sim$, fix a variable $z_\mathcal{C}$ such that $z_\mathcal{C} \neq z_{\mathcal{C}'}$ if $\mathcal{C} \neq \mathcal{C}'$. Now substitute, for each variable x occurring in c_3, every occurrence of x in c_3 by $z_{[x]_\sim}$. Then delete all occurrences of the $=$ predicate from c_3. The result of this step is the normal form c' of c.

Obviously, the normalization process preserves (un)satisfiability, i.e., a predicate conjunction c is satisfiable iff its normal form c' is satisfiable. The translation can be performed in polynomial time and the size of the resulting predicate conjunction c' is linear in the size of c.

Before the algorithm itself is given, we introduce some notions. Let c be a predicate conjunction (not necessarily in normal form). By $V(c)$, we denote the set of variables used in c. The conc-*graph* $G(c) = (V, E)$ of c is the directed graph described by occurrences of conc_w predicates in c, i.e., $V = V(c)$ and $(x, y) \in E$ iff $\mathsf{conc}_w(x, y)$ is a conjunct of c for some word w. A conjunction c is said to have a conc-cycle if $G(c)$ has a cycle. The *distance* $\mathsf{dist}(v, v')$ of two variables $v, v' \in V$ in c is 0 if $v = v'$ and the length of the *longest* path leading from v to v' in $G(c)$ otherwise. The *level* $\mathsf{lev}(v)$ of v in c is defined as

$$\mathsf{lev}(v) := \max\{k \mid \mathsf{dist}(v, v') = k \text{ and } v' \text{ is a sink}\}$$

where a sink is a node which has no outgoing edges. Note that there exist conc-graphs $G(c)$ with nodes v such that $\mathsf{lev}(v)$ is undefined since there exists no path from v to a sink. However, we will only use $\mathsf{lev}()$ for conc-graphs that contain no cycle. In this case, $\mathsf{lev}(v)$ is obviously always defined. Let $w, w' \in \Sigma^+$. The function pre is defined as follows:

$$\mathsf{pre}(w, w') = \begin{cases} v & \text{if } w = vw' \text{ with } v \neq \epsilon \\ \text{undefined} & \text{if no such } v \text{ exists} \end{cases}$$

The algorithm for deciding the satisfiability of predicate conjunctions in normal form can be found in Figure 5.2. Note that the parameter c to norm is passed "by reference", i.e., changes made to c in the norm function are also effective in the calling procedure. Before the correctness of the algorithm is proved formally, we explain the underlying intuition. Assume that the satisfiability of a conjunction is to be decided. The algorithm repeatedly performs several normalization steps and inconsistency checks. In Figure 5.2, normalizations are annotated with Ni while inconsistency checks are marked with Ci. All normalizations performed by the algorithm preserve (un)satisfiability of predicate conjunctions. If an inconsistency check succeeds, the algorithm may thus safely return $\mathsf{unsatisfiable}$. If no inconsistencies are found, the

define procedure sat-W(c)
  if c contains the nword predicate or norm(c) = unsatisfiable **then** (C1)
    return unsatisfiable
  for i = 0 **to** $|V(c)|$ **do**
    while there exist $x, y, y' \in V(c)$ with $\mathsf{lev}(x) = i$
        and $w, w' \in \Sigma^+$ with $w \neq w'$ such that
        $\mathsf{conc}_w(y, x)$ and $\mathsf{conc}_{w'}(y', x)$ are in c **do**
      // since norm was just applied, we have $y \neq y'$. (C2)
      if neither w is a suffix of w' nor vice versa **then**
        return unsatisfiable
      w.l.o.g., assume that w' is a suffix of w. (N1)
      // since norm was just applied, we have $w = vw'$ for a $v \neq \epsilon$.
      replace $\mathsf{conc}_w(y, x)$ by $\mathsf{conc}_{\mathsf{pre}(w,w')}(y, y')$ in c
      if norm(c) = unsatisfiable **then** (C3)
        return unsatisfiable
  if there exist $x, y \in V(c)$ and a $w \in \Sigma^+$ such that (C4)
      $\mathsf{conc}_w(y, x)$ and $=_\epsilon(x)$ are in c **then**
    return unsatisfiable
  if there are $x_0, \ldots, x_k, y_0, \ldots, y_\ell \in V(c)$ (C5)
      and $w_0, \ldots, w_{k-1}, v_0, \ldots, v_{\ell-1} \in \Sigma^+$ such that
      (i) $=_\epsilon (x_0)$, $=_\epsilon (y_0)$ are in c or $x_0 = y_0$,
      (ii) $\mathsf{conc}_{w_i}(x_i, x_{i+1})$ in c for $i < k$ and $\mathsf{conc}_{v_i}(y_i, y_{i+1})$ in c for $i < \ell$,
      (iii) $w_0 \cdots w_{k-1} = v_0 \cdots v_{\ell-1}$, and
      (iv) $\neq(x_k, y_k)$ is in c **then**
    return unsatisfiable
  return satisfiable

define procedure norm(c) // c is passed "by reference" (see text)
  while there exist $x, y, y' \in V(c)$ with $y \neq y'$ and a $w \in \Sigma^+$ such that (N2)
      $\mathsf{conc}_w(y, x)$ and $\mathsf{conc}_w(y', x)$ are in c **do**
    replace every occurrence of y' in c by y
  if c contains a conc-cycle **then** (C6)
    return unsatisfiable
  if there exist $x, y \in V(c)$ and $w, w' \in \Sigma^+$ with $w \neq w'$ such that (C7)
      $\mathsf{conc}_w(y, x)$ and $\mathsf{conc}_{w'}(y, x)$ are in c **then**
    return unsatisfiable
  return satisfiable

Figure 5.2: The W satisfiability algorithm.

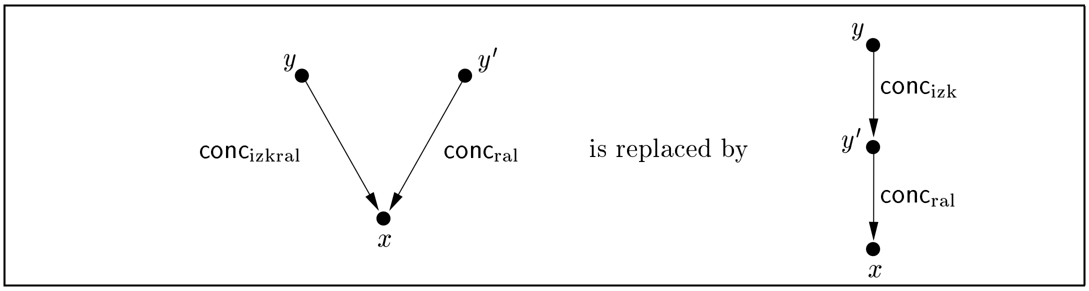

Figure 5.3: An example fork elimination.

algorithm ends up with a predicate conjunction that is satisfiable. This clearly implies satisfiability of the original predicate conjunction and thus the algorithm returns satisfiable.

All of the normalizations and most of the inconsistency checks are concerned with situations of the form

$$\mathsf{conc}_w(y, x), \mathsf{conc}_{w'}(y', x)$$

where several cases can be distinguished:

a) $y \neq y'$ and neither w is a suffix of w' nor vice versa. In this case, c is unsatisfiable (C2).

b) $y = y'$ and $w \neq w'$. In this case, c is unsatisfiable (C7).

c) $y \neq y'$ and w' is a true suffix of w.[1] In this case, we call the predicate expressions $\mathsf{conc}_w(y, x), \mathsf{conc}_{w'}(y', x)$ a *fork*. We may eliminate the fork by replacing $\mathsf{conc}_w(y, x)$ with $\mathsf{conc}_{\mathsf{pre}(w,w')}(y, y')$. An example for fork elimination can be found in Figure 5.3. It is easily checked that this operation preserves (un)satisfiability (N1).

d) $y \neq y'$ and $w = w'$. In this case, y and y' describe the same word and can be identified preserving (un)satisfiability (N2).

If a predicate conjunction does neither contain a conc-cycle nor any of the above situations, then its conc-graph has the form of a forest. In the correctness proof, this fact will play a crucial role: together with some additional conditions, it ensures that we can find a solution to the predicate conjunction if the algorithm returns satisfiable. We now formally prove correctness and termination of the algorithm. For a predicate conjunction c, let $|c|$ denote the number of conjuncts in c.

Lemma 5.10 (Correctness and Termination). *Let c_0 be an input to* sat-W. *The algorithm terminates after $\mathcal{O}(|c_0|^k)$ steps (where $k \in \mathbb{N}$ is constant) returning* satisfiable *if c_0 has a solution and* unsatisfiable *otherwise.*

[1]The case that w is a true suffix of w' is not listed explicitly since it is symmetric to Case c).

Proof. We first prove that c_0 has a solution if sat-W returns satisfiable and c_0 has no solution if sat-W returns unsatisfiable. Since it is readily checked that the normalization steps N1 and N2 performed by the algorithm preserve (un)satisfiability, it suffices to show that

(i) if sat-W returns satisfiable, then the predicate conjunction c constructed by normalization has a solution, and

(ii) if sat-W returns unsatisfiable, then the predicate conjunction c constructed by normalization has no solution.

Point (ii) can be proved straightforwardly by examining the inconsistency checks C1 to C7 which cause the algorithm to return unsatisfiable: either c contains the nword predicate, the situations a) or b) described above occur, c contains a conc-cycle, or one of the last two **if** clauses from the main procedure applies. Obviously, in all cases the constructed predicate conjunction c has no solution.

Now for Point (i). We first prove that, if the algorithm returns satisfiable, then the constructed predicate conjunction c satisfies the following conditions:

1. c does not contain the nword predicate and no conc-cycle,
2. c contains no situations of the form $\mathsf{conc}_w(y,x), \mathsf{conc}_{w'}(y',x)$ with $y \neq y'$, and
3. if $\mathsf{conc}_w(y,x), \mathsf{conc}_{w'}(y,x)$ are in c, then $w = w'$.

It is easily seen that Property 1 holds: firstly, occurrences of the nword predicate are checked in C1 and this predicate is not (re)introduced later; secondly, conc-cycles are checked for in C3/C6, i.e., after the last normalization step. Similarly, Property 3 obviously holds since it is checked for by C3/C7, also after the last normalization step. In the following, we show that Property 2 holds.

Obviously, if Property 2 fails, then c contains one of the situations a), c), or d). Normalization C3/N2 deals with situation d). Since N2 is applied after the last N1 application, c does not contain situation d). Hence, let us concentrate on a) and c). The **for** loop in the main procedure iterates from 0 to $|V(c)|$. In each iteration step, inconsistency check C2 deals with situation a) and normalization N1 with c) (i.e., with forks). In both cases, however, predicates

$$\mathsf{conc}_w(y,x), \mathsf{conc}_{w'}(y',x)$$

are considered only if $\mathsf{lev}(x) = i$. This is sufficient to ensure that c does not contain situations a) and c) due to the following facts:

- We consider only conc-graphs $G(c)$ that are cycle-free (otherwise we return unsatisfiable due to C3/C6). Hence, the maximum level of nodes is $|V(c)|$.
- The elimination of forks in N1 and the elimination of situations d) in C3/N2 may create new forks
$$\mathsf{conc}_{\tilde{w}}(\tilde{y},\tilde{x}), \mathsf{conc}_{\tilde{w}'}(\tilde{y}',\tilde{x}).$$

However, it is straightforward to check that (in both cases) $\mathsf{lev}(\tilde{x}) > \mathsf{lev}(x)$ and hence these newly generated forks will be eliminated during a later step of the **for** loop.

We now return to the proof of Point (i): using Properties 1 to 3, we show that the constructed conjunction c has a solution δ if the algorithm returns satisfiable. Properties 1 and 2 imply that the conc-graph $G(c) = (V, E)$ of c is a forest. We inductively define a solution δ for c. Our strategy is to start with defining $\delta(v)$ for the variables v which are roots of trees in the forest $G(c)$. Since the edges in the trees correspond to conc_w-predicates, our choice of $\delta(v)$ for the root v of a tree determines $\delta(v')$ for all remaining nodes v' in the same tree. We must, however, carefully choose $\delta(v)$ for the roots v of the trees to guarantee that all $=_\epsilon$ and $\neq$ predicates in c are satisfied. Let t be the number of trees in $G(c)$ and let $w_1, \dots, w_t$ be words from Σ^+ such that

$$|w_{i+1}| - |w_i| \geq |V| \cdot \max\{|w| \mid \mathsf{conc}_w \text{ is used in } c\} \text{ for } 1 \leq i < t.$$

For the induction start, fix an ordering on the trees in $G(c)$ and let $x_1, \dots, x_t \in V$ be such that x_i is the root of the i-th tree in $G(c)$. For all $1 \leq i \leq t$, set

- $\delta(x_i) = \epsilon$ if $=_\epsilon(x_i)$ is in c and
- $\delta(x_i) = w_i$ otherwise.

For the induction step, if x is a node with $\delta(x) = w$ and $\mathsf{conc}_{w'}(x, y)$ is in c, then set $\delta(y) = ww'$. δ is well-defined since $G(c)$ is a forest and Property 3 from above holds. Obviously, δ satisfies all conc_w predicates in c. Non-applicability of inconsistency check C4 ensures that, if $=_\epsilon(x)$ is in c, then x is the root of a tree and hence δ also satisfies all $=_\epsilon(x)$ predicates in c. Now for $\neq(x, y)$ predicates. We make a case distinction:

- x and y are in the same tree. By definition of δ and since the last C5 inconsistency check did not apply, $\neq(x, y)$ is satisfied.
- x and y are in different trees, and each tree has a root z with $\delta(z) = \epsilon$, i.e., $=_\epsilon(z)$ is in c. Identical to the above case.
- x and y are in different trees and at least one of the trees has a root z with $\delta(z) \neq \epsilon$. Let z and z' be the roots of the two trees. By definition of δ, we have

  $$\text{abs}(|\delta(z)| - |\delta(z')|) \geq |V| \cdot \max\{|w| \mid \mathsf{conc}_w \text{ is used in } c\}$$

  where abs(x) denotes the absolute value of x. This clearly implies that, for any two nodes x' and y', where x' is in the tree with root z and y' is in the tree with root z', we have $\delta(x') \neq \delta(y')$.

This completes the proof of (ii). It thus remains to show termination after at most polynomially many steps. This amounts to showing that the two **while** loops terminate after at most polynomially many steps since it is easy to see that all the tests (in the **while** conditions and **if** clauses) and operations (node and conjunction replacements) and also the computation of the level of nodes can be performed in polynomial time.

Termination after polynomially many steps is obvious for the loop in the norm procedure since, in every iteration, the number of variables in c decreases (and the algorithm introduces no new variables). Now for the **while** loop in the main procedure. If a fork $\mathsf{conc}_w(y,x), \mathsf{conc}_{w'}(y',x)$ is found, then $\mathsf{conc}_w(y,x)$ is replaced by $\mathsf{conc}_{\mathsf{pre}(w,w')}(y,y')$. As was already noted, this and the application of norm may generate new forks $\mathsf{conc}_{\tilde{w}}(\tilde{y},\tilde{x}), \mathsf{conc}_{\tilde{w}'}(\tilde{y}',\tilde{x})$ but only with the restriction $\mathsf{lev}(\tilde{x}) > \mathsf{lev}(x)$. Hence the newly generated fork will not be considered during the current iteration step of the **for** loop. Since the number of forks $\mathsf{conc}_w(y,x), \mathsf{conc}_{w'}(y',x)$ with $\mathsf{lev}(x) = i$ is clearly bounded by $|c|^2$, we conclude that the **while** loop terminates after polynomially many steps. ❑

The following proposition is an immediate consequence of the lemma just proved.

Proposition 5.11. *It is decidable in deterministic polynomial time whether a finite conjunction of predicates from* W *has a solution.*

Corollary 5.12. *The concrete domain* W *is admissible.*

On first sight, the concrete domain W may look somewhat artificial and one may question the relevance of lower bounds for Description Logics with concrete domains that have been obtained using W. However, it is straightforward to encode words as natural numbers and to define concatenation of words as rather simple operations on the naturals [Baader & Hanschke 1992]: words $w \neq \epsilon$ over the alphabet Σ can be interpreted as numbers written at base $|\Sigma| + 1$ where the symbol that is the "0 digit" does never occur. Hence, we can use the corresponding natural number (at base 10) to represent a word w and the number 0 to represent the empty word. The concatenation of two words v and w can then be expressed as $vw = v \cdot (|\Sigma| + 1)^{|w|} + w$, where $|w|$ denotes the length of the word w. Moreover, exponentiation and multiplication can be expressed as multiple additions.[2] These observations give rise to the following definition:

Definition 5.13. A concrete domain $\mathcal{D}$ is called *arithmetic* iff

1. $\Delta_\mathcal{D}$ contains the natural numbers and
2. $\Phi_\mathcal{D}$ contains
   - unary predicates for equality and inequality with zero,
   - binary predicates for equality and inequality,
   - a binary predicate expressing addition with 1, and
   - a ternary predicate expressing addition.

$\diamond$

[2] Thanks to Ralf Treinen who suggested this.

Using the above considerations, the following theorem is easily proved.

Theorem 5.14. *If $\mathcal{ALC}(\mathsf{W})$-concept satisfiability is hard for a complexity class* C *closed under polynomial reductions, then, for every arithmetic concrete domain $\mathcal{D}$, $\mathcal{ALC}(\mathcal{D})$-concept satisfiability is also hard for* C.

Proof. We only sketch the proof, which is by reduction. Let C be an $\mathcal{ALC}(\mathsf{W})$-concept. To translate C into an eqi-satisfiable $\mathcal{ALC}(\mathcal{D})$-concept C', we need to rewrite subconcepts of the form $\exists u_1, \dots, u_n.P$. Most cases are simple (we use self-explanatory names for the predicates in $\Phi_\mathcal{D}$):

$$\begin{array}{rclcrcl} \exists u.\mathsf{word} & \rightsquigarrow & \exists u,u.= & \quad & \exists u.\mathsf{nword} & \rightsquigarrow & \bot \\ \exists u.=_\epsilon & \rightsquigarrow & \exists u.=_0 & & \exists u.\neq_\epsilon & \rightsquigarrow & \exists u.\neq_0 \end{array}$$

$\exists u_1, u_2.=$ and $\exists u_1, u_2.\neq$ do not need to be rewritten.

Now for the only interesting case $\exists u_1, u_2.\mathsf{conc}_w$. As mentioned above, the result D of translating this concept must be such that, for all interpretations $\mathcal{I}$, $a \in D^\mathcal{I}$ implies $u_2^\mathcal{I}(a) = u_1^\mathcal{I}(a) \cdot (|\Sigma| + 1)^{|w|} + w$. Abbreviate $(|\Sigma| + 1)^{|w|}$ by n and let $m_2, \dots, m_n$ and $s_1, \dots, s_w$ be concrete features not occurring in C. Replace $\exists u_1, u_2.\mathsf{conc}_w$ with

$$\begin{array}{l} \exists u_1, u_1, m_2, + \sqcap \bigsqcap_{i=2,\dots,n-1} \exists m_i, u_1, m_{i+1}, + \\ \sqcap \exists m_n, s_1, +_1 \sqcap \bigsqcap_{i=1,\dots,w-1} \exists s_i, s_{i+1}, +_1 \\ \sqcap \exists s_w, u_2.=. \end{array}$$

Note that features m_i and s_i introduced for different subconcepts $\exists u_1, u_2.\mathsf{conc}_w$ of C must be distinct. It remains to note that concepts $\exists u_1, u_2.\mathsf{nconc}_w$ can be first translated to the $\mathcal{ALC}(\mathsf{W})$-concept $\exists u_1, g.\mathsf{conc}_w \sqcap \exists u_2, g. \neq$ (where g is a concrete feature not appearing in C) and then to an $\mathcal{ALC}(\mathcal{D})$-concept. As is easily seen, the $\mathcal{ALC}(\mathcal{D})$-concept C' obtained from C by the above translations is satisfiable iff C is satisfiable. ❏

It is straightforward to prove variants of Theorem 5.14 for concept satisfiability w.r.t. acyclic or general TBoxes, for concept subsumption, and for ABox consistency. Moreover, the Theorem can also be proved for extensions of $\mathcal{ALC}$. These results show that lower bounds obtained using W apply to a large class of concrete domains including e.g. the concrete domain R introduced in Section 2.4.2 which is easily seen to be arithmetic.

5.3 Lower Bounds

Having crafted appropriate tools in the form of the NExpTime-complete PCP and the concrete domain W, we now establish the lower bounds for the five extensions of $\mathcal{ALC}(\mathcal{D})$. The employed reductions are similar in all five cases since they all reduce the NExpTime-complete PCP, but also exhibit considerable differences since the expressive power of the five logics is rather different.

$$
\begin{aligned}
\mathsf{Ch}[u_1,u_2,u_3,u_4] := {} & (\exists(u_1,u_2).= \sqcap \exists(u_3,u_4).=) \\
& \sqcup \bigsqcup_{(\ell_i,r_i) \text{ in } P} (\exists(u_1,u_2).\mathsf{conc}_{\ell_i} \sqcap \exists(u_3,u_4).\mathsf{conc}_{r_i})
\end{aligned}
$$

$$
\begin{aligned}
C_0 &\doteq \exists\ell.C_1 \sqcap \exists r.C_1 \\
&\quad \sqcap \mathsf{Ch}[\ell r^{n-1} g_\ell, r\ell^{n-1} g_\ell, \ell r^{n-1} g_r, r\ell^{n-1} g_r] \\
&\quad \vdots \\
C_{n-2} &\doteq \exists\ell.C_{n-1} \sqcap \exists r.C_{n-1} \\
&\quad \sqcap \mathsf{Ch}[\ell r g_\ell, r\ell g_\ell, \ell r g_r, r\ell g_r] \\
C_{n-1} &\doteq \mathsf{Ch}[\ell g_\ell, r g_\ell, \ell g_r, r g_r] \\
C_P &\doteq C_0 \\
&\quad \sqcap \exists \ell^n g_\ell.=_\epsilon \ \sqcap \exists \ell^n g_r.=_\epsilon \\
&\quad \sqcap \exists r^n f_2.\exists g_\ell, g_r.= \ \sqcap \exists r^n f_2 g_\ell.\neq_\epsilon \\
&\quad \sqcap \forall r^n.\big(\mathsf{Ch}[g_\ell, f_1 g_\ell, g_r, f_1 g_r] \sqcap \mathsf{Ch}[f_1 g_\ell, f_2 g_\ell, f_1 g_r, f_2 g_r]\big)
\end{aligned}
$$

Figure 5.4: The $\mathcal{ALC}(\mathsf{W})$ reduction TBox $\mathcal{T}_P$ ($n = |P|$).

5.3.1 $\mathcal{ALC}(\mathcal{D})$-concept Satisfiability w.r.t. Acyclic TBoxes

We prove that the satisfiability of $\mathcal{ALC}(\mathsf{W})$-concepts w.r.t. acyclic TBoxes is NExpTime-hard. Since Theorem 5.14 can easily be extended to acyclic TBoxes, this implies that $\mathcal{ALC}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes is NExpTime-hard for *any* arithmetic concrete domain $\mathcal{D}$. These results are rather surprising since, in Section 4.1, we saw that, for many PSpace Description Logics, admitting acyclic TBoxes does not increase the complexity of reasoning. In this thesis, $\mathcal{ALC}(\mathcal{D})$ is the second example of a logic for which admitting acyclic TBoxes does *significantly* increase the complexity of reasoning (the first was $\mathcal{ALCF}$, c.f. Section 4.2).

The proof is by a reduction of the $2^n + 1$-PCP using the concrete domain W. Given a $2^n + 1$-PCP-instance $P = (\ell_1, r_1), \dots, (\ell_k, r_k)$, we define a TBox $\mathcal{T}_P$ of size polynomial in $|P|$ and a concept (name) C_P such that C_P is satisfiable w.r.t. $\mathcal{T}_P$ iff P has a solution. Figure 5.4 contains the reduction TBox and Figure 5.5 an example model for $|P| = 2$. In the figures, ℓ, r, f_1, and f_2 denote abstract features and g_ℓ and g_r denote concrete features. The first two lines in Figure 5.4 do not contain a concept definition but an abbreviation: replace every occurrence of $\mathsf{Ch}[u_1, u_2, u_3, u_4]$ in the TBox by the right-hand side of the first equality substituting $u_1, \dots, u_4$ appropriately.

The idea is to define $\mathcal{T}_P$ such that models of C_P and $\mathcal{T}_P$ have the form of a binary tree of depth $|P|$ whose leaves have attached two sequences of words from Δ_W related via conc_w predicates. These two sequences represent the "left concatenation" and the "right concatenation" of words from P according to some "guessed" index sequence. The definition of the tree is very similar to the corresponding part in the reduction of the NExpTime-hard domino problem to $\mathcal{ALCF}$-concept satisfiability presented in

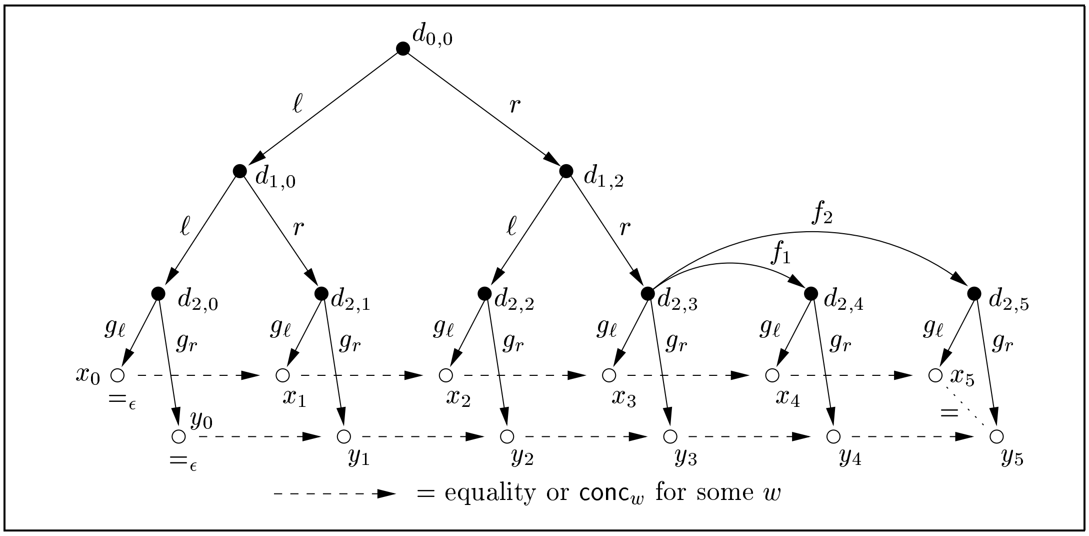

Figure 5.5: An example model of C_P and $\mathcal{T}_P$ for $n = 2$.

Section 4.2: the first line of the definitions of the $C_0, \ldots, C_{n-1}$ concepts ensures that models have the form of a binary tree of depth n (with $n = |P|$) whose left edges are labeled with the abstract feature ℓ and whose right edges are labeled with the abstract feature r. Let the domain elements $d_{n,0}, \ldots d_{n,2^n-1}$ be the leaves of this tree. By the second line of the definitions of the $C_0, \ldots, C_{n-1}$ concepts and the definition of the Ch concept, every $d_{n,i}$ has a g_ℓ-successor x_i and a g_r-successor y_i. These second lines also ensure that the sequences $x_0, \ldots, x_{2^n-1}$ and $y_0, \ldots, y_{2^n-1}$ are connected via two predicate chains, where the predicates on the chains are either equality or conc_w. More precisely, for $i < 2^n - 1$, we have either $x_i = x_{i+1}$ and $y_i = y_{i+1}$, or there exists a $j \in \{1, \ldots, k\}$ such that $(x_i, x_{i+1}) \in \mathsf{conc}_{\ell_j}^{\mathsf{W}}$ and $(y_i, y_{i+1}) \in \mathsf{conc}_{r_j}^{\mathsf{W}}$. By the second line of the definition of C_P, we have $x_0 = y_0 = \epsilon$. Intuitively, the disjunction in the Ch concept guesses appropriate indexes to find a solution and the equality predicate is used to allow for short solutions. Since we must consider solutions of a length up to $2^n + 1$, the 2^n domain elements on the fringe of the tree with their $2^n - 1$ connecting predicate edges are not sufficient, and we need to "add" two more elements $d_{n,2^n}$ and $d_{n,2^n+1}$ which behave analogously to the $d_{n,0}, \ldots d_{n,2^n-1}$. This is done by the last line of the definition of C_P. Finally, the third line of the definition of C_P ensures that $x_{2^n+1} = y_{2^n+1} \neq \epsilon$ and hence that (x_{2^n+1}, y_{2^n+1}) is in fact a solution to P.

Lemma 5.15. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP-instance. Then P has a 2^n+1-solution iff the concept (name) C_P is satisfiable w.r.t. the TBox $\mathcal{T}_P$.*

Proof. Let P be a PCP-instance as in the lemma with $|P| = n$. We assume $n \geq 2$ (otherwise, P has no or a trivial solution). First assume that C_P is satisfiable w.r.t. $\mathcal{T}_P$. Using induction on n and the definitions of the C_i concepts, it is easy to show that there exist domain elements $d_{i,j}$ for $0 \leq i \leq n$ and $0 \leq j < 2^i$ such that $d_{0,0} \in C_P^{\mathcal{I}}$,

1. $\ell^{\mathcal{I}}(d_{i,j}) = d_{(i+1),2j}$ and $r^{\mathcal{I}}(d_{i,j}) = d_{(i+1),(2j+1)}$ for $i < n$ and $j < 2^i$, and

2. $d_{i,j} \in (\mathsf{Ch}[\ell r^{n-(i+1)} g_\ell, r\ell^{n-(i+1)} g_\ell, \ell r^{n-(i+1)} g_r, r\ell^{n-(i+1)} g_r])^\mathcal{I}$ for $i < n$.

Intuitively, the first property states that the $d_{i,j}$ form a binary tree in which edges connecting left successors are labeled with ℓ and edges connecting right successors are labeled with r.[3] The naming scheme for nodes is as indicated in Figure 5.5.

We now establish a certain property for every two neighboring leaf nodes $d_{n,j}$ and $d_{n,(j+1)}$, which will allow us to deduce the existence of two sequences of words related by conc_w predicates or the equality predicate: by induction on n, it is straightforward to prove that, for any two nodes $d_{n,j}$ and $d_{n,(j+1)}$ with $j < 2^n - 1$, there exists a common ancestor $d_{m,k}$ of $d_{n,j}$ and $d_{n,(j+1)}$ such that

$$(\ell r^{n-(m+1)})^\mathcal{I}(d_{m,k}) = d_{n,j} \text{ and } (r\ell^{n-(m+1)})^\mathcal{I}(d_{m,k}) = d_{n,j+1}.$$

By Property 2 from above, we have

$$d_{m,k} \in (\mathsf{Ch}[\ell r^{n-(m+1)} g_\ell, r\ell^{n-(m+1)} g_\ell, \ell r^{n-(m+1)} g_r, r\ell^{n-(m+1)} g_r])^\mathcal{I}.$$

Since this holds independently from the choice of j, we may use the definition of the $\mathsf{Ch}[u_1, u_2, u_3, u_4]$ concept to conclude that there exist words $x_0, \dots, x_{2^n-1}$ and $y_0, \dots, y_{2^n-1}$ and indexes $i_1, \dots, i_{2^n-1} \in \{1, \dots, k\} \cup \{\clubsuit\}$ such that

1. $g_\ell^\mathcal{I}(d_{n,j}) = x_j$ and $g_r^\mathcal{I}(d_{n,j}) = y_j$ for $j < 2^n$ and,
2. for $1 \leq j < 2^n$,
   - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
   - $(x_{j-1}, x_j) \in \mathsf{conc}^\mathsf{W}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in \mathsf{conc}^\mathsf{W}_{r_{i_j}}$ otherwise.

Analogously, by the last two lines of the definition of C_P, there exist domain elements $d_{n,2^n}, d_{n,(2^n+1)}$, words $x_{2^n}, x_{2^n+1}, y_{2^n}, y_{2^n+1}$, and indexes $i_{2^n}, i_{2^n+1} \in \{1, \dots, k\} \cup \{\clubsuit\}$ such that

1. $f_1^\mathcal{I}(d_{n,2^n-1}) = d_{n,2^n}$ and $f_2^\mathcal{I}(d_{n,2^n-1}) = d_{n,(2^n+1)}$,
2. $g_\ell^\mathcal{I}(d_{n,i}) = x_i$ and $g_r^\mathcal{I}(d_{n,i}) = y_i$ for $i \in \{2^n, 2^n+1\}$, and,
3. for all $j \in \{2^n, 2^n+1\}$,
   - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
   - $(x_{j-1}, x_j) \in \mathsf{conc}^\mathsf{W}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in \mathsf{conc}^\mathsf{W}_{r_{i_j}}$ otherwise.

Moreover, by the second and third line of the definition of C_P, we have $x_0 = y_0 = \epsilon$ and $x_{2^n+1} = y_{2^n+1} \neq \epsilon$. Taking together these observations, it is clear that the sequence $i'_1, \dots, i'_p$, which can be obtained from $i_1, \dots, i_{2^n+1}$ by eliminating all i_j with $i_j = \clubsuit$, is a solution to P. Furthermore, we obviously have $1 \leq p \leq 2^n + 1$.

Now for the "only if" direction. Assume that P has a solution $i_1, \dots, i_m$ with $m \leq 2^{|P|}+1$. By L_j (resp. R_j), we denote the concatenation $\ell_{i_1} \cdots \ell_{i_j}$ (resp. $r_{i_1} \cdots r_{i_j}$)

[3] As in Section 4.2, nodes in the tree are not necessarily distinct.

for $1 \leq j \leq m$ and set $L_0 = R_0 = \epsilon$ and $L_j = L_m$ (resp. $R_j = R_m$) for all $j > m$. In the following, we define a model $\mathcal{I}$ of C_P and $\mathcal{T}_P$ which has the form of a binary tree of depth n. Again, the object names in Figure 5.5 indicate the naming scheme used.

$$\Delta^{\mathcal{I}} := \{d_{i,j} \mid 0 \leq i \leq n, 0 \leq j < 2^i\} \cup \{d_{n,2^n}, d_{n,(2^n+1)}\}$$

$$\text{for } i < n, \text{ set } C_i^{\mathcal{I}} := \{d_{i,j} \mid j < 2^i\}$$

$$\text{for } i < n \text{ and } j < 2^i, \text{ set } \ell^{\mathcal{I}}(d_{i,j}) := d_{(i+1),(2j)} \text{ and } r^{\mathcal{I}}(d_{i,j}) := d_{(i+1),(2j+1)}$$

$$f_1^{\mathcal{I}}(d_{n,(2^n-1)}) := d_{n,2^n} \text{ and } f_2^{\mathcal{I}}(d_{n,(2^n-1)}) := d_{n,(2^n+1)}$$

$$\text{for } i \leq 2^n+1, \text{ set } g_\ell^{\mathcal{I}}(d_{n,i}) := L_i \text{ and } g_r^{\mathcal{I}}(d_{n,i}) := R_i$$

$$C_P^{\mathcal{I}} := \{d_{0,0}\}$$

It is not hard to verify that $\mathcal{I}$ is a model of $\mathcal{T}_P$ and that $d_{0,0} \in C_P^{\mathcal{I}}$. ❑

Obviously, the size of $\mathcal{T}_P$ is polynomial in $|P|$ and $\mathcal{T}_P$ can be constructed in time polynomial in $|P|$ which yields the following theorem:

Theorem 5.16. *$\mathcal{ALC}(\mathsf{W})$-concept satisfiability w.r.t. acyclic TBoxes is* NExpTime-*hard.*

To contrast this result with Theorem 3.13, it is interesting to take into account the complexity of W-satisfiability.

Corollary 5.17. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and $\mathcal{ALC}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes is* NExpTime-*hard.*

Finally, the fact that Theorem 5.14 can be straightforwardly extended to take into account TBoxes yields the following corollary:

Corollary 5.18. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t. acyclic TBoxes is* NExpTime-*hard.*

In all three cases, we obtain a co-NExpTime lower bound for subsumption since unsatisfiability w.r.t. acyclic TBoxes can be reduced to subsumption w.r.t. acyclic TBoxes.

5.3.2 $\mathcal{ALC}^{\sqcap}(\mathcal{D})$-concept Satisfiability

We now turn our attention towards $\mathcal{ALC}^{\sqcap}(\mathcal{D})$, the extension of $\mathcal{ALC}(\mathcal{D})$ with the role conjunction constructor from Section 2.1.2. This constructor is allowed only inside existential and universal value restrictions where it may be applied to both roles and abstract features (even intermixed). Hence, $\exists(R \sqcap f \sqcap f').A$ is an $\mathcal{ALC}^{\sqcap}(\mathcal{D})$-concept, but neither $\exists(g \sqcap g').P$ nor $\exists(f \sqcap f')g, g'.P$ are.[4] In this section, we show that $\mathcal{ALC}^{\sqcap}(\mathsf{W})$-concept satisfiability (without reference to TBoxes) is NExpTime-hard. As in the previous section, it is the addition of a seemingly harmless looking means of expressivity that causes a considerable increase in complexity.

[4] Although, with the obvious semantics, the latter concept is equivalent to the $\mathcal{ALC}^{\sqcap}(\mathcal{D})$-concept $\exists(f \sqcap f').\top \sqcap \exists fg, g'.P$.

$$
\begin{aligned}
\mathsf{Ch}[u_1,u_2,u_3,u_4] :=\ & (\exists(u_1,u_2).= \ \sqcap \exists(u_3,u_4).=) \\
& \sqcup \bigsqcup_{(\ell_i,r_i) \text{ in } P} (\exists(u_1,u_2).\mathsf{conc}_{\ell_i} \sqcap \exists(u_3,u_4).\mathsf{conc}_{r_i})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Tree} :=\ & \exists(R\sqcap\ell).\top \sqcap \exists(R\sqcap r).\top \\
& \sqcap \mathsf{Ch}[\ell r^{n-1}g_\ell, r\ell^{n-1}g_\ell, \ell r^{n-1}g_r, r\ell^{n-1}g_r] \\
& \sqcap \forall R.\big(\exists(R\sqcap\ell).\top \sqcap \exists(R\sqcap r).\top \\
& \qquad \sqcap \mathsf{Ch}[\ell r^{n-2}g_\ell, r\ell^{n-2}g_\ell, \ell r^{n-2}g_r, r\ell^{n-2}g_r]\big) \\
& \qquad\qquad \vdots \\
& \sqcap \forall R^{n-2}.\big(\exists(R\sqcap\ell).\top \sqcap \exists(R\sqcap r).\top \\
& \qquad \sqcap \mathsf{Ch}[\ell r g_\ell, r\ell g_\ell, \ell r g_r, r\ell g_r]\big) \\
& \sqcap \forall R^{n-1}.\mathsf{Ch}[\ell g_\ell, r g_\ell, \ell g_r, r g_r]
\end{aligned}
$$

$$
\begin{aligned}
C_P :=\ & \mathsf{Tree} \\
& \sqcap \exists\ell^n g_\ell. =_\epsilon \ \sqcap \exists\ell^n g_r. =_\epsilon \\
& \sqcap \exists r^n f_2.\exists g_\ell, g_r. = \ \sqcap \exists r^n f_2 g_\ell. \neq_\epsilon \\
& \sqcap \forall r^n.\big(\mathsf{Ch}[g_\ell, f_1 g_\ell, g_r, f_1 g_r] \sqcap \mathsf{Ch}[f_1 g_\ell, f_2 g_\ell, f_1 g_r, f_2 g_r]\big)
\end{aligned}
$$

Figure 5.6: The $\mathcal{ALC}^{\sqcap}(\mathsf{W})$ reduction concept C_P ($n = |P|$).

The proof is again by a reduction of the 2^n+1-PCP, which is very similar to the reduction presented in the previous section. However, we have to compensate the lack of acyclic TBoxes by using the role conjunction constructor. The key observation is that TBoxes have been used to "propagate" the concepts $C_0, \ldots, C_{n-1}$ to the proper positions in the tree. If we try to do this using $\mathcal{ALC}(\mathcal{D})$ without TBoxes, we obtain a concept of size exponential in $|P|$, which is obviously not acceptable. Fortunately, the role conjunction constructor can help: we construct the tree such that left edges are labeled with $\ell \sqcap R$ and right edges with $r \sqcap R$. This allows to use the standard universal value restriction "over" the role R to propagate the $C_0, \ldots, C_{n-1}$ concepts (resp. their equivalents in the current reduction, which are not assigned a name) to the appropriate levels of the tree. The reduction concept C_P can be found in Figure 5.6. The equations are not to be read as concept definitions from a TBox but serve as abbreviations. Models of C_P have the form of the interpretation displayed in Figure 5.5, with the only difference that all edges labeled with ℓ and r are additionally labeled with R.

The proof of the following lemma is omitted since it is very similar to the proof of Lemma 5.15.

Lemma 5.19. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP-instance. Then P has a 2^n+1-solution iff the concept C_P is satisfiable.*

The size of C_P is polynomial in $|P|$ and C_P can be constructed in time polynomial in $|P|$.

Theorem 5.20. *$\mathcal{ALC}^{\sqcap}(\mathsf{W})$-concept satisfiability is* NExpTime*-hard.*

As in the previous section, the following corollary is easily obtained:

Corollary 5.21. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and $\mathcal{ALC}^{\sqcap}(\mathcal{D})$-concept satisfiability is* NExpTime*-hard.*

Since the concrete domain W can be replaced by any arithmetic concrete domain as in the proof of Theorem 5.14, we obtain:

Corollary 5.22. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALC}^{\sqcap}(\mathcal{D})$-concepts is* NExpTime*-hard.*

In all three cases, we obtain a corresponding co-NExpTime lower bound for concept subsumption. It is worth noting that the reduction crucially depends on the fact that abstract features may be used inside role conjunctions. Let $\mathcal{ALC}^{(\sqcap)}(\mathcal{D})$ be the Description Logic obtained from $\mathcal{ALC}^{\sqcap}(\mathcal{D})$ by disallowing the use of features in role conjunctions. $\mathcal{ALC}^{(\sqcap)}(\mathcal{D})$ is the *fusion* or *independent join* of the logics $\mathcal{ALC}^{\sqcap}$ and $\mathcal{ALC}(\mathcal{D})$. Intuitively, this means that, in $\mathcal{ALC}^{(\sqcap)}(\mathcal{D})$, there is "no interaction" between the constructors of $\mathcal{ALC}^{\sqcap}$ and the constructors of $\mathcal{ALC}(\mathcal{D})$ [Baader *et al.* 2002a]. It is well-known that, in many cases, the complexity of the fusion of two logics is identical to the complexity of the "harder one" of the two component logics [Spaan 1993a]. Because of this and since, in the case of $\mathcal{ALC}^{(\sqcap)}(\mathcal{D})$, both component logics are PSpace-complete if $\mathcal{D}$-satisfiability is in PSpace (see [Donini *et al.* 1997] and Chapter 3), we conjecture that $\mathcal{ALC}^{(\sqcap)}(\mathcal{D})$-concept satisfiability is also PSpace-complete if $\mathcal{D}$-satisfiability is in PSpace. We shall discuss fusions in more detail in Section 5.6.

5.3.3 $\mathcal{ALC}^{-}(\mathcal{D})$-concept Satisfiability

We consider $\mathcal{ALC}^{-}(\mathcal{D})$, i.e. $\mathcal{ALC}(\mathcal{D})$ extended with inverse roles, and show that this logic is another example for a seemingly harmless extension of $\mathcal{ALC}(\mathcal{D})$ for which concept satisfiability is much harder than for $\mathcal{ALC}(\mathcal{D})$ itself. More precisely, the logic $\mathcal{ALC}^{-}(\mathcal{D})$ is obtained from $\mathcal{ALC}(\mathcal{D})$ by allowing the use of inverse roles (and inverse abstract features) inside existential and universal value restrictions. Hence, $\exists R^{-}.\forall f^{-}.A$ is an $\mathcal{ALC}^{-}(\mathcal{D})$-concept, but $\exists f^{-}g, f'g'.P$ and $\exists fg^{-}, f'g'.P$ are not. The reason for *not* admitting inverse features inside the concrete domain constructor is that we want to keep this constructor restricted to functional roles and inverse features are not necessarily functional.

As in the previous sections, we reduce the 2^n+1-PCP using the concrete domain W. Although models of the reduction concept will again have the form of a binary tree, the employed strategy differs: in the case of inverse roles, it is not possible to enforce chains of predicates connecting the leaves of the tree. Therefore, we construct the reduction concept such that the predicate chains emulate the structure of the tree following the scheme indicated in Figure 5.7. The reduction concept C_P

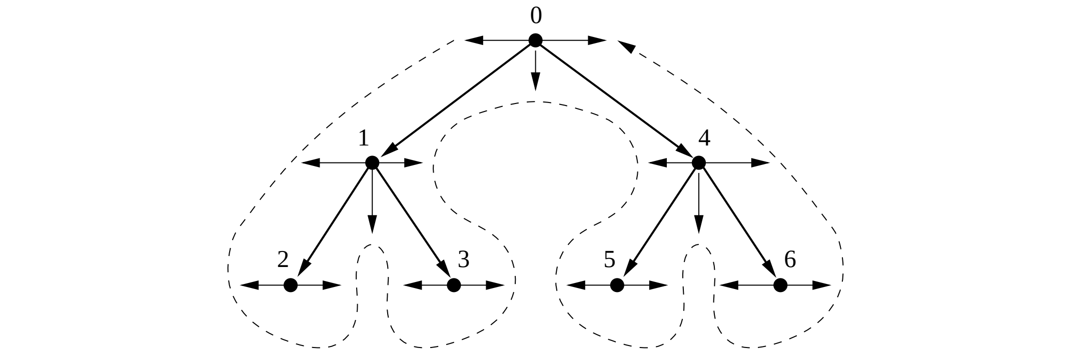

Figure 5.7: Predicate chains in models of C_P.

for the 2^n+1-PCP $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ can be found in Figure 5.8. In the figure, $g_\ell, g_r, h_\ell, h_r, p_\ell, p_r, x_\ell, x_r, y_\ell, y_r, z_\ell$, and z_r are concrete features. Note that the figure does only define abbreviations, but does not contain concept equalities from a TBox.

Before giving a formal correctness proof, we dicuss the structure of models of C_P on an intuitive level. Let P be a 2^n+1-PCP with $|P| = n$. Due to the first line in the definition of C_P and the $\exists f^-$ quantifiers in the definition of X, models of C_P have the form of a tree of depth $n-1$ in which all edges are labeled with f^-. This edge labeling scheme is possible since, as already noted, the inverse of an abstract feature is not necessarily functional. Now for the two chains of concrete domain predicates. To illustrate the use of the various concrete features for establishing the chains, Figure 5.9 shows a more detailed clipping from a model of C_P. The predicate chains are enforced as follows: the concept X establishes the edges of the predicate chains as depicted in Figure 5.9 (in fact, Figure 5.9 is a model of the concept X) while the second line of C_P establishes the edges "leading around" the leaves. Edges of the latter type and the dotted edges in Figure 5.9 are labeled with the equality predicate. To see why this is the case, let us investigate the length of the chains.

The length of the two predicate chains is twice the number of edges in the tree plus the number of leaves, i.e., $2 \cdot (2^n - 2) + 2^{n-1}$. To eliminate the factor 2 and the summand 2^{n-1}, C_P is defined such that every edge in the predicate chains leading "up" in the tree and every edge "leading around" a leaf is labeled with the equality predicate, i.e., these edges do not contribute to PCP-solutions. To extend the chains to length 2^n+1, we need to add three additional edges (definition of C_P, lines three, four, and five). Finally, the last two lines in the definition of C_P ensure that the first words on both chains represents the empty word and that the last words represent a (non-empty) solution to P.

Lemma 5.23. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP-instance. Then P has a 2^n+1-solution iff the concept C_P is satisfiable.*

Proof. Let $|P| = n$ and assume $n \geq 2$. For the "if" direction, let C_P be satisfiable, i.e., assume that there exists an interpretation $\mathcal{I}$ and a $d \in \Delta^\mathcal{I}$ such that $d \in C_P^\mathcal{I}$. Using induction over n and considering the first line of the definition of C_P and the

$$
\begin{aligned}
\mathsf{Ch}[u_1,u_2,u_3,u_4] &:= (\exists(u_1,u_2).= \ \sqcap \ \exists(u_3,u_4).=) \\
&\qquad \sqcup \bigsqcup_{(\ell_i,r_i) \text{ in } P} \exists(u_1,u_2).\mathsf{conc}_{\ell_i} \sqcap \exists(u_3,u_4).\mathsf{conc}_{r_i} \\
X &:= \exists f^-.(\mathsf{Ch}[fg_\ell, g_\ell, fg_r, g_r] \sqcap \exists(h_\ell, fp_\ell).= \ \sqcap \ \exists(h_r, fp_r).=) \\
&\quad \sqcap \exists f^-.(\mathsf{Ch}[fp_\ell, g_\ell, fp_r, g_r] \sqcap \exists(h_\ell, fh_\ell).= \ \sqcap \ \exists(h_r, fh_r).=) \\
C_P &:= X \sqcap \forall f^-.X \sqcap \cdots \sqcap \forall (f^-)^{n-2}.X \\
&\quad \sqcap \forall (f^-)^{n-1}.(\exists(g_\ell, h_\ell).= \ \sqcap \ \exists(g_r, h_r).=) \\
&\quad \sqcap \mathsf{Ch}[h_\ell, x_\ell, h_r, x_r] \\
&\quad \sqcap \mathsf{Ch}[x_\ell, y_\ell, x_r, y_r] \\
&\quad \sqcap \mathsf{Ch}[y_\ell, z_\ell, y_r, z_r] \\
&\quad \sqcap \exists g_\ell.=_\epsilon \ \sqcap \ \exists g_r.=_\epsilon \\
&\quad \sqcap \exists z_\ell, z_r.= \ \sqcap \ \exists z_\ell. \neq_\epsilon
\end{aligned}
$$

Figure 5.8: The $\mathcal{ALC}^-(\mathsf{W})$ reduction concept C_P ($n = |P|$).

definition of X, it is easy to show that there exist domain elements $d_{i,j}$ for $0 \leq i < n$ and $0 \leq j < 2^i$ such that $d_{0,0} \in C_P^{\mathcal{I}}$ and, for $i < n-1$ and $j < 2^i$, we have

1. $\{(d_{i,j}, d_{(i+1),2j}), (d_{i,j}, d_{(i+1),(2j+1)})\} \subseteq (f^-)^{\mathcal{I}}$,
2. $d_{(i+1),2j} \in (\mathsf{Ch}[fg_\ell, g_\ell, fg_r, g_r] \sqcap \exists(h_\ell, fp_\ell).= \sqcap \exists(h_r, fp_r).=)^{\mathcal{I}}$, and
3. $d_{(i+1),(2j+1)} \in (\mathsf{Ch}[fp_\ell, g_\ell, fp_r, g_r] \sqcap \exists(h_\ell, fh_\ell).= \sqcap \exists(h_r, fh_r).=)^{\mathcal{I}}$.

The first property states that the $d_{i,j}$ form a binary tree of depth $n-1$ whose edges are labeled with f^-. For the remaining proof, it is convenient to number the nodes in the tree in a different way. To do this, in turn, it is convenient to define some auxiliary functions.

Let T be a binary tree of depth $n-1$ whose nodes are labeled with natural numbers in preorder such that the root is labeled with 0 (this tree is independent of the $d_{i,j}$ and of $\mathcal{I}$ in general).[5] By $\mathsf{sucl}(i)$ and $\mathsf{sucr}(i)$, we respectively denote the node label of the left and right successor of the node labeled with i in T ($\mathsf{sucl}(i)$ and $\mathsf{sucr}(i)$ are undefined if the given node has no successors). Furthermore, for $i \in \mathbb{N}$, $\mathsf{lev}(i)$ denotes the level of the node in T labeled with i (where the root is on level 0) and is undefined if no such node exists. By "renaming" the nodes $d_{i,j}$, it is easy to show that there exist domain elements $e_0, \ldots, e_{2^n-2}$ such that, for $i \leq 2^n - 2$ with $\mathsf{lev}(i) < n-1$, we have

1. $f^{\mathcal{I}}(e_{\mathsf{sucl}(i)}) = e_i$ and $f^{\mathcal{I}}(e_{\mathsf{sucr}(i)}) = e_i$,

[5]To label a tree in preorder, first label its root, then inductively label the subtree induced by the root's left successor and finally label the subtree induced by the root's right successor. This numbering scheme is demonstrated in Figure 5.7.

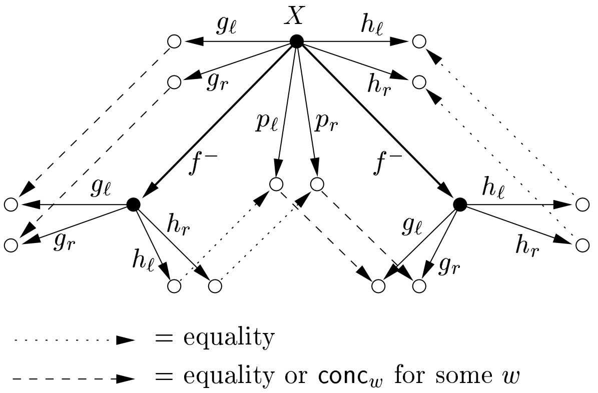

Figure 5.9: A clipping from a model of C_P.

2. $e_{\mathsf{sucl}(i)} \in (\mathsf{Ch}[fg_\ell, g_\ell, fg_r, g_r] \sqcap \exists(h_\ell, fp_\ell).{=} \sqcap \exists(h_r, fp_r).{=})^\mathcal{I}$, and

3. $e_{\mathsf{sucr}(i)} \in (\mathsf{Ch}[fp_\ell, g_\ell, fp_r, g_r] \sqcap \exists(h_\ell, fh_\ell).{=} \sqcap \exists(h_r, fh_r).{=})^\mathcal{I}$.

Intuitively, the e_i form a binary tree of depth $n-1$ labeled in preorder whose edges are labeled with f^-. In the following, when we talk of the nodes of the tree, we mean the elements $e_0, \ldots, e_{2^n-2}$. By the second line of C_P and definition of X, there exist words $x_0, \ldots, x_{2^n-2}, y_0, \ldots, y_{2^n-2}$ such that $g_\ell^\mathcal{I}(e_i) = x_i$ and $g_r^\mathcal{I}(e_i) = y_i$ for $i \leq 2^n - 2$. Next, we prove the following claim:

Claim: For $j < 2^n - 2$, we have either $x_j = x_{j+1}$ and $y_j = y_{j+1}$ or there exists an $i \in \{1, \ldots, k\}$ such that $(x_j, x_{j+1}) \in \mathsf{conc}_{\ell_i}^{\mathsf{W}}$ and $(y_j, y_{j+1}) \in \mathsf{conc}_{r_i}^{\mathsf{W}}$.

Fix a j with $j < 2^n - 2$. From the pre-order numbering scheme, it follows that two cases can be distinguished:

(i) $\mathsf{lev}(e_j) < n - 1$ (i.e., e_j is not a leaf node). Then $j + 1 = \mathsf{sucl}(j)$. By Property 2 from above, we have $e_{j+1} \in (\mathsf{Ch}[fg_\ell, g_\ell, fg_r, g_r])^\mathcal{I}$. By definition of Ch, this implies the statement from the claim.

(ii) $\mathsf{lev}(e_j) = n - 1$ (i.e., e_j is a leaf node). Then there exists a node e_t and nodes $e_{s_0}, \ldots, e_{s_m}$ ($m \geq 0$) such that

- $j + 1 = \mathsf{sucr}(t)$,
- $s_0 = \mathsf{sucl}(t)$,
- for ℓ with $\ell < m$, $s_{\ell+1} = \mathsf{sucr}(s_\ell)$, and
- $s_m = j$

By Properties 1-3 from above, we have

- $e_{j+1} \in (\mathsf{Ch}[fp_\ell, g_\ell, fp_r, g_r])^\mathcal{I}$,
- $e_{s_0} \in (\exists(h_\ell, fp_\ell).{=} \sqcap \exists(h_r, fp_r).{=})^\mathcal{I}$, and

- $e_{s_1}, \ldots, e_{s_m} \in (\exists(h_\ell, fh_\ell).{=} \sqcap \exists(h_r, fh_r).{=})^{\mathcal{I}}$.

  Moreover, since e_j is a leaf node, by the second line of the definition of C_P, we have
  $$e_j \in (\exists(g_\ell, h_\ell).{=} \sqcap \exists(g_r, h_r).{=})^{\mathcal{I}}.$$
  Using the definition of Ch, it is now straightforward to verify that the claim holds.

It is an immediate consequence of the claim that there exist indexes $i_1, \ldots, i_{2^n-2} \in \{1, \ldots, k\} \cup \{\clubsuit\}$ such that, for $0 < j \leq 2^n - 2$,

- if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
- $(x_{j-1}, x_j) \in \mathsf{conc}^{\mathsf{W}}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in \mathsf{conc}^{\mathsf{W}}_{r_{i_j}}$ otherwise.

Similarly, by the third, fourth, and fifth line of the definition of C_P, there exist words $x_{2^n-1}, x_{2^n}, x_{2^n+1}, y_{2^n-1}, y_{2^n}, y_{2^n+1}$ and indexes $i_{2^n-1}, i_{2^n}, i_{2^n+1} \in \{1, \ldots, k\} \cup \{\clubsuit\}$ such that

1. $x_\ell^{\mathcal{I}}(e_0) = x_{2^n-1}$ and $x_r^{\mathcal{I}}(e_0) = y_{2^n-1}$,
   $y_\ell^{\mathcal{I}}(e_0) = x_{2^n}$ and $y_r^{\mathcal{I}}(e_0) = y_{2^n}$,
   $z_\ell^{\mathcal{I}}(e_0) = x_{2^n+1}$ and $z_r^{\mathcal{I}}(e_0) = y_{2^n+1}$, and
2. for $j \in \{2^n - 1, 2^n, 2^n + 1\}$
   - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
   - $(x_{j-1}, x_j) \in \mathsf{conc}^{\mathsf{W}}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in \mathsf{conc}^{\mathsf{W}}_{r_{i_j}}$ otherwise.

Moreover, by the last two lines of the definition of C_P, we have $x_0 = y_0 = \epsilon$ and $x_{2^n+1} = y_{2^n+1} \neq \epsilon$. Taking together these observations, it is clear that the sequence $i'_1, \ldots, i'_p$, which can be obtained from $i_1, \ldots, i_{2^n+1}$ by eliminating all i_j with $i_j = \clubsuit$, is a $2^n + 1$-solution for P.

Now for the "only if" direction. Assume that P has a solution $i_1, \ldots, i_m$ with $m \leq 2^n + 1$. By K^ℓ_j (resp. K^r_j), we denote the concatenation $\ell_{i_1} \cdots \ell_{i_j}$ (resp. $r_{i_1} \cdots r_{i_j}$) for $1 \leq j \leq m$ and set $K^\ell_0 = K^r_0 = \epsilon$ and $K^\ell_j = K^\ell_m$ (resp. $K^r_j = K^r_m$) for all $j > m$. We define a model for C_P with the form of a binary tree of depth $n - 1$.

$$\Delta^{\mathcal{I}} := \{d_i \mid 0 \leq i < 2^n - 2\}$$

For i with $i < 2^n - 2$ and $\mathsf{lev}(i) < n$ set
$$f^{\mathcal{I}}(d_{\mathsf{sucl}(i)}) = d_i \text{ and } f^{\mathcal{I}}(d_{\mathsf{sucr}(i)}) = d_i.$$

It remains to define the interpretation of the concrete features. We first define only some of them:

1. $g_\ell^{\mathcal{I}}(d_i) = K^\ell_{i-1}$ and $g_r^{\mathcal{I}}(d_i) = K^r_{i-1}$ for $i \leq 2^n - 2$
2. $x_\ell^{\mathcal{I}}(d_0) = K^\ell_{2^n-1}$ and $x_r^{\mathcal{I}}(d_0) = K^r_{2^n-1}$

3. $y_\ell^\mathcal{I}(d_0) = K_{2^n}^\ell$ and $y_r^\mathcal{I}(d_0) = K_{2^n}^r$
4. $z_\ell^\mathcal{I}(d_0) = K_{2^n+1}^\ell$ and $z_r^\mathcal{I}(d_0) = K_{2^n+1}^r$

Based on this, we now define the interpretation of the remaining concrete features. By $\mathsf{sucr}^j(i)$, we denote the j-fold composition of sucr. For $i \leq 2^n - 2$ and $t \in \{\ell, r\}$, set

$$h_t^\mathcal{I}(d_i) := \begin{cases} g_t^\mathcal{I}(d_i) & \text{if } \mathsf{lev}(i) = n-1 \\ g_t^\mathcal{I}(d_{\mathsf{sucr}^{n-(\mathsf{lev}(i)+1)}(i)}) & \text{otherwise} \end{cases}$$

$$p_t^\mathcal{I}(d_i) := \begin{cases} g_t^\mathcal{I}(d_{\mathsf{sucl}(i)}) & \text{if } \mathsf{lev}(i) = n-2 \\ g_t^\mathcal{I}(d_{\mathsf{sucr}^{n-(\mathsf{lev}(i)+2)}(\mathsf{sucl}(i))}) & \text{if } \mathsf{lev}(i) < n-2 \end{cases}$$

Nodes d_i with $\mathsf{lev}(i) = n-1$ do not need to have successors for the concrete features p_ℓ and p_r. It is straightforward to check that $\mathcal{I}$ is well-defined and that $d_0 \in C_P^\mathcal{I}$. ❑

The size of C_P is polynomial in $|P|$ and C_P can be constructed in time polynomial in $|P|$.

Theorem 5.24. *$\mathcal{ALC}^-(\mathsf{W})$-concept satisfiability is* NExpTime*-hard.*

Again, we obtain two corollaries and corresponding co-NExpTime lower bounds for subsumption.

Corollary 5.25. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and $\mathcal{ALC}^-(\mathcal{D})$-concept satisfiability is* NExpTime*-hard.*

Corollary 5.26. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALC}^-(\mathcal{D})$-concepts is* NExpTime*-hard.*

Similar to the interaction of abstract features and the role conjunction constructor in the $\mathcal{ALC}^\sqcap(\mathcal{D})$-reduction, the $\mathcal{ALC}^-(\mathcal{D})$-reduction crucially depends on the fact that we allow the application of the inverse constructor to abstract features. If the application of this constructor is restricted to roles from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$, then we obtain the logic $\mathcal{ALC}^{(-)}(\mathcal{D})$ which is the fusion of the two DLs $\mathcal{ALC}^-$ and $\mathcal{ALC}(\mathcal{D})$. On the same grounds as for $\mathcal{ALC}^{(\sqcap)}(\mathcal{D})$ in the previous section, we thus conjecture that $\mathcal{ALC}^{(-)}(\mathcal{D})$-concept satisfiability is in PSpace if $\mathcal{D}$-satisfiability is in PSpace.

5.3.4 $\mathcal{ALCP}(\mathcal{D})$-concept Satisfiability

The reduction strategy presented in the previous section can be adapted for proving NExpTime-hardness of $\mathcal{ALCP}(\mathcal{D})$-concept satisfiability, where $\mathcal{ALCP}(\mathcal{D})$ is the Description Logic obtained by extending $\mathcal{ALC}(\mathcal{D})$ with the generalized concrete domain constructors introduced in Section 2.3.2. Hence, this logic is the fourth example for an extension of $\mathcal{ALC}(\mathcal{D})$ with a NExpTime-hard concept satisfiability problem.

The crucial task in the reduction described in the previous section is to enforce a structure as displayed in Figure 5.9. For the $\mathcal{ALCP}(\mathcal{D})$-reduction, however, the inverse constructor is not available and thus we cannot achieve a labeling of the edges connecting left and right successors in the tree with the inverse of an abstract feature. Instead, we use a single role R from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$ to label both left and right successors. The main problem with this approach is to establish the predicate edges:

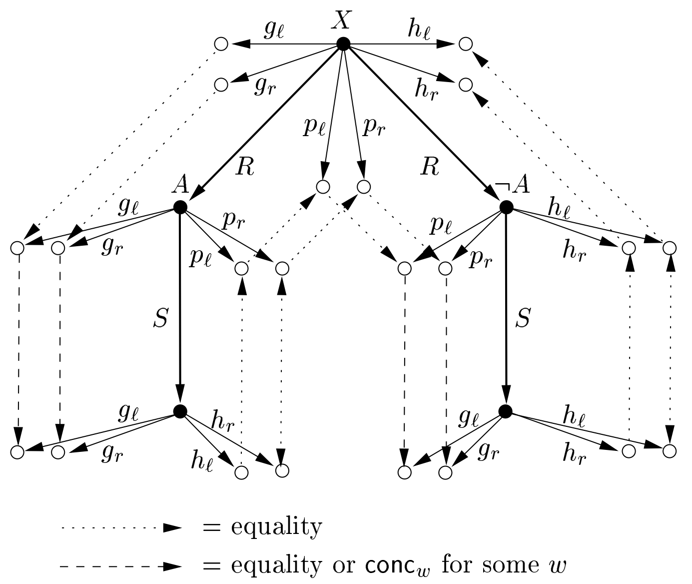

Figure 5.10: A clipping from a model of C_P.

- First assume that the existential version of the generalized concrete domain constructor is used to establish the predicate edges. Then we cannot establish the two leftmost edges in Figure 5.9, i.e., the ones connecting the g_ℓ and g_r-successors of the root node with the corresponding successors of its left child. Using the concept $\exists g_\ell, Rg_\ell.P \sqcap \exists g_r, Rg_r.P$ to establish these edges will not work since R is not functional and the two existential restrictions can be satisfied by *two* distinct R-successors.

- Now assume that the universal version of the generalized concrete domain constructor is used to establish the predicate edges. Using the concept

$$\exists R.\top \ \sqcap \ \forall g_\ell, Rg_\ell.P \ \sqcap \ \forall g_r, Rg_r.P \ \sqcap \ \forall Rh_\ell, p_\ell.= \ \sqcap \ \forall Rh_r, p_r.=,$$

  we can enforce the existence of the left successor in the tree together with all predicate edges leading from the root note to this left successor. However, if we additionally try to establish the right successor, it will be equipped with exactly the same predicate edges as the left successor due to the use of the universal concrete domain constructor. Hence, we cannot construct a structure as in Figure 5.9.

The solution to this problem is to replace the structure from Figure 5.9 by a structure as displayed in Figure 5.10. Note that both R-successors of the root node agree with the root node on all concrete successors (only dotted edges on the upper level). Intuitively, the upper level of the structure deals with branching while the lower one deals with establishing the required predicate edges. The latter task is now easy since it suffices to have a single S-successor and thus we may use the universal concrete

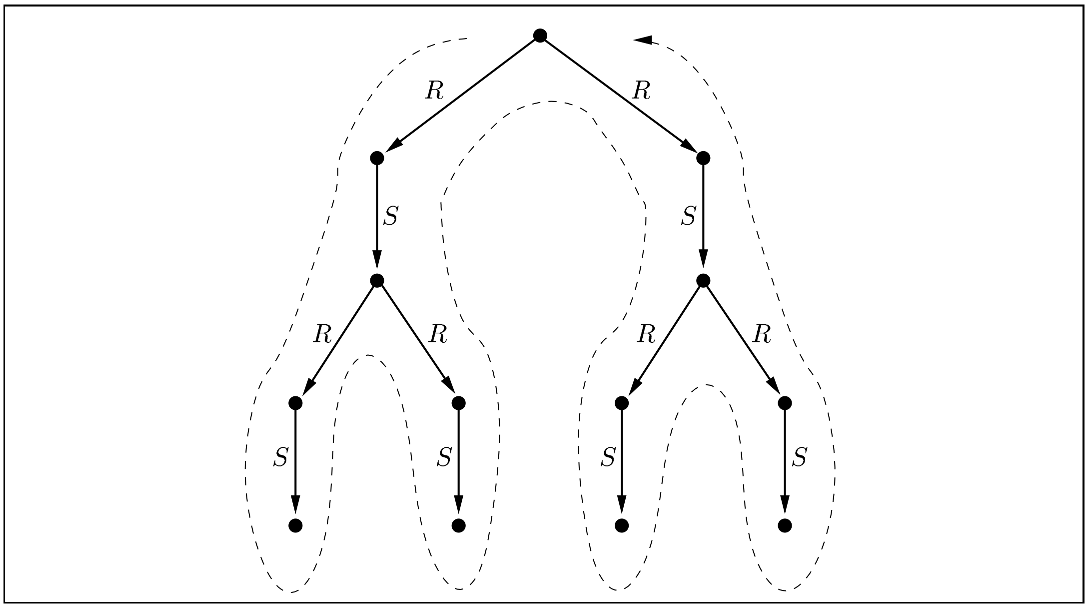

Figure 5.11: Predicate chains in models of C_P.

domain constructor as described above. To give a more global picture, the structure of models of the reduction concept is as shown in Figure 5.11: one level of the tree in Figure 5.7 is replaced by two levels in Figure 5.11. The reduction concept itself can be found in Figure 5.12. The only real difference between the concepts in Figures 5.12 and 5.8 is the more complex definition of X. This concept is best understood by considering the structure in Figure 5.10, which is a model of X. It is interesting to note that abstract features are not used in the reduction concept. Hence, removing them from the language does still yield a Description Logic with a NExpTime-hard concept satisfiability problem.

The proof of the following lemma is omitted since it is very similar to the proof of Lemma 5.23.

Lemma 5.27. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP-instance. Then P has a 2^n+1-solution iff the concept C_P is satisfiable.*

Since C_P can be constructed in time polynomial in $|P|$ and $|C_P|$ is polynomial in $|P|$, we obtain the following theorem:

Theorem 5.28. *$\mathcal{ALCP}(\mathsf{W})$-concept satisfiability is* NExpTime*-hard.*

It remains to state the usual two corollaries and note that we obtain co-NExpTime-hardness for concept subsumption.

Corollary 5.29. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and $\mathcal{ALCP}(\mathcal{D})$-concept satisfiability is* NExpTime*-hard.*

$$
\begin{aligned}
\mathsf{Ch}[U_1,U_2,U_3,U_4] &:= (\forall(U_1,U_2).{=} \ \sqcap \ \forall(U_3,U_4).{=}) \\
&\quad \sqcup \bigsqcup_{(\ell_i,r_i) \text{ in } P} \forall(U_1,U_2).\mathsf{conc}_{\ell_i} \sqcap \forall(U_3,U_4).\mathsf{conc}_{r_i} \\
X &:= \exists g_\ell.\mathsf{word} \sqcap \exists g_r.\mathsf{word} \sqcap \exists h_\ell.\mathsf{word} \sqcap \exists h_r.\mathsf{word} \\
&\quad \sqcap \exists R.A \sqcap \exists R.\neg A \sqcap \forall R.\exists S.\top \\
&\quad \sqcap \forall R.\big(A \rightarrow (\exists g_\ell.\mathsf{word} \sqcap \exists g_r.\mathsf{word} \sqcap \exists p_\ell.\mathsf{word} \sqcap \exists p_r.\mathsf{word})\big) \\
&\quad \sqcap \forall R.\big(\neg A \rightarrow (\exists p_\ell.\mathsf{word} \sqcap \exists p_r.\mathsf{word} \sqcap \exists h_\ell.\mathsf{word} \sqcap \exists h_r.\mathsf{word})\big) \\
&\quad \sqcap \forall(g_\ell, Rg_\ell).{=} \sqcap \forall(g_r, Rg_r).{=} \\
&\quad \sqcap \forall(p_\ell, Rp_\ell).{=} \sqcap \forall(p_r, Rp_r).{=} \\
&\quad \sqcap \forall(h_\ell, Rh_\ell).{=} \sqcap \forall(h_r, Rh_r).{=} \\
&\quad \sqcap \forall RS.\big(\exists g_\ell.\mathsf{word} \sqcap \exists g_r.\mathsf{word} \sqcap \exists h_\ell.\mathsf{word} \sqcap \exists h_r.\mathsf{word}\big) \\
&\quad \sqcap \forall R.\big(A \rightarrow (\mathsf{Ch}[g_\ell, Sg_\ell, g_r, Sg_r] \sqcap \exists(Sh_\ell, p_\ell).{=} \ \sqcap \ \exists(Sh_r, p_r).{=})\big) \\
&\quad \sqcap \forall R.\big(\neg A \rightarrow (\mathsf{Ch}[p_\ell, Sg_\ell, p_r, Sg_r] \sqcap \exists(gh_\ell, h_\ell).{=} \ \sqcap \ \exists(Sh_r, h_r).{=})\big) \\
C_P &:= X \sqcap \forall RS.X \sqcap \cdots \sqcap \forall(RS)^{n-2}.X \\
&\quad \sqcap \forall(RS)^{n-1}.(\exists(g_\ell, h_\ell).{=} \ \sqcap \ \exists(g_r, h_r).{=}) \\
&\quad \sqcap \exists x_\ell.\mathsf{word} \sqcap \exists x_r.\mathsf{word} \sqcap \exists y_\ell.\mathsf{word} \sqcap \exists y_r.\mathsf{word} \sqcap \exists z_\ell.\mathsf{word} \sqcap \exists z_r.\mathsf{word} \\
&\quad \sqcap \mathsf{Ch}[h_\ell, x_\ell, h_r, x_r] \\
&\quad \sqcap \mathsf{Ch}[x_\ell, y_\ell, x_r, y_r] \\
&\quad \sqcap \mathsf{Ch}[y_\ell, z_\ell, y_r, z_r] \\
&\quad \sqcap \exists g_\ell, {=_\epsilon} \ \sqcap \exists g_r, {=_\epsilon} \\
&\quad \sqcap \exists z_\ell, z_r.{=} \ \sqcap \ \exists z_\ell. \neq_\epsilon
\end{aligned}
$$

Figure 5.12: The $\mathcal{ALCP}(\mathsf{W})$ reduction concept C_P ($n = |P|$).

Corollary 5.30. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALCP}(\mathcal{D})$-concepts is* NExpTime*-hard.*

5.3.5 $\mathcal{ALC}^{rp}(\mathcal{D})$-concept Satisfiability

In Section 2.3.2, we presented the Description Logic $\mathcal{ALC}^{rp}(\mathcal{D})$, an extension of $\mathcal{ALC}(\mathcal{D})$ with a concrete domain role constructor. We prove that satisfiability of (restricted[6]) $\mathcal{ALC}^{rp}(\mathsf{W})$-concepts without reference to TBoxes is NExpTime-hard. Hence, $\mathcal{ALC}^{rp}(\mathcal{D})$ is the fifth example for an extension of $\mathcal{ALC}(\mathcal{D})$ in which reasoning is much harder than in $\mathcal{ALC}(\mathcal{D})$ itself.

Given a PCP-instance $P = (\ell_1, r_1), \dots, (\ell_k, r_k)$, we define a concept C_P of size polynomial in $|P|$ which has a model iff P has a $2^n + 1$-solution. The concept C_P can

[6] Recall that we generally assume $\mathcal{ALC}^{rp}(\mathcal{D})$ concepts to be in the restricted syntactical form introduced in Section 2.3.2.

$$
\begin{aligned}
\mathsf{DistB}[k] &:= \bigsqcap_{i=0}^{k}((B_i \rightarrow \forall R.B_i) \sqcap (\neg B_i \rightarrow \forall R.\neg B_i)) \\
\mathsf{Tree} &:= \exists R.B_0 \sqcap \exists R.\neg B_0 \\
&\quad \sqcap \forall R.(\mathsf{DistB}[0] \sqcap \exists R.B_1 \sqcap \exists R.\neg B_1) \\
&\quad \vdots \\
&\quad \sqcap \forall R^{n-1}.(\mathsf{DistB}[n-1] \sqcap \exists R.B_{n-1} \sqcap \exists R.\neg B_{n-1}) \\
S[g,p] &:= \exists(g),(g).\overline{p} \\
\mathsf{Edge}[g,p] &= \Big(\bigsqcup_{k=0}^{n-1}\Big(\bigsqcap_{j=0}^{k-1} B_j\Big) \sqcap \Big((B_k \sqcap \forall S[g,p].B_k) \sqcup (\neg B_k \sqcap \forall S[g,p].\neg B_k)\Big) \\
&\quad \sqcup \bigsqcup_{k=0}^{n-1}\Big(\bigsqcup_{j=0}^{k-1} \neg B_j\Big) \sqcap \Big((B_k \sqcap \forall S[g,p].\neg B_k) \sqcup (\neg B_k \sqcap \forall S[g,p].B_k)\Big)\Big) \\
\mathsf{DEdge} &:= (\mathsf{Edge}[g_\ell, =] \sqcap \mathsf{Edge}[g_r, =]) \sqcup \\
&\quad \bigsqcup_{(\ell_i,r_i) \text{ in } P} (\mathsf{Edge}[g_\ell, \mathsf{conc}_{\ell_i}] \sqcap \mathsf{Edge}[g_r, \mathsf{conc}_{r_i}]) \\
\mathsf{Ch}[u_1,u_2,u_3,u_4] &:= (\exists(u_1,u_2).= \;\sqcap \exists(u_3,u_4).=) \\
&\quad \sqcup \bigsqcup_{(\ell_i,r_i) \text{ in } P} (\exists(u_1,u_2).\mathsf{conc}_{\ell_i} \sqcap \exists(u_3,u_4).\mathsf{conc}_{r_i}) \\
C_P &:= \mathsf{Tree} \sqcap \forall R^n.\exists g_\ell.\mathsf{word} \sqcap \forall R^n.\exists g_r.\mathsf{word} \\
&\quad \sqcap \forall R^n.\big[(\neg B_0 \sqcap \cdots \sqcap \neg B_{n-1}) \rightarrow (\exists g_\ell.=_\epsilon \;\sqcap \exists g_r.=_\epsilon) \\
&\qquad \sqcap \neg(B_0 \sqcap \cdots \sqcap B_{n-1}) \rightarrow \mathsf{DEdge} \\
&\qquad \sqcap (B_0 \sqcap \cdots \sqcap B_{n-1}) \rightarrow \\
&\qquad\qquad \big(\mathsf{Ch}(g_\ell, f_1g_\ell, g_r, f_1g_r) \sqcap \mathsf{Ch}(f_1g_\ell, f_2g_\ell, f_1g_r, f_2g_r) \\
&\qquad\qquad \sqcap\; \exists(f_2g_\ell),(f_2g_r).= \sqcap \exists(f_2g_\ell).\neq_\epsilon\big)\big]
\end{aligned}
$$

Figure 5.13: The $\mathcal{ALC}^{rp}(\mathsf{W})$ reduction concept C_P ($n = |P|$).

be found in Figure 5.13, where p denotes a predicate (written in lowercase to avoid confusion with the PCP-instance P). Note that $S[g,p]$ denotes a predicate role and not a concept, i.e., $S[g,p]$ is an abbreviation for the role-forming concrete domain constructor $\exists(g),(g).\overline{p}$.

Figure 5.14 contains an example model of C_P with $|P| = n = 2$. Obviously, the models of C_P are rather similar to the ones from the reduction in Section 5.3.1: models have the form of a binary tree of depth n whose leaves—together with two "extra" nodes—are connected by two predicate chains of length $2^n + 1$. However, in contrast to the reduction in Section 5.3.1, the edges of the tree are labeled with the role R and not with abstract features ℓ and r. The Tree concept enforces the existence of the binary tree. The concept names $B_0, \ldots, B_{n-1}$ are used for a binary numbering (from

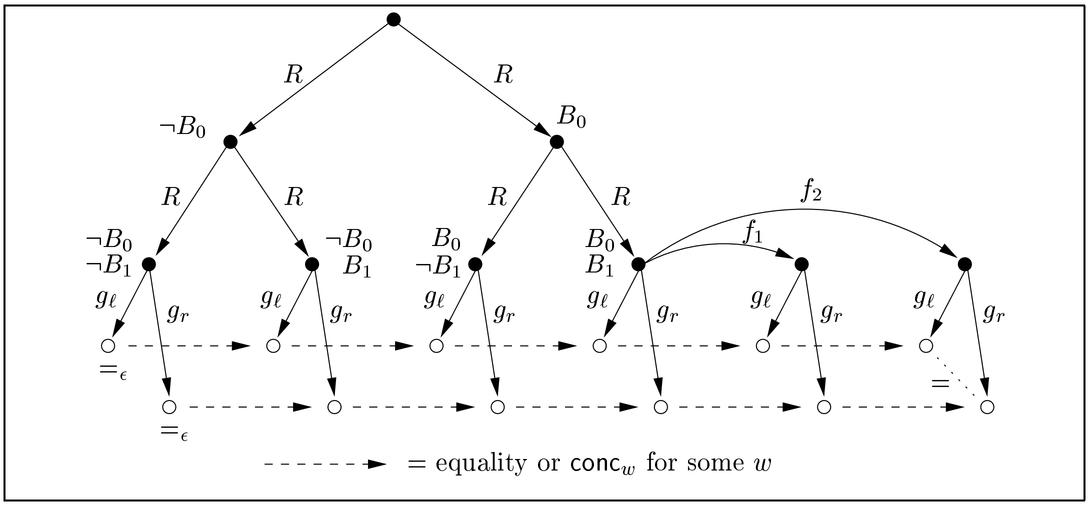

Figure 5.14: An example model of C_P with $|P| = 2$.

0 to $2^n - 1$) of the leaves of the tree. More precisely, for a domain object $d \in \Delta^\mathcal{I}$, set

$$\mathsf{pos}(d) = \Sigma_{i=0}^{n-1} \beta_i(d) \cdot 2^i \quad \text{where} \quad \beta_i(d) = \begin{cases} 1 & \text{if } d \in B_i^\mathcal{I} \\ 0 & \text{otherwise.} \end{cases}$$

The Tree and DistB concepts ensure that, for each i with $0 \leq i < 2^n$, there exists a leaf d in the tree such that $\mathsf{pos}(d) = i$. This numbering will be used to establish the two predicate chains. Due to the first line of the C_P concept, every leaf has (concrete) g_ℓ- and g_r-successors. Lines 4 and 5 of C_P guarantee the existence of the two extra nodes which are connected by predicate edges due to the use of the Ch concept. Hence, it remains to describe how the predicate edges between the leaf nodes are established.

There are two main ideas underlying the establishment of these edges: (i) use the role-forming predicate constructor to establish single edges and (ii) use the position $\mathsf{pos}()$ of leaf nodes together with the fact that counting modulo 2^n can be expressed by $\mathcal{ALC}$-concepts to do this with a concept of size polynomial in $|P|$. We first illustrate Point (i). Assume that we have two abstract objects d and e, d has a g_ℓ-successor x, and e has a g_ℓ-successor y. Moreover, let $e \in X^\mathcal{I}$ for some concept X. We may then establish a p-edge (for some binary predicate $p \in \Phi_\mathsf{W}$) between x and y by enforcing that $d \in (\forall S[g_\ell, p].\neg X)^\mathcal{I}$: since $e \in X^\mathcal{I}$, it follows that $(d,e) \notin S[g_\ell, p]^\mathcal{I}$, i.e., $(d,e) \notin (\exists (g_\ell), (g_\ell).\overline{p})^\mathcal{I}$ and thus $(x,y) \notin \overline{p}^\mathsf{W}$, which obviously implies that $(x,y) \in p^\mathsf{W}$.

Now for Point (ii) from above. In the third line of the C_P-concept, the DEdge concept is used to establish edges between the leaf nodes. The DEdge concept itself is just a disjunction over the various edge types while the Edge concept actually establishes the edges. In principle, it does this in the way described above. However, the Edge concept is not only used for two *fixed* nodes d and e together with a fixed concept X but establishes the edges between *all* neighboring leaf nodes. This is achieved by exploiting the binary numbering of the leaf nodes: the Edge concept is essentially the negation of the well-known propositional formula

$$\bigwedge_{k=0}^{n-1}(\bigwedge_{j=0}^{k-1} x_j = 1) \rightarrow (x_k = 1 \leftrightarrow x'_k = 0) \;\wedge\; \bigwedge_{k=0}^{n-1}(\bigvee_{j=0}^{k-1} x_j = 0) \rightarrow (x_k = x'_k)$$

which encodes incrementation modulo 2^n, i.e., if t is the number (binarily) encoded by the propositional variables $x_0, \ldots, x_{n-1}$ and t' is the number encoded by the propositional variables $x'_0, \ldots, x'_{n-1}$, then we have $t' = t + 1$ modulo 2^n, c.f. [Börger *et al.* 1997]. Let d be a leaf of the tree with $d \in (\mathsf{Edge}[g_\ell, p])^\mathcal{I}$ (where p is "=", conc_{ℓ_i}, or conc_{r_i} for some i), e a leaf with $\mathsf{pos}(e) = \mathsf{pos}(d) + 1$, x the g_ℓ-successor of d, and y the g_ℓ-successor of e. The Edge concept ensures that, for each $S[g_\ell, p]$-successor d' of d, we have $\mathsf{pos}(d') \neq \mathsf{pos}(d) + 1$, i.e., there exists an i with $0 \leq i \leq n$ such that d' differs from e in the interpretation of B_i. It follows that $(d, e) \notin S[g_\ell, p]^\mathcal{I}$. As described above, we can conclude $(x, y) \in p^\mathcal{I}$. All remaining issues, such as ensuring that the last pair of words on the predicate chains is in fact a solution to P, are as in the reduction given in Section 5.3.1. Note that the reduction concept is restricted in the sense of Section 2.3.2: for all subconcepts $\forall S[g, p].D$ of the reduction concept C_P, D is either a concept name or the negation of a concept name.

Lemma 5.31. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP-instance. Then P has a $2^n + 1$-solution iff the concept C_P is satisfiable.*

Proof. Let $|P| = n \geq 2$. First assume that C_P is satisfiable. Using induction over n and the definitions of the Tree and DistB concepts, it is easy to show that there exist domain elements $d_{i,j}$ for $0 \leq i \leq n$ and $0 \leq j < 2^i$ such that

1. $\{(d_{i,j}, d_{(i+1),2j}), (d_{i,j}, d_{(i+1),(2j+1)})\} \subseteq R^\mathcal{I}$ for $i < n$ and $j < 2^i$ and
2. $\mathsf{pos}(d_{n,j}) = j$ for $j < 2^n$.

The first property states that the $d_{i,j}$ form a binary tree whose edges are labeled by R. The naming scheme for nodes is as indicated in Figure 5.5. By the first line of the C_P concept, there exist words $x_0, \ldots, x_{2^n-1}$ and $y_0, \ldots, y_{2^n-1}$ such that

$$g_\ell^\mathcal{I}(d_{n,j}) = x_j \text{ and } g_r^\mathcal{I}(d_{n,j}) = y_j \text{ for } j < 2^n.$$

By the third line of C_P, we have $d_{n,j} \in \mathsf{DEdge}^\mathcal{I}$ for all $d_{n,j}$ with $\mathsf{pos}(d_{n,j}) \neq 2^n - 1$, i.e., for all $d_{n,j}$ with $j < 2^n - 1$. By definition of DEdge, for each $j < 2^n - 1$, we have either

$$d_{n,j} \in (\mathsf{Edge}[g_\ell, =] \sqcap \mathsf{Edge}[g_r, =])^\mathcal{I}$$

or there exists a pair $(\ell_i, r_i) \in P$ such that

$$d_{n,j} \in (\mathsf{Edge}[g_\ell, \mathsf{conc}_{\ell_i}] \sqcap \mathsf{Edge}[g_r, \mathsf{conc}_{r_i}])^\mathcal{I}.$$

As already argued in the intuitive explanations, the first property implies $x_j = x_{j+1}$ and $y_j = y_{j+1}$ while the second implies $(x_j, x_{j+1}) \in \mathsf{conc}_{\ell_i}^\mathsf{W}$ and $(y_j, y_{j+1}) \in \mathsf{conc}_{r_i}^\mathsf{W}$ (we refrain from repeating the arguments here). Hence, there exist indexes $i_1, \ldots, i_{2^n-1} \in \{1, \ldots, k\} \cup \{\clubsuit\}$ such that, for $0 < j \leq 2^n - 1$, we have

- if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
- $(x_{j-1}, x_j) \in \mathsf{conc}^{\mathsf{W}}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in \mathsf{conc}^{\mathsf{W}}_{r_{i_j}}$ otherwise.

By definition of the Ch concept and Lines 4 and 5 of the definition of C_P, there exist domain elements $d_{n,2^n}$ and $d_{n,(2^n+1)}$, words $x_{2^n}, x_{2^n+1}, y_{2^n}, y_{2^n+1}$, and indexes $i_{2^n}, i_{2^n+1} \in \{1, \dots, k\} \cup \{\clubsuit\}$ such that

1. $f_1^{\mathcal{I}}(d_{n,2^n-1}) = d_{n,2^n}$ and $f_2^{\mathcal{I}}(d_{n,2^n-1}) = d_{n,(2^n+1)}$,
2. $g_\ell^{\mathcal{I}}(d_{n,i}) = x_i$ and $g_r^{\mathcal{I}}(d_{n,i}) = y_i$ for $i \in \{2^n, 2^n + 1\}$,
3. for $j \in \{2^n, 2^n + 1\}$,
   - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
   - $(x_{j-1}, x_j) \in \mathsf{conc}^{\mathsf{W}}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in \mathsf{conc}^{\mathsf{W}}_{r_{i_j}}$ otherwise.

By the second and last line of the definition of C_P, we have $x_0 = y_0 = \epsilon$ and $x_{2^n+1} = y_{2^n+1} \neq \epsilon$. Taking together these observations, it is clear that the sequence $i'_1, \dots, i'_p$, which can be obtained from $i_1, \dots, i_{2^n+1}$ by eliminating all i_j with $i_j = \clubsuit$, is a solution for P. Furthermore, we obviously have $1 \leq p \leq 2^n + 1$.

Now for the "only if" direction. Assume that P has a solution $i_1, \dots, i_m$ with $m \leq 2^n + 1$. By L_j (resp. R_j), we denote the concatenation $\ell_{i_1} \cdots \ell_{i_j}$ (resp. $r_{i_1} \cdots r_{i_j}$) for $1 \leq j \leq m$ and set $L_0 = R_0 = \epsilon$ and $L_j = L_m$ (resp. $R_j = R_m$) for $j > m$. We define a model $\mathcal{I}$ for C_P with the form of a binary tree of depth n. Figure 5.5 indicates the naming scheme used. Set

$$\Delta^{\mathcal{I}} := \{d_{i,j} \mid 0 \leq i \leq n, 0 \leq j < 2^i\} \cup \{d_{n,2^n}, d_{n,(2^n+1)}\}.$$

For $j < n$, $B_j^{\mathcal{I}}$ is the smallest superset S of $\{d_{(j+1),i} \mid 0 \leq i < 2^j \text{ and } i \bmod 2 \neq 0\}$ which is closed under the following condition:

$$d_{i,j} \in S \text{ and } i < n \text{ implies } d_{(i+1),(2j)}, d_{(i+1),(2j+1)} \in S.$$

Now for the interpretation of the roles and concrete features.

$$R^{\mathcal{I}} := \{(d_{i,j}, d_{(i+1),(2j)}), (d_{i,j}, d_{(i+1),(2j+1)}) \mid i < n \text{ and } j < 2^i\}$$

$$\text{Set } f_1^{\mathcal{I}}(d_{n,(2^n-1)}) := d_{n,2^n} \text{ and } f_2^{\mathcal{I}}(d_{n,(2^n-1)}) := d_{n,(2^n+1)}.$$

$$\text{For } i \leq 2^n + 1, \text{ set } g_\ell^{\mathcal{I}}(d_{n,i}) := L_i \text{ and } g_r^{\mathcal{I}}(d_{n,i}) := R_i.$$

It is not hard to verify that $\mathcal{I}$ is a model of C_P. ❑

The size of C_P is polynomial in $|P|$ and C_P can be constructed in time polynomial in $|P|$.

Theorem 5.32. *$\mathcal{ALC}^{rp}(\mathsf{W})$-concept satisfiability is* NEXPTIME*-hard.*

Once more, we obtain two corollaries:

Corollary 5.33. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTIME *and $\mathcal{ALC}^{rp}(\mathcal{D})$-concept satisfiability is* NEXPTIME*-hard.*

Corollary 5.34. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALC}^{rp}(\mathcal{D})$-concepts is* NEXPTIME*-hard.*

Of course, we again obtain corresponding co-NEXPTIME lower bounds for concept subsumption.

5.4 The Upper Bound

We combine the five extensions of $\mathcal{ALC}(\mathcal{D})$ into a single Description Logic and prove a NEXPTIME upper complexity bound matching the lower bounds established in the previous sections. For this upper bound, we concentrate on admissible concrete domains $\mathcal{D}$ for which $\mathcal{D}$-satisfiability is in NP. This captures a large class of interesting concrete domains such as the concrete domain W used to prove the lower bounds and the temporal concrete domains P and $\mathtt{I}$ from Section 2.4.3. Note that, unlike in Chapter 3, we do *not* consider concrete domains $\mathcal{D}$ for which $\mathcal{D}$-satisfiability is in PSPACE. The reason is that, here, we are heading for a time complexity bound rather than for a space complexity bound.

Let us introduce $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$, the Description Logics obtained by combining the five extensions of $\mathcal{ALC}(\mathcal{D})$, in detail. We start with defining the roles that may be used inside existential and universal value restrictions.

Definition 5.35 ($\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-roles). The set of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-roles $\mathcal{R}$ is defined inductively as follows:

- each element of $\mathsf{N_R}$ is an $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-role,
- each predicate role (c.f. Section 2.3.2) is an $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-role, and
- if R and R' are $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-roles, then R^- and $R \sqcap R'$ are $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-roles.

Predicate roles and inverses of predicate roles are called *complex* roles. For each $R \in \mathcal{R}$, $\mathsf{Inv}(R)$ denotes R^- and $\mathsf{Inv}(R^-)$ denotes R. $\diamond$

The semantics of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-roles is defined in the obvious way and we omit the details (see Sections 2.1.2 and 2.3.2). In what follows, we generally assume $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-roles to be of the form $R_1 \sqcap \cdots \sqcap R_n$, where the R_i are role names, inverses of role names, or complex roles. Clearly, every element of $\mathcal{R}$ can be converted into this form by exhaustively applying the transformations

$$(R_1 \sqcap \cdots \sqcap R_n)^- \rightsquigarrow (R_1^- \sqcap \cdots \sqcap R_n^-) \quad \text{and} \quad (R^-)^- \rightsquigarrow R.$$

The Description Logic $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$ is obtained from $\mathcal{ALC}(\mathcal{D})$ by allowing the use of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-roles in existential and universal value restrictions and admitting the generalized concrete domain constructors $\exists U_1, \ldots, U_n.P$ and $\forall U_1, \ldots, U_n.P$ introduced in Section 2.3.2. Recall that the U_i are role paths, i.e. sequences $R_1 \cdots R_n g$, where $R_1, \ldots, R_n \in \mathsf{N_R}$ and $g \in \mathsf{N_{aF}}$. Hence, the role conjunction constructor, the inverse role constructor, and predicate roles are not allowed inside role paths. Since concrete paths are (a special case of) role paths, we do not distinguish between the standard version and the generalized version of the concrete domain constructor. Note, however, that role paths are only allowed inside the concrete domain concept constructor and not inside the concrete domain role constructor.

Since $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$ includes the concrete domain role constructor, we must restrict its syntax to avoid inheriting undecidability from $\mathcal{ALC}^{rp}(\mathcal{D})$ (see Section 2.3.2). As for $\mathcal{ALC}^{rp}(\mathcal{D})$, we first need a procedure to convert $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts to NNF.

Fact 5.36. *Every $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept can be converted into an equivalent one in NNF by exhaustively applying the following rewrite rules:*

$$
\begin{array}{rclrcl}
\neg\neg C & \rightsquigarrow & C & \neg(g\uparrow) & \rightsquigarrow & \exists g.\top_{\mathcal{D}} \\
\neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D & \neg(C \sqcup D) & \rightsquigarrow & \neg C \sqcap \neg D \\
\multicolumn{6}{l}{\neg(\exists(R_1 \sqcap \cdots \sqcap R_n).C) \rightsquigarrow \forall(R_1 \sqcap \cdots \sqcap R_n).\neg C} \\
\multicolumn{6}{l}{\neg(\forall(R_1 \sqcap \cdots \sqcap R_n).C) \rightsquigarrow \exists(R_1 \sqcap \cdots \sqcap R_n).\neg C} \\
\multicolumn{6}{l}{\neg(\exists U_1, \ldots, U_n.P) \rightsquigarrow \forall U_1, \ldots, U_n.\overline{P}} \\
\multicolumn{6}{l}{\neg(\forall U_1, \ldots, U_n.P) \rightsquigarrow \exists U_1, \ldots, U_n.\overline{P}}
\end{array}
$$

We can now define restricted concepts.

Definition 5.37 (Restricted $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept). An $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept C is called *restricted* iff the result C' of converting C to NNF satisfies the following conditions:

1. For any $\forall(R_1 \sqcap \cdots \sqcap R_n).D \in \mathsf{sub}(C')$, where all $R_1, \ldots, R_n$ are complex roles,
   (a) $\mathsf{sub}(D)$ does not contain any concepts $\exists(S_1 \sqcap \cdots \sqcap S_m).E$ such that some S_i is a complex role, and
   (b) $\mathsf{sub}(D)$ contains no concepts $\exists U_1, \ldots, U_n.P$ or $\forall U_1, \ldots, U_n.P$
2. For any $\exists(R_1 \sqcap \cdots \sqcap R_n).D \in \mathsf{sub}(C')$, where all $R_1, \ldots, R_n$ are complex roles,
   (a) $\mathsf{sub}(D)$ does not contain any concepts $\forall(S_1 \sqcap \cdots \sqcap S_m).E$ such that some S_i is a complex role, and
   (b) $\mathsf{sub}(D)$ contains no concepts $\exists U_1, \ldots, U_n.P$ or $\forall U_1, \ldots, U_n.P$

◇

In what follows, we generally assume that $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts are restricted without further notice. Note that converting a restricted $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept into NNF does again yield a restricted $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept. We may thus w.l.o.g. assume $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts to be in NNF. Also observe that the restrictions 1b and 2b from above are slightly stronger than the corresponding ones for restricted

$\mathcal{ALC}^{rp}(\mathcal{D})$-concepts presented in Section 2.3.2: in Definition 2.13, we admit the occurrence of concepts $\exists f_1, \dots, f_n.P$ in $\mathsf{sub}(D)$ which is not allowed in Definition 5.37. The reason is that, when constructing a completion algorithm for $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$ with the weaker restrictions, one runs into termination problems. Consider, for example, the concept

$$\exists g.\top_{\mathcal{D}} \sqcap \exists f^{-}.\top \sqcap \forall(\exists(g),(fg).\top_2).(\exists g.\top_{\mathcal{D}} \sqcap \exists f^{-}.\top)$$

where $\top_2^{\mathcal{D}} = \Delta_{\mathcal{D}} \times \Delta_{\mathcal{D}}$. A straightforward completion algorithm would generate an infinite "f^{-}-chain" of objects, each having a concrete g-successor. In fact, it seems rather easy to prove undecidability of $\mathcal{ALC}^{rp,-}(\mathcal{D})$ with the weaker restrictions (role conjunction is not needed) using a technique similar to the one used in [Lutz & Möller 1997] to show undecidability of unrestricted $\mathcal{ALC}^{rp}(\mathcal{D})$.

5.4.1 The Completion Algorithm

We establish the NExpTime upper bound using a completion algorithm. To simplify presentation, we first treat $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept satisfiability without reference to TBoxes and, in a second step, modify this algorithm to take into account acyclic TBoxes. In what follows, we generally assume that $\mathcal{D}$ is an admissible concrete domain.

As usual, the completion algorithm expects the input concept to be in NNF. Since models constructed by the completion algorithm have the form of a tree, we reflect this explicitly in the data structure instead of using ABoxes as in Chapter 3.

Definition 5.38 (Completion System). A *completion tree* for an $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept D is a tree whose set of nodes is a subset of $\mathsf{O_a} \uplus \mathsf{O_c}$ such that all nodes from $\mathsf{O_c}$ are leaves. In this context, we call elements of $\mathsf{O_a}$ *abstract nodes* and elements of $\mathsf{O_c}$ *concrete nodes*. The tree is labeled as follows:

1. each node $a \in \mathsf{O_a}$ is labeled with a subset $\mathcal{L}(a)$ of $\mathsf{sub}(D)$;
2. each edge (a, b) with $a, b \in \mathsf{O_a}$ is labeled with a set $\mathcal{L}(a, b)$ of roles from $\mathcal{R}$ occurring in D;
3. each edge (a, x) with $a \in \mathsf{O_a}$ and $x \in \mathsf{O_c}$ is labeled with a concrete feature $\mathcal{L}(a, x)$ occurring in D.

A *completion system* for an $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept D is a pair $(\mathbf{T}, \mathcal{P})$, where $\mathbf{T}$ is a completion tree for D and $\mathcal{P}$ is a function mapping each $P \in \Phi_{\mathcal{D}}$ with arity n appearing in D to a subset of $(\mathsf{O_c})^n$. $\diamond$

To simplify the formulation of the completion rules and the proofs, it is convenient to define several notions of successorship and neighborhood in completion trees.

Definition 5.39 (Successor, Neighbor, Relative). Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system, a and b abstract nodes in $\mathbf{T}$, and x a concrete node in $\mathbf{T}$. Moreover, let R be a role name, the inverse of a role name, or a complex role, and let $g \in \mathsf{N_{aF}}$. Then

- b is an *R-successor* of a in $\mathbf{T}$ iff b is a successor of a and $R \in \mathcal{L}(a, b)$;
- x is a *g-successor* of a in $\mathbf{T}$ iff x is successor of a and $\mathcal{L}(a, x) = g$;
- b is an *R-neighbor* of a iff either b is an R-successor of a or a is an $\mathsf{Inv}(R)$-successor of b.

It is straightforward to extend the notion "neighbor" to role paths: let $U = R_1 \cdots R_n g$ be a role path. Then x is a *U-neighbor* of a in $\mathbf{T}$ iff there exist nodes $b_1, \ldots, b_n \in \mathsf{O_a}$ such that b_1 is an R_1-neighbor of a, b_i is an R_i-neighbor of b_{i-1} for $1 < i \leq n$, and x is a g-successor of b_n. By $\mathsf{neighb}_{\mathbf{T}}(a, U)$, we denote the set of U-neighbors of a in $\mathbf{T}$. The index $\mathbf{T}$ is omitted if clear from the context.

To deal with complex roles, it is convenient to further generalize the notion "neighbor" into the notion "relative". For non-complex roles $R \in \mathsf{N_R} \cup \{R^- \mid R \in \mathsf{N_R}\}$, these two notions coincide. Let $S = \exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P$ be a predicate role. Then b is an *S-relative* of a iff either b is an S-neighbor of a or there exist concrete domain elements $x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O_c}$ such that

1. $x_i \in \mathsf{neighb}_{\mathbf{T}}(a, u_i)$ for $1 \leq i \leq n$,
2. $y_i \in \mathsf{neighb}_{\mathbf{T}}(b, v_i)$ for $1 \leq i \leq m$, and
3. $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$.

Moreover, b is an *S^--relative* of a iff a is an S-relative of b.[7]

We now generalize these notions to conjunctions of roles. For $R_1 \sqcap \cdots \sqcap R_n \in \mathcal{R}$, b is an *$(R_1 \sqcap \cdots \sqcap R_n)$-successor* / *$(R_1 \sqcap \cdots \sqcap R_n)$-neighbor* / *$(R_1 \sqcap \cdots \sqcap R_n)$-relative* of a iff b is an R_i-successor / R_i-neighbor / R_i-relative of a for $1 \leq i \leq n$.

$\diamond$

If the satisfiability of a concept D is to be decided, the completion algorithm is started with the *initial completion system* $S_D = (\mathbf{T}_D, \mathcal{P}_\emptyset)$, where $\mathbf{T}_D$ is the tree consisting of a single node a with $\mathcal{L}(a) = \{D\}$ and $\mathcal{P}_\emptyset$ maps each $P \in \Phi_\mathcal{D}$ occurring in D to $\emptyset$. The algorithm repeatedly applies the non-deterministic completion rules until either it finds a completion system to which no more rules are applicable or it finds a completion system containing a contradiction. If the rules can be applied such that the final completion system does not contain a contradiction, then this final completion system represents a model of D, and thus D is satisfiable. Otherwise, D is unsatisfiable. As in Chapter 3, we define a "+" operation for introducing new nodes, which facilitates the succinct presentation of the completion rules.

Definition 5.40 ("+" operation). An abstract or concrete node is called *fresh* w.r.t. a completion tree $\mathbf{T}$ if it does not appear in $\mathbf{T}$. Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system. The operation

$$S + a(R_1 \sqcap \cdots \sqcap R_n)b$$

where $a \in \mathsf{O_a}$ is a node in $\mathbf{T}$, $b \in \mathsf{O_a}$ is fresh in $\mathbf{T}$, and $R_1 \sqcap \cdots \sqcap R_n \in \mathcal{R}$, (non-deterministically) yields a completion system that can be obtained from S by either

[7] It is easily checked that "S^--relative" subsumes "S^--neighbor" although this is not explicitly stated.

1. augmenting $\mathbf{T}$ with a new successor b of a and setting $\mathcal{L}(a,b) := \{R_1,\ldots,R_n\}$ and $\mathcal{L}(b) := \emptyset$ or
2. choosing a neighbor c of a in $\mathbf{T}$, renaming c in $\mathbf{T}$ with b, and then
   - setting $\mathcal{L}(a,b) := \mathcal{L}(a,b) \cup \{R_1,\ldots,R_n\}$ if b is a successor of a in $\mathbf{T}$ and
   - setting $\mathcal{L}(b,a) := \mathcal{L}(b,a) \cup \{\mathsf{Inv}(R_1),\ldots,\mathsf{Inv}(R_n)\}$ if a is a successor of b in $\mathbf{T}$.

By $S + agx$, where $x \in \mathsf{O_c}$ is fresh in $\mathbf{T}$ and $g \in \mathsf{N_{cF}}$, we denote the completion system $S' = (\mathbf{T}', \mathcal{P}')$ that can be obtained from S as follows:

1. if a has no g-successor, then augment $\mathbf{T}$ with a new successor x of a and set $\mathcal{L}(a,x) := g$;
2. if a already has a g-successor y, then replace y in $\mathbf{T}$ and $\mathcal{P}$ by x.

When nesting the $+$-operation, we omit brackets writing, e.g., $S + aR_1b + bR_2c$ for $(S + aR_1b) + bR_2c$. Let $U = R_1 \cdots R_n g$ be a role path. By $S + aUx$, where x is fresh in S, we denote the completion system S' which can be obtained from S by choosing distinct objects $b_1,\ldots,b_n \in \mathsf{O_a}$ which are fresh in S and setting

$$S' := S + aR_1b_1 + b_1R_2b_2 + \cdots + b_{n-1}R_nb_n + b_ngx.$$

◇

Intuitively, the above version of the "+"-operation allows to "reuse" nodes when generating successors. For example, let $S = (\mathbf{T}, \mathcal{P})$ be a completion system and a be an abstract node in $\mathbf{T}$ having a successor b. If $S + aRc$ is executed, the "+"-operation may either make the existing node b an R-successor of a by extending the edge label $\mathcal{L}(a,b)$ with R, or it may generate a new R-successor c of a. Together with an appropriate clash condition, this approach replaces fork elimination (c.f. Section 3.1.2). More precisely, it guarantees that every abstract node has at most a single successor for each abstract and concrete feature at any given time: assume for example that, in the above situation, we have $R = f$ for some $f \in \mathsf{N_{aF}}$ and that $f \in \mathcal{L}(a,b)$ before the "+"-operation was executed. In this case, it is obviously desirable to reuse the node b instead of generating a new one. The wrong choice, i.e., the generation of a new R-successor, will lead to a clash (see below).

The non-determinism introduced by the "+"-operation is true "don't know" non-determinism. Thus, the completion algorithm to be devised is a non-deterministic algorithm. Why don't we use a fork elimination rule such as in Section 3.1.2 or even integrate fork elimination into the other rules as done in [Baader & Hanschke 1991a]? Both approaches are problematic due to complex interactions between features, role conjunction, and the inverse role constructor. For example, we can express feature agreements for abstract paths of length one by writing $\exists f \sqcap f'.\top$. Moreover, due to the presence of inverses, such agreements may be "detected rather late". For example, a straightforward completion algorithm started on the concept

$$\exists f.\top \sqcap \exists f'.\top \sqcap \forall f.\forall f^-.\exists(f \sqcap f').\top$$

R$\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{L}(a)$, $C_1 \notin \mathcal{L}(a)$, or $C_2 \notin \mathcal{L}(a)$ then $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C_1, C_2\}$
R$\sqcup$	if $C_1 \sqcup C_2 \in \mathcal{L}(a)$, $C_1 \notin \mathcal{L}(a)$ and $C_2 \notin \mathcal{L}(a)$ then $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C\}$ for some $C \in \{C_1, C_2\}$
R$\exists$	if $\exists(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ and, for all $(R_1 \sqcap \cdots \sqcap R_n)$-relatives b of a, $C \notin \mathcal{L}(b)$ then set $S := S + a(R_1 \sqcap \cdots \sqcap R_n)b$ for a fresh $b \in \mathsf{O_a}$ and $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$
R$\forall$	if $\forall(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$, b is an $(R_1 \sqcap \cdots \sqcap R_n)$-relative of a, and $C \notin \mathcal{L}(b)$ then set $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$
R$\exists$c	if $\exists U_1, \ldots, U_n.P \in \mathcal{L}(a)$ and there exist no $x_1, \ldots, x_n \in \mathsf{O_c}$ such that $x_i \in \mathsf{neighb}(a, U_i)$ for $1 \leq i \leq n$ and $(x_1, \ldots, x_n) \in \mathcal{P}(P)$ then set $S := (S + aU_1x_1 + \cdots + aU_nx_n)$ with $x_1, \ldots, x_n \in \mathsf{O_c}$ fresh in S and $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, \ldots, x_n)\}$
R$\forall$c	if $\forall U_1, \ldots, U_n.P \in \mathcal{L}(a)$ and there exist $x_1, \ldots, x_n \in \mathsf{O_c}$ such that $x_i \in \mathsf{neighb}(a, U_i)$ for $1 \leq i \leq n$ and $(x_1, \ldots, x_n) \notin \mathcal{P}(P)$ then set $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, \ldots, x_n)\}$
Rpr	if b is $\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P$-neighbor of a and there exist no $x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O_c}$ such that $x_i \in \mathsf{neighb}(a, u_i)$ for $1 \leq i \leq n$, $y_i \in \mathsf{neighb}(b, v_i)$ for $1 \leq i \leq m$, and $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$ then set $S := (S + au_1x_1 + \cdots + au_nx_n + bv_1y_1 + \cdots + bv_my_m)$ with $x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O_c}$ fresh in S and set $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, \ldots, x_n, y_1, \ldots, y_m)\}$
Rch	if $\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P$ occurs in D, $x_i \in \mathsf{neighb}(a, u_i)$ for $1 \leq i \leq n$, $y_i \in \mathsf{neighb}(b, v_i)$ for $1 \leq i \leq m$, and $(x_1, \ldots, x_n, y_1, \ldots, y_m) \notin \mathcal{P}(P) \cup \mathcal{P}(\overline{P})$ then set $\mathcal{P}(P') := \mathcal{P}(P') \cup \{(x_1, \ldots, x_n, y_1, \ldots, y_m)\}$ for a $P' \in \{P, \overline{P}\}$

Figure 5.15: Completion rules for $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$ on input D.

would first generate distinct f- and f'-successors and later identify them. This and similar effects make brute force guessing as implemented by the "+"-operation the most manageable approach to fork elimination.

The completion rules can be found in Figure 5.15. Some explanatory notes are in order. The rules R$\sqcap$, R$\sqcup$, R$\exists$, R$\forall$, and R$\exists$c are straightforward generalizations of the corresponding rules from Figures 3.2 and 3.5. R$\forall$c deals with the universal version of the (generalized) concrete domain concept constructor and Rpr closely resembles R$\exists$c but deals with predicate roles. The Rch rule is a "choose rule" (c.f., e.g., [Baader *et al.* 1996]) that is necessary to ensure completeness of the algorithm in the presence of predicate roles. With "S occurs in D" in the Rch rule, we mean that the predicate

```
define procedure sat(S)
   if S contains a clash then
      return unsatisfiable
   if S is complete then
      return satisfiable
   Apply a completion rule to S yielding S'
   return sat(S')
```

Figure 5.16: The $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$ completion algorithm.

role S is used (directly or as inverse) in the input concept D. Note that the R$\sqcup$ and Rch rules are non-deterministic.

The notion "clash" formalizes what it means for a completion system to be contradictory.

Definition 5.41 (Clash). Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system for a concept D. S is *concrete domain satisfiable* iff the conjunction

$$\zeta_{\mathcal{P}} = \bigwedge_{P \text{ used in } D} \bigwedge_{(x_1,\ldots,x_n)\in\mathcal{P}(P)} (x_1, \ldots, x_n) : P$$

is satisfiable. S is said to contain a *clash* iff there exist $a, b, c \in \mathsf{O_a}$ such that

1. $\{A, \neg A\} \subseteq \mathcal{L}(a)$ for some concept name A,
2. both b and c are f-neighbors of a for some $f \in \mathsf{N_{aF}}$ and $b \neq c$,
3. $g\uparrow \in \mathcal{L}(a)$ and there exists an $x \in \mathsf{O_c}$ such that x is g-successor of a, or
4. S is not concrete domain satisfiable.

If S does not contain a clash, S is called *clash-free*. S is called *complete* iff no completion rule is applicable to S. ◇

As discussed above, "wrong" decisions concerning the reuse of neighbors made by the non-deterministic "+"-operation need to be ruled out by an appropriate clash condition. This is done by Condition 2 in Definition 5.41. An analogous condition for concrete features is not necessary since the "+"-operation will obviously never generate more than one successor per concrete feature.

The completion algorithm itself can be found in Figure 5.16. If a completion system S' can be obtained by repeated rule application to another completion system S, then we call S' *derivable* from S.

5.4.2 Termination, Soundness, and Completeness

To prove termination in a succinct way, it is convenient to introduce several auxiliary notions. We start with defining the role depth of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts (c.f. Section 2.1.1) and generalizing it to sets of concepts.

- The *role depth* of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts C is denoted by $\mathsf{rd}(C)$ and defined inductively as follows (for technical reasons, we also define the role depth of *roles*):

  1. $\mathsf{rd}(A) = \mathsf{rd}(\neg A) = 0$ for concept names A,
  2. $\mathsf{rd}(R) = 0$ for role names R,
  3. $\mathsf{rd}(\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P)$ is the length of the longest path among $u_1, \ldots, u_n, v_1, \ldots, v_m$ (where the length of a concrete path $u = f_1 \cdots f_n g$ is $n + 1$),
  4. $\mathsf{rd}(R^-) = \mathsf{rd}(R)$,
  5. $\mathsf{rd}(C_1 \sqcap C_2) = \mathsf{rd}(C_1 \sqcup C_2) = \max(\mathsf{rd}(C_1), \mathsf{rd}(C_2))$,
  6. $\mathsf{rd}(\exists(R_1 \sqcap \cdots \sqcap R_n).C) = \mathsf{rd}(\forall(R_1 \sqcap \cdots \sqcap R_n).C) = \max(\mathsf{rd}(R_1), \ldots, \mathsf{rd}(R_n), \mathsf{rd}(C) + 1)$,
  7. $\mathsf{rd}(\exists U_1, \ldots, U_n.P)$ and $\mathsf{rd}(\forall U_1, \ldots, U_n.P)$ is the length of the longest role path among $U_1, \ldots, U_n$, and
  8. $\mathsf{rd}(g\uparrow) = 0$;

- Let $\mathcal{C}$ be a finite set of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts. By $\mathsf{rd}(\mathcal{C})$, we denote the maximum role depth of concepts in $\mathcal{C}$ and 0 if $\mathcal{C}$ is the empty set;

We introduce two more notions concerning sets of concepts and define the level of nodes in a completion tree.

- Let $\mathcal{C}$ be a set of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts. We set

  $$\mathcal{C}|_{\exists cR} := \{C \in \mathcal{C} \mid \mathsf{sub}(C) \text{ contains a concept of the form } \exists(R_1 \sqcap \cdots \sqcap R_n).E \text{ with some } R_i \text{ being a complex role}\}$$

  and

  $$\mathcal{C}|_{\exists P} := \{C \in \mathcal{C} \mid \mathsf{sub}(C) \text{ contains a concept of the form } \exists U_1, \ldots, U_n.P\}.$$

- For nodes $a \in \mathsf{O_a} \cup \mathsf{O_c}$ in a completion tree $\mathbf{T}$, $\mathsf{lev}(a)$ denotes the *level* of a in $\mathbf{T}$, i.e., its distance to the root node.

Since the proof of termination involves establishing several lemmas, we start with giving an overview. Our main aim is to prove an upper bound for the size of completion trees constructed by the algorithm since, once this bound is established, the proof of termination is more or less straightforward. This bound is proved by establishing two other bounds: one for the depth of completion trees and one for the outdegree of completion trees. In order to establish the depth bound, we first prove that the level of abstract nodes having concrete successors is bounded. This is important since it implies that, if a node b is an S-relative of a node a with S complex role, then the depth of b is bounded. It is not hard to see that this latter bound is crucial for the boundedness of the depth of completion trees, since, if it would not exist, the $\mathsf{R}\forall$ rule could propagate concepts $\exists(R_1 \sqcap \cdots \sqcap R_n).C$ or $\exists U_1, \ldots, U_n.P$ to nodes on an arbitrary level. These concepts would then generate new successors on an even deeper level and this process could repeat indefinitely. To show the bound on the level of nodes with concrete successors, in turn, we prove that, if $\mathcal{L}(a)$ contains a concept of the form $\exists U_1, \ldots, U_n.P$ or $\exists(R_1 \sqcap \cdots \sqcap R_n).C$ with some R_i a complex role, then the level of the node a is bounded. We start with establishing these latter two bounds (one for each concept type).

Lemma 5.42. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derivable from an initial completion system S_D. For all $a \in \mathsf{O_a}$ in $\mathbf{T}$, we have $\mathsf{rd}(\mathcal{L}(a)|_{\exists cR}) \leq \mathsf{rd}(D) - \mathsf{lev}(a)$.*

Proof. The proof is by induction on the number of rule applications. The lemma is obviously true for the initial completion system S_D. For the induction step, we make a case distinction according to the rule applied. R$\sqcap$ and R$\sqcup$ are straightforward since they only add concepts C to labels $\mathcal{L}(a)$ with $\mathsf{rd}(C) \leq \mathsf{rd}(\mathcal{L}(a))$. R$\exists$c, R$\forall$c, Rpr, and Rch are trivial since they do not change existing node labels at all and introduce new nodes only with empty labels. The remaining cases are:

- Assume R$\exists$ is applied to a concept $C = \exists(R_1 \sqcap \cdots \sqcap R_n).C' \in \mathcal{L}(a)$ where $\mathsf{sub}(C')$ contains a concept of the form $\exists(R'_1 \sqcap \cdots \sqcap R'_m).E$ with R'_i complex role for some i. According to the definition of the "+" operation, the rule application either (i) generates a new $(R_1 \sqcap \cdots \sqcap R_n)$-successor b of a and sets $\mathcal{L}(b) = \{C'\}$ or (ii) chooses a neighbor b of a, appropriately augments the label of the edge between a and b (or vice versa) and sets $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C'\}$. In both cases, we obviously have $\mathsf{lev}(b) \leq \mathsf{lev}(a) + 1$. By induction hypothesis, we have $\mathsf{rd}(C) \leq \mathsf{rd}(D) - \mathsf{lev}(a)$. These two facts imply $\mathsf{rd}(C') \leq \mathsf{rd}(D) - \mathsf{lev}(b)$ since, clearly, $\mathsf{rd}(C) \geq \mathsf{rd}(C') + 1$ ("$\geq$" since one of the R_i may be a complex role).
- Assume R$\forall$ is applied to a concept $C = \forall(R_1 \sqcap \cdots \sqcap R_n).C' \in \mathcal{L}(a)$ adding C' to $\mathcal{L}(b)$ where $\mathsf{sub}(C')$ contains a concept of the form $\exists(R'_1 \sqcap \cdots \sqcap R'_m).E$ with R'_i complex role for some i. Since D is in restricted form, C is also in restricted form, and, hence, by Property 1a from Definition 5.37, one of the R_i is not a complex role (see Definition 5.37). This, together with the fact that b is a $(R_1 \sqcap \cdots \sqcap R_n)$-relative of a, implies $\mathsf{lev}(b) \in \{\mathsf{lev}(a) - 1, \mathsf{lev}(a) + 1\}$, i.e., $\mathsf{lev}(b) \leq \mathsf{lev}(a) + 1$. By induction hypothesis, $\mathsf{rd}(C) \leq \mathsf{rd}(D) - \mathsf{lev}(a)$. As in the R$\exists$ case, it follows that $\mathsf{rd}(C') \leq \mathsf{rd}(D) - \mathsf{lev}(b)$. ❑

Lemma 5.43. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derivable from an initial completion system S_D. For all $a \in \mathsf{O_a}$ in $\mathbf{T}$, we have $\mathsf{rd}(\mathcal{L}(a)|_{\exists P}) \leq \mathsf{rd}(D) - \mathsf{lev}(a)$.*

Proof. The proof is analogous to the one of Lemma 5.42, i.e., by induction on the number of rule applications. In the R$\forall$ case, we need to employ Property 1b from Definition 5.37 instead of Property 1a. ❑

Now for the bound on the level of nodes having concrete successors.

Lemma 5.44. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derivable from an initial completion system S_D. Then, for all $a \in \mathsf{O_a}$ in $\mathbf{T}$, $\mathsf{lev}(a) > \mathsf{rd}(D)$ implies that a has no concrete successors.*

Proof. Only the $\mathsf{R}\exists\mathsf{c}$ and Rpr rules may introduce successors for concrete features. We first treat the $\mathsf{R}\exists\mathsf{c}$ rule. Assume that the rule was applied to a concept $\exists U_1, \ldots, U_n.P \in \mathcal{L}(a)$ and generates a g-successor x for an abstract node b. By Lemma 5.43, we have $\mathsf{lev}(a) \leq \mathsf{rd}(D) - \mathsf{rd}(\exists U_1, \ldots, U_n.P)$. Furthermore, by definition of the $\mathsf{R}\exists\mathsf{c}$ rule and the "+" operation, we have $\mathsf{lev}(b) < \mathsf{lev}(a) + \mathsf{rd}(\exists U_1, \ldots, U_n.P)$, and, hence, $\mathsf{lev}(b) < \mathsf{rd}(D)$.

Now assume that the Rpr rule was applied to a node a and its R-neighbor b with R a predicate role. Observe that predicate roles are added to edge labels only by the $\mathsf{R}\exists$ rule. By definition of this rule and the "+" operation, b being an R-neighbor of a implies that one of the following cases holds:

1. b is an R-successor of a and $\mathcal{L}(a)$ contains a concept $\exists(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ with $R_i = R$ for some i;

2. b is an R-successor of a and $\mathcal{L}(b)$ contains a concept $\exists(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ with $R_i = R^-$ for some i;

3. a is an R^--successor of b and $\mathcal{L}(b)$ contains a concept $\exists(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ with $R_i = R^-$ for some i;

4. a is an R^--successor of b and $\mathcal{L}(a)$ contains a concept $\exists(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ with $R_i = R$ for some i.

In all four cases, it is not hard to prove that if the Rpr application generates a g-successor for an abstract node c, then $\mathsf{lev}(c) \leq \mathsf{rd}(D)$. We only deal with Case 1 exemplarily. In this case, it follows from Lemma 5.42 that

$$\mathsf{lev}(a) \leq \mathsf{rd}(D) - \mathsf{rd}(\exists(R_1 \sqcap \cdots \sqcap R_n).C). \tag{$*$}$$

Furthermore, we have $\mathsf{lev}(b) = \mathsf{lev}(a) + 1$. Suppose that the Rpr application generates a g-successor x for an abstract node c. By definition of the Rpr rule, it is easy to see that we have $\mathsf{lev}(c) < \mathsf{lev}(b) + \mathsf{rd}(R)$. Since $\mathsf{lev}(b) = \mathsf{lev}(a) + 1$, this yields $\mathsf{lev}(c) < \mathsf{lev}(a) + 1 + \mathsf{rd}(R)$. Together with $(*)$, we obtain

$$\mathsf{lev}(c) < \mathsf{rd}(D) - \mathsf{rd}(\exists(R_1 \sqcap \cdots \sqcap R_n).C) + 1 + \mathsf{rd}(R)$$

which clearly implies $\mathsf{lev}(c) \leq \mathsf{rd}(D)$ since $R \in \{R_1, \ldots, R_n\}$. ❏

We can now prove the bounds on the size of completion trees. We start with the bound on the depth.

Lemma 5.45. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derivable from an initial completion system S_D. Then the depth of $\mathbf{T}$ is bounded by $3 \cdot \mathsf{rd}(D)$.*

Proof. We prove the following claim which obviously implies that the depth of $\mathbf{T}$ is bounded by $3 \cdot \mathsf{rd}(D)$.

Claim: for all abstract nodes a in $\mathbf{T}$, $\mathsf{rd}(\mathcal{L}(a)) < 3 \cdot \mathsf{rd}(D) - \mathsf{lev}(a)$.

The proof is by induction on the number of rule applications. The claim is clearly true for the initial completion system S_D. Now for the induction step. We obviously

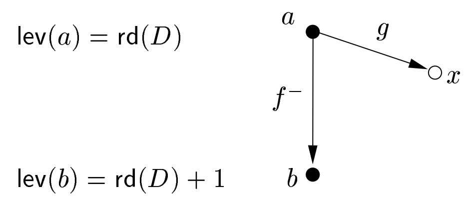

Figure 5.17: Node b is on level $\mathsf{rd}(D) + 1$ but nevertheless has an fg-successor.

have $\mathsf{rd}(\mathcal{L}(a)) \leq \mathsf{rd}(D)$ for all abstract nodes a in $\mathbf{T}$. This implies that the claim holds true for all nodes a with $\mathsf{lev}(a) < 2 \cdot \mathsf{rd}(D)$. Hence, we will in the following consider only nodes a with $\mathsf{lev}(a) \geq 2 \cdot \mathsf{rd}(D)$. We make a case distinction according to the rule applied. R$\sqcap$ and R$\sqcup$ are straightforward since they only add concepts C to labels $\mathcal{L}(a)$ with $\mathsf{rd}(C) \leq \mathsf{rd}(\mathcal{L}(a))$. R$\forall$c and Rch are trivial since they neither add new nodes nor do they change node labels.

- Assume R$\exists$ is applied to a concept $C = \exists(R_1 \sqcap \cdots \sqcap R_n).C' \in \mathcal{L}(a)$. According to the definition of the "+" operation, the rule application either (i) generates a new $(R_1 \sqcap \cdots \sqcap R_n)$-successor b of a and sets $\mathcal{L}(b) = \{C'\}$ or (ii) chooses a neighbor b of a, appropriately augments the label of the edge between a and b (or vice versa) and sets $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C'\}$. In both cases, we obviously have $\mathsf{lev}(b) \leq \mathsf{lev}(a) + 1$. By induction hypothesis, we have $\mathsf{rd}(C) < 3 \cdot \mathsf{rd}(D) - \mathsf{lev}(a)$. These two facts imply $\mathsf{rd}(C') < 3 \cdot \mathsf{rd}(D) - \mathsf{lev}(b)$ since, clearly, $\mathsf{rd}(C) \geq \mathsf{rd}(C') + 1$.

- Assume R$\forall$ is applied to a concept $\forall(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ adding C to $\mathcal{L}(b)$. We first show $\mathsf{lev}(b) \leq \mathsf{lev}(a) + 1$. For assume that the contrary holds. By definition of "relative" and since b is a $(R_1 \sqcap \cdots \sqcap R_n)$-relative of a, this can only be the case if all the roles $R_1, \ldots, R_n$ are complex. Since the maximum length of concrete paths in D is bounded by $\mathsf{rd}(D)$, Lemma 5.44 implies that $u^{\mathcal{I}}(c)$ is undefined for each concrete path u in D and each c with $\mathsf{lev}(c) \geq 2 \cdot \mathsf{rd}(D)$ (as Figure 5.17 shows, due to the presence of the inverse role constructor this does not necessarily hold for nodes c with $\mathsf{rd}(D) < \mathsf{lev}(c) < 2 \cdot \mathsf{rd}(D)$). Thus, since we assume $\mathsf{lev}(b) \geq 2 \cdot \mathsf{rd}(D)$, $u^{\mathcal{I}}(b)$ is undefined for each concrete path u in D. It follows that b is no R-relative of a for any complex role R: a contradiction. Hence we have shown that $\mathsf{lev}(b) \leq \mathsf{lev}(a) + 1$. We can now argue as in the R$\exists$ case.

- The R$\exists$c rule does not change concept labels but adds new abstract and concrete nodes to the completion tree. We must show that the level of all new nodes is at most $3 \cdot \mathsf{rd}(D) - 1$. First assume to the contrary that an abstract node a with $\mathsf{lev}(a) \geq 3 \cdot \mathsf{rd}(D)$ is generated. By definition of the R$\exists$c rule and of the "+" operation and since the inverse constructor is not admitted inside role paths, this implies the existence of an abstract node b with $\mathsf{lev}(b) \geq \mathsf{lev}(a)$ such that b has a concrete successor. This is a contradiction to Lemma 5.44. Now assume that a concrete node x with $\mathsf{lev}(x) \geq 3 \cdot \mathsf{rd}(D)$ is generated. Then there clearly

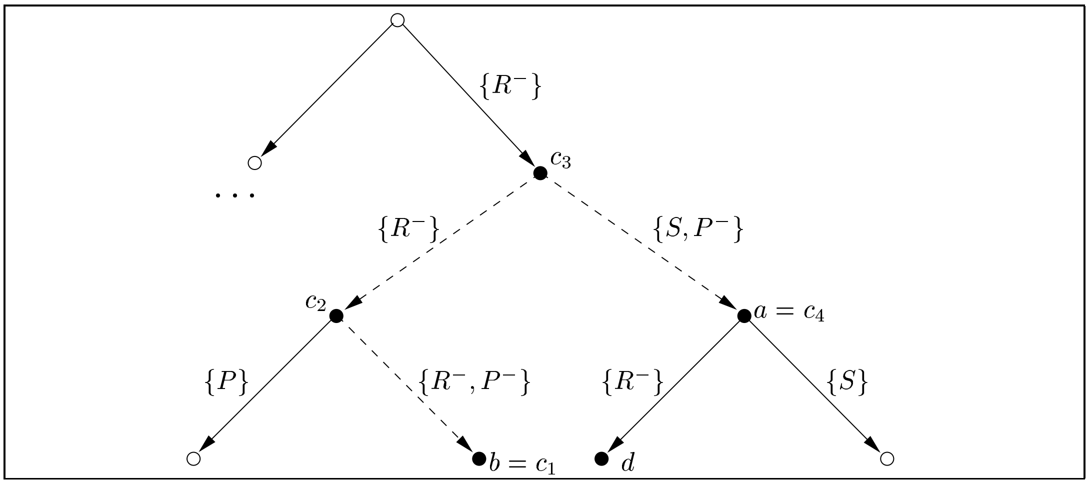

Figure 5.18: The neighborhood of node a.

exists an abstract node a with $\mathsf{lev}(a) = \mathsf{lev}(x) - 1$ such that x is successor of a which is again a contradiction to Lemma 5.44.

- The Rpr rule can be treated identical to the previous case.

❏

Before we can prove the upper bound on the outdegree, we introduce one more notion and establish a technical lemma. Due to the reuse of nodes by the "+"-operation and the presence of role and feature *chains* in the concrete domain concept and role constructors, the outdegree of an abstract node a in a completion tree $\mathbf{T}$ does not only depend on the concept label of a (and the labels of this node's incoming and outgoing edges) but also on the concept labels of other nodes in the "vicinity". The following notion will help to make this more precise:

Definition 5.46 (Neighborhood). Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derivable from an initial completion system S_D and let a be an abstract node in $\mathbf{T}$. The *neighborhood* of a in $\mathbf{T}$ is denoted by $\mathsf{nhood}(a, \mathbf{T})$ and defined as the set of abstract nodes b in $\mathbf{T}$ that satisfy the following condition: there exists a sequence of abstract nodes $c_1, \ldots, c_n$ and role names $R_1, \ldots, R_{n-1}$ with $n \leq \mathsf{rd}(D)$ such that

1. $c_1 = b$, $c_n = a$, and
2. for $1 \leq i < n$, either c_{i+1} is an R_i-successor of c_i and or c_i is an R_i^--successor of c_{i+1}.

◇

Note that $a \in \mathsf{nhood}(a, \mathbf{T})$ since we may have $n = 1$. Figure 5.18 displays an example completion tree $\mathbf{T}$. All nodes in the neighborhood of the node a are displayed as filled circles while nodes not in this neighborhood are displayed as unfilled circles. The sequence of nodes $c_1, \ldots, c_4$, whose existence implies that b is in the neighborhood of a, is highlighted by dashed lines.

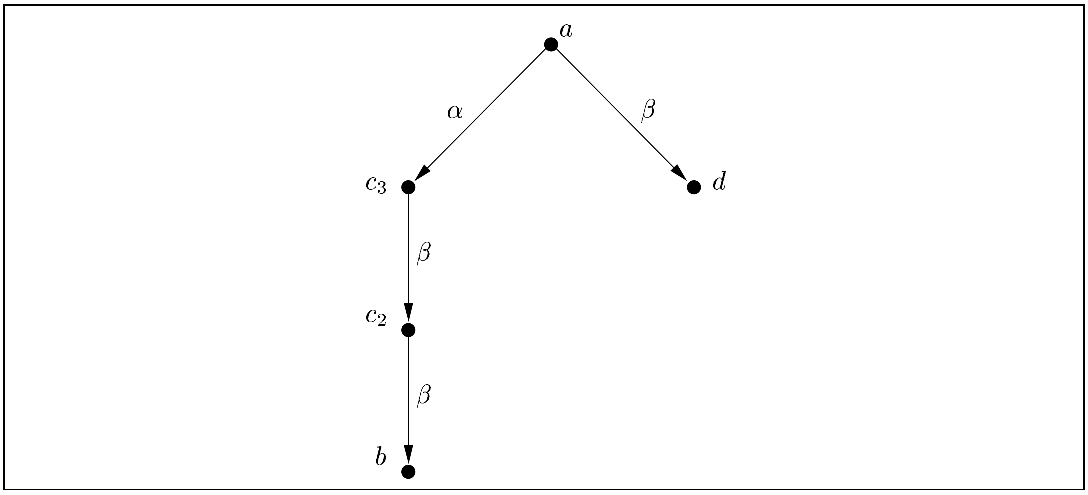

Figure 5.19: The tree Υ for the completion tree $\mathbf{T}$ and node a from Figure 5.18.

Intuitively, the neighborhood of a node a contains all nodes whose node labels and connecting edge labels have an impact on a's outdegree. For the node labels, the impact arises from the presence of concepts $\exists R.C$ and $\exists U_1, \ldots, U_n.P$. For edge labels, the impact arises from the presence of predicate roles. Note that the inverse constructor is not allowed inside paths, and the lengths of the "abstract part" of paths occurring in D is bounded by $\mathsf{rd}(D) - 1$, which is both reflected by the definition of "neighborhood".

Lemma 5.47. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derivable from an initial completion system S_D and a be an abstract node in $\mathbf{T}$. Then we have $|\mathsf{nhood}(a, \mathbf{T})| \leq (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$.*

Proof. Let S and a be as in the lemma. We define a tree Υ whose nodes are from the set $\{b \in \mathsf{O_a} \mid b \text{ used in } \mathbf{T}\} \times \mathbb{N}$ and which has two types of edges called α-edges and β-edges. For the construction, we will assume that nodes in Υ are either marked or unmarked. More precisely, Υ is constructed by starting with the tree consisting only of the single, unmarked node $(a, 0)$, initializing a counter variable m with 0, and then exhaustively performing the following induction step: choose an unmarked node (b, n) such that the depth of (b, n) in Υ is at most $\mathsf{rd}(D) - 2$. Then add the following successors of (b, n) to Υ:

1. if b is R-successor of c in $\mathbf{T}$ for some $R \in \mathsf{N_R}$, then add an α-successor (c, m);
2. if c is R^--successor of b in $\mathbf{T}$ for some $R \in \mathsf{N_R}$, then add a β-successor (c, m).

To complete the induction step, increment m by one and mark the node (b, n). Figure 5.19 shows the tree Υ for the completion tree $\mathbf{T}$ and node a from Figure 5.18.

It is not hard to verify that the first components of nodes in Υ are precisely the elements of $\mathsf{nhood}(a, \mathbf{T})$. Hence, it suffices to show that the number of nodes in Υ is bounded by $|\mathsf{sub}(D)|^{\mathsf{rd}(D)}$. Clearly, the depth of Υ is bounded by $\mathsf{rd}(D) - 1$. Let us investigate the outdegree:

1. Since each node in $\mathbf{T}$ has at most a single predecessor, each node in Υ has at most a single α-successor;

2. Since the inverse constructor is not allowed inside paths, $\mathsf{R}\exists$ is the only rule that may create successors in $\mathbf{T}$ whose connecting edge contains an inverse role. Since this rule may clearly generate at most a single successor per node and concept in $\mathsf{sub}(D)$, each node in $\mathbf{T}$ has at most $|\mathsf{sub}(D)|$ successors whose connecting edge contains an inverse role. It follows that each node in Υ has at most $|\mathsf{sub}(D)|$ β-successors.

Summing up, the maximum outdegree of nodes in Υ is $|\mathsf{sub}(D)| + 1$, which yields an upper bound of $(|\mathsf{sub}(D)| + 1)^{\mathsf{rd}(D)-1}$ for the number of nodes in Υ, which clearly implies the bound given in the lemma. ❏

Let C be a concept. In what follows, we use

- $\mathsf{maxar}(C)$ to denote the maximum arity of concrete domain predicates occurring in C and 1 if no such predicate occurs in C and

- $\mathsf{ncr}(C)$ to denote the number of distinct complex roles in C.

We may now establish the upper bound on the outdegree.

Lemma 5.48. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derivable from an initial completion system S_D. Then the outdegree of $\mathbf{T}$ is bounded by*

$$|\mathsf{sub}(D)| + \big(|\mathsf{sub}(D)| + \mathsf{ncr}(D) \cdot (|\mathsf{sub}(D)| + 2)\big) \cdot \mathsf{maxar}(D) \cdot (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$$

Proof. Let a be a node in $\mathbf{T}$. We analyze the maximum number of successors of a (not distinguishing between successors from $\mathsf{O_a}$ and successors from $\mathsf{O_c}$). The following rules may generate successors:

$\mathsf{R}\exists$ Only rule applications to concepts $\exists(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ can add successors to a. Since each rule application generates at most a single successor, the number of successors of a generated by $\mathsf{R}\exists$ is bounded by $|\mathsf{sub}(D)|$.

$\mathsf{R}\exists\mathsf{c}$ Since the length of role paths is bounded by $\mathsf{rd}(D)$ and the inverse constructor may not occur inside role paths, it is easily seen that only applications to concepts $\exists U_1, \ldots, U_n.P \in \mathcal{L}(b)$ with $b \in \mathsf{nhood}(a, \mathbf{T})$ may lead to the generation of successors for a (if $b \neq a$, this can only happen if the "+" operation reuses neighbors). Again since the inverse constructor may not occur inside role paths, each such application generates at most $\mathsf{maxar}(D)$ successors for a. Thus, by Lemma 5.47 the number of successors generated by this rule is bounded by $|\mathsf{sub}(D)| \cdot \mathsf{maxar}(D) \cdot (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$.

Rpr Only applications to complex roles $R \in \mathcal{L}(b, c)$ with $\{b, c\} \cap \mathsf{nhood}(a, \mathbf{T}) \neq \emptyset$ may generate new successors for a. Let us determine the number of edges (b, c) in $\mathbf{T}$ such that $\{b, c\} \cap \mathsf{nhood}(a, \mathbf{T}) \neq \emptyset$. We make a case distinction as follows:

1. $b \notin \mathsf{nhood}(a, \mathbf{T})$ and $c \in \mathsf{nhood}(a, \mathbf{T})$. Since $\mathbf{T}$ is a tree and thus every node has at most a single incoming edge, the number of such edges is bounded by $|\mathsf{nhood}(a, \mathbf{T})|$, i.e., by $(2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$ due toLemma 5.47.
2. $\{b, c\} \subseteq \mathsf{nhood}(a, \mathbf{T})$. Since $\mathbf{T}$ is a tree and by Lemma 5.47, the number of such edges is bounded by $(2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$.
3. $b \in \mathsf{nhood}(a, \mathbf{T})$ and $c \notin \mathsf{nhood}(a, \mathbf{T})$. The number of such edges is bounded by the number of successors of nodes in $\mathsf{nhood}(a, \mathbf{T})$ such that the label of the connecting edge contains a complex role. Since complex roles are added to node labels only by applications of the R$\exists$ rule, and, as shown above, the number of R$\exists$ applications per node is bounded by $|\mathsf{sub}(D)|$, every node in $\mathbf{T}$ has at most $|\mathsf{sub}(D)|$ successors such that the connecting edge is labeled with a complex role. Hence, by Lemma 5.47 the number of edges (b, c) in $\mathbf{T}$ such that $b \in \mathsf{nhood}(a, \mathbf{T})$ and $c \notin \mathsf{nhood}(a, \mathbf{T})$ is bounded by $|\mathsf{sub}(D)| \cdot (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$.

Summing up, the number of edges (b, c) in $\mathbf{T}$ such that $\{b, c\} \cap \mathsf{nhood}(a, \mathbf{T}) \neq \emptyset$ is bounded by $(|\mathsf{sub}(D)| + 2) \cdot (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$. Since each such edge may contain at most $\mathsf{ncr}(D)$ complex roles and each application of Rpr generates at most $\mathsf{maxar}(D)$ successors, the number of successors generated by this rule is bounded by $\mathsf{ncr}(D) \cdot \mathsf{maxar}(D) \cdot (|\mathsf{sub}(D)| + 2) \cdot (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}$.

Summing up, we obtain the bound in the lemma. ❑

Using the lemmas just established, we can now prove termination.

Proposition 5.49 (Termination). *Let D be an input to the completion algorithm and let*

$$k = \Big[|\mathsf{sub}(D)| + \big(|\mathsf{sub}(D)| + \mathsf{ncr}(D) \cdot (|\mathsf{sub}(D)| + 2)\big) \cdot \mathsf{maxar}(D) \cdot (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}\Big]^{3 \cdot \mathsf{rd}(D)}.$$

The algorithm terminates after at most

$$(|\mathsf{sub}(D)| \cdot k^{\mathsf{maxar}(D)}) + k \cdot (2 \cdot |\mathsf{sub}(D)| + \mathsf{ncr}(D))$$

rule applications.

Proof. Lemmas 5.48 and 5.45 imply that the number of nodes generated by the completion algorithm is bounded by k. Using this observation, we first investigate the maximum number of applications of the R$\sqcap$, R$\sqcup$, R$\exists$, and R$\forall$ rules. Each such application adds a new concept to a node label. Since the size of each node label is obviously bounded by $|\mathsf{sub}(D)|$, nodes are never removed from the tree, and concepts are never removed from node labels, there may be at most $|\mathsf{sub}(D)| \cdot k$ applications of the mentioned rules. It remains to treat applications of the remaining rules:

R$\exists$c This rule may be applied at most once per concept $\exists U_1, \ldots, U_n.P$ appearing in a node label. Hence, the R$\exists$c rule may be applied at most $|\mathsf{sub}(D)| \cdot k$ times.

R∀c/Rch These rules add new tuples $(x_1, \ldots, x_n)$ to $\mathcal{P}(P)$ for some predicate P appearing in D. Since the number of concrete nodes is bounded by k and the number of distinct predicates in D is bounded by $|\mathsf{sub}(D)|$, these rule may be applied at most $|\mathsf{sub}(D)| \cdot (k^{\mathsf{maxar(D)}})$ times.

Rpr This rule may be applied at most once per complex role appearing in an edge label. Since each node has at most one incoming edge, the number of Rpr applications is bounded by $k \cdot \mathsf{ncr}(D)$.

Taking the above observations together, we obtain the bound given in the lemma.

❑

We now prove soundness and completeness of the completion algorithm.

Proposition 5.50 (Soundness). *If there exists a complete and clash-free completion system $S = (\mathbf{T}, \mathcal{P})$ derivable from the initial completion system S_D, then D is satisfiable.*

Proof. Let $S = (\mathbf{T}, \mathcal{P})$ be as in Proposition 5.50. Since S is clash-free, there exists a solution δ for $\zeta_{\mathcal{P}}$, i.e., a mapping from the set of concrete nodes used in $\mathbf{T}$ to $\Delta_{\mathcal{D}}$ such that $\mathcal{P}$ is satisfied. Define the interpretation $\mathcal{I}$ by setting $\Delta_{\mathcal{I}}$ to the set of abstract nodes in $\mathbf{T}$,

$$\begin{aligned}
&A^{\mathcal{I}} \text{ to } \{a \mid A \in \mathcal{L}(a)\} \text{ for all } A \in \mathsf{N_C},\\
&R^{\mathcal{I}} \text{ to } \{(a, b) \mid b \text{ is } R\text{-neighbor of } a\} \text{ for all } R \in \mathsf{N_R}, \text{ and}\\
&g^{\mathcal{I}} \text{ to } \{(a, \delta(x)) \mid \mathcal{L}(a, x) = g\} \text{ for all } g \in \mathsf{N_{cF}}.
\end{aligned}$$

The relations $f^{\mathcal{I}}$ for $f \in \mathsf{N_{aF}}$ are functional since S is clash-free. The relations $g^{\mathcal{I}}$ for $g \in \mathsf{N_{cF}}$ are functional by definition of the "+" operation. The following claim shows that roles are "properly" interpreted:

Claim: For all $a, b \in \Delta_{\mathcal{I}}$ and roles $R_1 \sqcap \cdots \sqcap R_n \in \mathcal{R}$, we have $(a, b) \in (R_1 \sqcap \cdots \sqcap R_n)^{\mathcal{I}}$ iff b is an $R_1 \sqcap \cdots \sqcap R_n$-relative of a.

Due to the definition of $R_1 \sqcap \cdots \sqcap R_n$-relatives, it suffices to show that $(a, b) \in R_i^{\mathcal{I}}$ iff b is an R_i-relative of a for $1 \leq i \leq n$. We make a case distinction according to the type of R_i:

1. $R_i \in \mathsf{N_R}$. By definition of $\mathcal{I}$.

2. $R_i = R^-$ with $R \in \mathsf{N_R}$. Clearly, b is an R_i-relative of a iff a is an R-neighbor of b. By definition of $\mathcal{I}$, a is an R-neighbor of b iff $(b, a) \in R^{\mathcal{I}}$. By the semantics, $(b, a) \in R^{\mathcal{I}}$ iff $(a, b) \in R_i^{\mathcal{I}}$.

3. $R_i = \exists (u_1, \ldots, u_n), (v_1, \ldots, v_m).P$ is a predicate role. By definition, b is an R_i-relative of a iff either (i) b is an R_i-neighbor of a or (ii) there exist concrete domain elements $x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O_c}$ such that

   - $x_i \in \mathsf{neighb}(a, u_i)$ for $1 \leq i \leq n$,

- $y_i \in \mathsf{neighb}(b, v_i)$ for $1 \leq i \leq m$, and
- $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$.

Since the Rpr rule is not applicable, (i) implies (ii). Hence, we need to show that (ii) holds iff $(*)$ there exist $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in \Delta_\mathcal{D}$ such that

- $u_i^\mathcal{I}(a) = \alpha_i$ for $1 \leq i \leq n$,
- $v_i^\mathcal{I}(b) = \beta_i$ for $1 \leq i \leq m$, and
- $(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in P^\mathcal{D}$.

This is sufficient since, by the semantics, $(*)$ holds iff $(a, b) \in R_i^\mathcal{I}$. The direction from (ii) to $(*)$ is straightforward by definition of $\mathcal{I}$. Now for the direction from $(*)$ to (ii). Assume that $(*)$ holds. By definition of $\mathcal{I}$, this implies the existence of $x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O_c}$ such that

- $x_i \in \mathsf{neighb}(a, u_i)$ for $1 \leq i \leq n$,
- $y_i \in \mathsf{neighb}(b, v_i)$ for $1 \leq i \leq m$,
- $\delta(x_i) = \alpha_i$ for $1 \leq i \leq n$, and
- $\delta(y_i) = \beta_i$ for $1 \leq i \leq m$.

Since the Rch rule is not applicable to S and R_i occurs in D, we have either $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$ or $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(\overline{P})$. The latter implies $(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in \overline{P}^\mathcal{D}$ which is a contradiction. Hence, we conclude $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$.

4. $R_i = R^-$ with R predicate role. By definition, b is an R_i-relative of a iff a is an R-relative of b. As in the previous case, we conclude that this is the case iff $(b, a) \in R^\mathcal{I}$. By the semantics, $(b, a) \in R^\mathcal{I}$ iff $(a, b) \in R_i^\mathcal{I}$.

This finishes the proof of the claim. By induction on the concept structure, we now show that $C \in \mathcal{L}(a)$ implies $a \in C^\mathcal{I}$ for all $a \in \Delta_\mathcal{I}$ and $C \in \mathsf{sub}(D)$. The induction start consists of several cases:

- C is a concept name. Immediate consequence of the definition of $\mathcal{I}$.

- $C = \neg E$. Since D is in negation normal form, E is a concept name. Since S is clash-free, $E \notin \mathcal{L}(a)$ and, by definition of $\mathcal{I}$, $a \notin E^\mathcal{I}$. Hence, $a \in (\neg E)^\mathcal{I}$.

- $C = \exists U_1, \ldots, U_n.P$. Since the $\mathsf{R\exists c}$ rule is not applicable, by definition of $\mathcal{I}$ there exist nodes $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is U_i-neighbor of a for $1 \leq i \leq n$ and $(x_1, \ldots, x_n) \in \mathcal{P}(P)$. By multiple applications of the claim and definition of $\mathcal{I}$, we obtain $(a, \delta(x_i)) \in U_i^\mathcal{I}$ for $1 \leq i \leq n$. Moreover, since δ is a solution for $\zeta_\mathcal{P}$, we also have $(\delta(x_1), \ldots, \delta(x_n)) \in P^\mathcal{D}$. Hence, $a \in (\exists U_1, \ldots, U_n.P)^\mathcal{I}$.

- $C = \forall U_1, \ldots, U_n.P$. Let $\alpha_1, \ldots, \alpha_n \in \Delta_\mathcal{D}$ be such that $(a, \alpha_i) \in U_i^\mathcal{I}$ for $1 \leq i \leq n$. By the claim and definition of $\mathcal{I}$, this implies that there exist $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is a U_i-neighbor of a and $\delta(x_i) = \alpha_i$ for $1 \leq i \leq n$. Since the $\mathsf{R\forall c}$

rule is not applicable, we have $(x_1, \ldots, x_n) \in \mathcal{P}(P)$. The fact that δ is a solution for $\zeta_\mathcal{P}$ yields $(\alpha_1, \ldots, \alpha_n) \in P^\mathcal{D}$. Since this holds for all $\alpha_1, \ldots, \alpha_n \in \Delta_\mathcal{D}$ as above, we conclude $a \in (\forall U_1, \ldots, U_n.P)^\mathcal{I}$.

- $C = g\uparrow$. Since S is clash-free, a has no g-successor x in $\mathbf{T}$. By definition of $\mathcal{I}$, $g^\mathcal{I}(a)$ is undefined and hence $a \in (g\uparrow)^\mathcal{I}$.

For the induction step, we make a case distinction according to the topmost constructor in C.

- $C = C_1 \sqcap C_2$. Since the R$\sqcap$ rule is not applicable to S, we have $\{C_1, C_2\} \subseteq \mathcal{L}(a)$. By induction, $a \in C_1^\mathcal{I}$ and $a \in C_2^\mathcal{I}$, which implies $a \in (C_1 \sqcap C_2)^\mathcal{I}$.
- $C = C_1 \sqcup C_2$. Similar to the previous case.
- $C = \exists(R_1 \sqcap \cdots \sqcap R_n).E$. Since the R$\exists$ rule is not applicable to S, there exists an abstract node b in $\mathbf{T}$ such that b is $R_1 \sqcap \cdots \sqcap R_n$-relative of b in $\mathcal{T}$ and $E \in \mathcal{L}(b)$. The claim yields $(a, b) \in (R_1 \sqcap \cdots \sqcap R_n)^\mathcal{I}$. By induction, we have $b \in E^\mathcal{I}$. Hence, we conclude $a \in (\exists(R_1 \sqcap \cdots \sqcap R_n).E)^\mathcal{I}$.
- $C = \forall(R_1 \sqcap \cdots \sqcap R_n).E$. Let $b \in \Delta_\mathcal{I}$ such that $(a, b) \in (R_1 \sqcap \cdots \sqcap R_n)^\mathcal{I}$. By the claim, b is an $(R_1 \sqcap \cdots \sqcap R_n)$-relative of a in $\mathbf{T}$. Since the R$\forall$ rule is not applicable to S, we have $E \in \mathcal{L}(b)$. By induction, it follows that $b \in E^\mathcal{I}$. Since this holds for all b, we can conclude $a \in (\forall(R_1 \sqcap \cdots \sqcap R_n).E)^\mathcal{I}$.

Since $D \in \mathcal{L}(a_0)$ for the root a_0 of $\mathbf{T}$, we have $D^\mathcal{I} \neq \emptyset$ and hence $\mathcal{I}$ is a model of D. ❑

It remains to prove completeness.

Proposition 5.51 (Completeness). *For any satisfiable $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept D, the completion rules can be applied to S_D such that a complete and clash-free completion system for D is obtained.*

Proof. Let $\mathcal{I}$ be a model of D. We use this model to "guide" the application of the non-deterministic completion rules R$\sqcup$ and Rch and the non-deterministic "+"-operation such that a complete and clash-free completion system for D is obtained. A completion system $S = (\mathbf{T}, \mathcal{P})$ is called *$\mathcal{I}$-compatible* iff there exists a function π mapping the abstract nodes in $\mathbf{T}$ to $\Delta_\mathcal{I}$ and the concrete nodes in $\mathbf{T}$ to $\Delta_\mathcal{D}$ such that

a) $C \in \mathcal{L}(a) \Rightarrow \pi(a) \in C^\mathcal{I}$

b) b is an $(R_1 \sqcap \cdots \sqcap R_n)$-relative of $a \Rightarrow (\pi(a), \pi(b)) \in (R_1 \sqcap \cdots \sqcap R_n)^\mathcal{I}$

c) x is a g-successor of $a \Rightarrow g^\mathcal{I}(\pi(a)) = \pi(x)$

d) $(x_1, \ldots, x_n) \in \mathcal{P}(P) \Rightarrow (\pi(x_1), \ldots, \pi(x_n)) \in P^\mathcal{D}$

e) there exists at most a single f-neighbor of a

for all abstract nodes a, b in $\mathbf{T}$, concepts $C \in \mathsf{sub}(D)$, roles $R_1 \sqcap \cdots \sqcap R_n \in \mathcal{R}$, concrete nodes $x, x_1, \ldots, x_n$ in $\mathbf{T}$, abstract features f, concrete features g, and predicates $P \in \Phi_\mathcal{D}$. Note that, due to the definition of the "+"-operation, nodes in completion trees derived by rule application may contain at most a single g-successor per node. Thus, we do not need an equivalent of Property e) for concrete features.

Claim: If a completion system S is $\mathcal{I}$-compatible and a rule R is applicable to S, then R can be applied such that an $\mathcal{I}$-compatible completion system S' is obtained.

Let S be an $\mathcal{I}$-compatible completion system, π be a function satisfying a) to e), and let R be a completion rule applicable to S. We make a case distinction according to the type of R.

R$\sqcap$ The rule is applied to a concept $C_1 \sqcap C_2 \in \mathcal{L}(a)$. By a), $C_1 \sqcap C_2 \in \mathcal{L}(a)$ implies $\pi(a) \in (C_1 \sqcap C_2)^\mathcal{I}$ and hence $\pi(a) \in C_1^\mathcal{I}$ and $\pi(a) \in C_2^\mathcal{I}$. Since the rule adds C_1 and C_2 to $\mathcal{L}(a)$, π clearly satisfies a) to e) w.r.t. the obtained completion system S'.

R$\sqcup$ The rule is applied to $C_1 \sqcup C_2 \in \mathcal{L}(a)$. $C_1 \sqcup C_2 \in \mathcal{L}(a)$ implies $\pi(a) \in C_1^\mathcal{I}$ or $\pi(a) \in C_2^\mathcal{I}$. Since the rule adds either C_1 or C_2 to $\mathcal{L}(a)$, it can be applied such that π satisfies a) to e) w.r.t. the obtained completion system S'.

R$\exists$ The rule is applied to a concept $\exists(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$. By a), this implies $\pi(a) \in (\exists(R_1 \sqcap \cdots \sqcap R_n).C)^\mathcal{I}$ and, hence, there exists a $d \in \Delta_\mathcal{I}$ such that $(\pi(a), d) \in (R_1 \sqcap \cdots \sqcap R_n)^\mathcal{I}$ and $d \in C^\mathcal{I}$. We distinguish three cases:

1. There already exists a successor b of a such that $\pi(b) = d$. In this case, apply the rule and the "+"-operation such that b is renamed to c, $\mathcal{L}(a, c)$ is set to $\mathcal{L}(a, c) \cup \{R_1, \ldots, R_n\}$ and $\mathcal{L}(c)$ to $\mathcal{L}(c) \cup \{C\}$. Define π' as $\pi \cup \{c \mapsto d\}$.
2. For the predecessor b of a, we have $\pi(b) = d$. In this case, apply the rule such that b is renamed to c, $\mathcal{L}(c, a)$ is set to $\mathcal{L}(c, a) \cup \{\mathsf{Inv}(R_1), \ldots, \mathsf{Inv}(R_n)\}$ and $\mathcal{L}(c)$ to $\mathcal{L}(c) \cup \{C\}$. Define π' as $\pi \cup \{c \mapsto d\}$.
3. Otherwise add a fresh $(R_1 \sqcap \cdots \sqcap R_n)$-successor b of a and set $\mathcal{L}(b) := \{C\}$. Define π' as $\pi \cup \{b \mapsto d\}$.

It all three cases, it is readily checked that π' satisfies a) to e) w.r.t. S'.

R$\forall$ The rule is applied to a concept $\forall(R_1 \sqcap \cdots \sqcap R_n).C \in \mathcal{L}(a)$ and an $(R_1 \sqcap \cdots \sqcap R_n)$-relative b of a. By a), b), and the semantics, this implies $\pi(a) \in (\forall(R_1 \sqcap \cdots \sqcap R_n).C)^\mathcal{I}$, $(\pi(a), \pi(b)) \in (R_1 \sqcap \cdots \sqcap R_n)^\mathcal{I}$, and $\pi(b) \in C^\mathcal{I}$. The rule application adds C to $\mathcal{L}(b)$. Obviously, π satisfies a) to e) w.r.t. the obtained completion system S'.

R$\exists$c The rule is applied to a concept $\exists U_1, \ldots, U_n.P \in \mathcal{L}(a)$ with $U_i = R_1^{(i)} \cdots R_{k_i}^{(i)} g_i$ for $1 \leq i \leq n$. By a), this implies $\pi(a) \in (\exists U_1, \ldots, U_n.P)^\mathcal{I}$. Hence, there exist $d_j^{(i)} \in \Delta_\mathcal{I}$ for $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and $\alpha_1, \ldots, \alpha_n \in \Delta_\mathcal{D}$ such that

- $(\pi(a), d_1^{(i)}) \in (R_1^{(i)})^\mathcal{I}$ for $1 \leq i \leq n$,

– $(d_{j-1}^{(i)}, d_j^{(i)}) \in (R_j^{(i)})^{\mathcal{I}}$ for $1 \leq i \leq n$ and $1 < j \leq k_i$,
– $g_i^{\mathcal{I}}(d_{k_i}^{(i)}) = \alpha_i$ for $1 \leq i \leq n$, and
– $(\alpha_1, \ldots, \alpha_n) \in P^{\mathcal{D}}$.

We use these domain elements to (i) perform a step-by-step guidance of the "+"-operation and (ii) extend the mapping π to a new mapping π' that satisfies a) to e). More precisely, we will guide the "+"-operation in the same way as for the R∃ rule. To construct π', we start with π and then modify this mapping everytime a node is renamed or a new node is introduced. The rule application performs n calls to the "+"-operation, each one of the form $S' + aU_ix$. Fix an i. By definition of "+", the call $S' + aU_ix$ is broken down into the following, simpler "+"-calls:

– A call $S' + aR_1b_1$. We can distinguish three subcases:
  1. There already exists a successor c of a such that $\pi(c) = d_1^{(i)}$. In this case, apply the "+"-operation such that c is renamed to b_1 and $\mathcal{L}(a, b_1)$ is set to $\mathcal{L}(a, b_1) \cup \{R_1\}$. Moreover, set $\pi'(b_1) := d_1^{(i)}$.
  2. For the predecessor c of a, we have $\pi(c) = d_1^{(i)}$. In this case, apply the "+"-operation such that c is renamed to b_1 and $\mathcal{L}(b_1, a)$ is set to $\mathcal{L}(b_1, a) \cup \{R_1^-\}$. Moreover, set $\pi'(b_1) := d_1^{(i)}$.
  3. Otherwise add a fresh R_1-successor b_1 of a. Set $\pi'(b_1) = d_1^{(i)}$.
– Calls $S' + b_jRb_{j+1}$ for $1 \leq j < k$. We can distinguish the same three subcases as above with b_j playing the role of a and b_{j+1} playing the role of b_1. In every case, we set $\pi'(b_{j+1}) := d_{j+1}^{(i)}$.
– A call to the "+"-operation of the form $S' + b_kgx$. If b_k already has a g-successor y, replace y with x in $\mathbf{T}$ and $\mathcal{P}$. Otherwise extend $\mathbf{T}$ with a new g-successor x of a. Set $\pi'(x) := \alpha_i$.

Assume that, for all i with $1 \leq i \leq n$, the "+"-operation has been called and the π' mapping has been modified as indicated. Call the completion system obtained by rule application S'. It is readily checked that π' satisfies a) to e) w.r.t. S' (note, however, that the rule application may generate new S-relative relationships for complex roles S).

R∀c The rule is applied to a concept $\forall U_1, \ldots, U_n.P \in \mathcal{L}(a)$ and nodes $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is a U_i-neighbor of a for $1 \leq i \leq n$. By a), $a \in (\forall U_1, \ldots, U_n.P)^{\mathcal{I}}$. By b) and c), $(\pi(a), \pi(x_i)) \in U_i^{\mathcal{I}}$ for $1 \leq i \leq n$. Hence, by the semantics, we have $(\pi(x_1), \ldots, \pi(x_n)) \in P^{\mathcal{D}}$. Since rule application adds $(x_1, \ldots, x_n)$ to $\mathcal{P}(P)$, it is easy to see that π satisfies a) to e) w.r.t. the resulting completion system S'.

Rpr The rule is applied to a node a and its R-neighbor b, where R is a predicate role $\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P$. By b), b being an R-neighbor of a implies $(\pi(a), \pi(b)) \in R^{\mathcal{I}}$. We may now proceed as in the case of the R∃c rule, i.e., deduce the existence of certain abstract and concrete nodes and then use these to guide the rule application.

Rch If the rule is applied to two nodes $a, b \in \mathsf{O_a}$ and their neighbors $x_1, \ldots, x_n \in \mathsf{O_c}$ and $y_1, \ldots, y_m \in \mathsf{O_c}$, then there exists a predicate role

$$\exists (u_1, \ldots, u_n), (v_1, \ldots, v_m).P \text{ occurring in } D,$$

and we have $x_i \in \mathsf{neighb}(a, u_i)$ for $1 \leq i \leq n$ and $y_i \in \mathsf{neighb}(b, v_i)$ for $1 \leq i \leq m$. By b) and c), this implies $(\pi(a), \pi(x_i)) \in u_i^{\mathcal{I}}$ for $1 \leq i \leq n$ and $(\pi(b), \pi(y_i)) \in u_i^{\mathcal{I}}$ for $1 \leq i \leq m$. The rule application adds $(x_1, \ldots, x_n, y_1, \ldots, y_n)$ either to $\mathcal{P}(P)$ or to $\mathcal{P}(\overline{P})$. By the semantics, we have either

$$(\pi(x_1), \ldots, \pi(x_n), \pi(y_1), \ldots, \pi(y_m)) \in P^{\mathcal{D}}$$

or

$$(\pi(x_1), \ldots, \pi(x_n), \pi(y_1), \ldots, \pi(y_m)) \in \overline{P}^{\mathcal{D}}.$$

Hence, the rule Rch can be applied such that π satisfies a) to e) w.r.t. the obtained completion system S'.

It remains to show that Proposition 5.51 is a consequence of the above claim. Let $S_D = (\mathbf{T}_D, \mathcal{P}_\emptyset)$ be the initial completion system for D and let a_0 be the node in $\mathbf{T}_D$. Set $\pi(a_0)$ to d for a $d \in D^{\mathcal{I}}$. Obviously, π satisfies a) to e) and hence S_D is $\mathcal{I}$-compatible. By the claim, the completion rules can be applied such that only $\mathcal{I}$-compatible completion systems are obtained. By Lemma 5.49, every sequence of rule applications terminates yielding a complete completion system. Hence, we can obtain a complete and $\mathcal{I}$-compatible completion system $S = (\mathbf{T}, \mathcal{P})$ by rule application. It remains to show that this implies clash-freeness of S. Let π be a mapping for S satisfying a) to e). We make a case distinction according to the various clash types (c.f. Definition 5.41).

1. S does not contain a clash of the form $\{A, \neg A\} \subseteq \mathcal{L}(a)$ since, together with a), this would imply $\pi(a) \in A^{\mathcal{I}} \cap (\neg A)^{\mathcal{I}}$, which is impossible.

2. By e), $\mathbf{T}$ contains at most a single f-neighbor for each node a in $\mathbf{T}$ and each $f \in \mathsf{N_{aF}}$.

3. Assume that $\mathbf{T}$ contains an abstract node a and a concrete node x such that $g\uparrow \in \mathcal{L}(a)$ and x is g-successor of a in $\mathbf{T}$ for some $g \in \mathsf{N_{cF}}$. By a), we have $\pi(a) \in (g\uparrow)^{\mathcal{I}}$. By c), we have $g^{\mathcal{I}}(\pi(a)) = \pi(x)$, which is a contradiction.

4. It remains to show that S is concrete domain satisfiable, i.e., that the predicate conjunction $\zeta_{\mathcal{P}}$ is satisfiable. However, using d), it is straightforward to show that the "concrete part" of π is a solution for $\zeta_{\mathcal{P}}$.

❑

As described in Section 2.2.1, concept satisfiability w.r.t. TBoxes can be reduced to concept satisfiability without reference to TBoxes by using *unfolding* [Nebel 1990]. Together with Lemmas 5.49, 5.50, and 5.51, this gives the following decidability result:

Theorem 5.52. *If $\mathcal{D}$ is admissible, then satisfiability of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concepts w.r.t. acyclic TBoxes is decidable.*

From the well-known reduction of subsumption to (un)satisfiability, it follows that $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept subsumption w.r.t. acyclic TBoxes is decidable as well. In the next section, we modify the presented completion algorithm to *directly* take into account acyclic TBoxes and then investigate its complexity.

5.4.3 Adding Acyclic TBoxes

We use the modification scheme from Section 4.1.1 to extend the presented completion algorithm to acyclic TBoxes. More precisely, the modified algorithm is capable of deciding satisfiability of concept names w.r.t. *simple* $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$*-TBoxes*, which are defined as in Definition 4.2 with the only difference that right-hand sides of concept equations may now also be of the form $\exists(R_1 \sqcap \cdots \sqcap R_n).C$, $\forall(R_1 \sqcap \cdots \sqcap R_n).C$, $\exists U_1, \ldots, U_n.P$, $\forall U_1, \ldots, U_n.P$, and $g\uparrow$. Since Lemma 4.3 clearly generalizes from $\mathcal{ALC}$ to $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$, it is easily seen that satisfiability of concepts w.r.t. acyclic TBoxes can be reduced to this task (c.f. Section 4.1.1). In analogy to the modified completion algorithm from Section 4.1.1, the modified $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-completion algorithm works on *simple* completion systems, i.e., completion systems in which node labels are sets of concept *names*.

Definition 5.53 (Modified Completion Algorithm).
The *modified* $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$*-completion algorithm* is obtained from the completion algorithm in Figure 5.16 by performing the following modifications:

1. To decide the satisfiability of a concept name A w.r.t. a simple TBox $\mathcal{T}$, the modified algorithm starts with the initial completion system $S_A := (\mathbf{T}_A, \mathcal{P}_\emptyset)$, where $\mathbf{T}_A$ is the tree consisting of a single node a with $\mathcal{L}(a) = \{A\}$ and $\mathcal{P}_\emptyset$ maps each $P \in \Phi_\mathcal{D}$ to $\emptyset$.

2. The completion rules are modified as follows: in the premise of each rule, substitute
$$\text{“}C \in \mathcal{L}(a)\text{”} \quad \text{with} \quad \text{“}B \in \mathcal{L}(a) \text{ and } B \doteq C \in \mathcal{T}\text{”}.$$
For example, in the conjunction rule, “$C_1 \sqcap C_2 \in \mathcal{L}(a)$” is replaced with “$B \in \mathcal{L}(a)$ and $(B \doteq C_1 \sqcap C_2) \in \mathcal{T}$”;

◇

In Section 4.1.1, we defined what it means for a simple ABox $\mathcal{A}$ to be a “variant” of an ABox $\mathcal{A}'$. Based on this notion, we then proved soundness, completeness, and termination of the modified $\mathcal{ALC}$-completion algorithm by establishing a correspondence between runs of the algorithm on input $A, \mathcal{T}$ and runs of the original algorithm on input C, where C is obtained by unfolding A w.r.t. $\mathcal{T}$. Since correctness and termination of the modified $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-completion algorithm is proved in the same way, we adapt the notion “variant” from ABoxes to completion systems.

Definition 5.54. A simple completion system $S = (\mathbf{T}, \mathcal{P})$ is a *variant* of a completion system $S' = (\mathbf{T}', \mathcal{P}')$ w.r.t. a TBox $\mathcal{T}$ iff the following conditions hold:

1. $A \in \mathcal{L}(a)$ and $\mathsf{unfold}(A, \mathcal{T}) = C$ implies $C \in \mathcal{L}'(a)$,
2. $C \in \mathcal{L}'(a)$ implies the existence of an $A \in \mathsf{N_C}$ such that $A \in \mathcal{L}(a)$ and $\mathsf{unfold}(A, \mathcal{T}) = C$,
3. b is an $(R_1 \sqcap \cdots \sqcap R_n)$-successor of a in $\mathbf{T}$ iff b is an $(R_1 \sqcap \cdots \sqcap R_n)$-successor of a in $\mathbf{T}'$,
4. x is a g-successor of a in $\mathbf{T}$ iff x is a g-successor of a in $\mathbf{T}'$, and
5. $\mathcal{P} = \mathcal{P}'$.

$\diamond$

We can now establish a lemma that describes the correspondence between runs of the original algorithm and runs of the modified algorithm as described above.

Lemma 5.55. *Let S_1 be a simple completion system that is a variant of a completion system S'_1 w.r.t. a simple TBox $\mathcal{T}$.*

- *If the modified completion algorithm can apply a completion rule* R *to S_1 yielding a completion system S_2, then the original completion algorithm can apply* R *to S'_1 yielding a completion system S'_2 such that S_2 is a variant of S'_2 w.r.t. $\mathcal{T}$.*
- *Conversely, if the original completion algorithm can apply a completion rule* R *to S'_1 yielding a completion system S'_2, then the modified completion algorithm can apply* R *to S_1 yielding a variant S_2 of S'_2 w.r.t. $\mathcal{T}$.*

Proof. The proof is by a straightforward case analysis (c.f. the proof of Lemma 4.6). ❑

In order to analyze the space requirements of the modified completion algorithm, we need to investigate the connection between unfolding and the various measures used in the formulation of Proposition 5.49.

Lemma 5.56. *Let A be a concept name and $\mathcal{T}$ a simple TBox such that A occurs in $\mathcal{T}$. If C is the result of unfolding A w.r.t. $\mathcal{T}$, then*

1. $\mathsf{maxar}(C) \leq |\mathcal{T}|$,
2. $\mathsf{ncr}(C) \leq |\mathcal{T}|$,
3. $\mathsf{rd}(C) \leq |\mathcal{T}|$, *and*
4. $|\mathsf{sub}(C)| \leq |\mathcal{T}|$.[8]

[8] I.e, $\mathsf{maxar}()$, $\mathsf{ncr}()$, $\mathsf{rd}()$, and $|\mathsf{sub}()|$ are polynomial under unfolding of simple TBoxes.

Proof. Properties 1 and 2 are trivial and Property 3 can be proved as in the case of $\mathcal{ALC}$ (c.f. the proof of Proposition 4.7). Hence, we concentrate on the proof of Property 4. Let A be a concept name, $\mathcal{T}$ a simple TBox, and χ be the set of concept names used in $\mathcal{T}$ (thus, $A \in \chi$). We prove Property 4 by defining a total, injective function f from $\mathsf{sub}(C)$ to χ. The existence of such a function implies $|\mathsf{sub}(C)| \leq |\mathcal{T}|$ since, clearly, $|\chi| \leq |\mathcal{T}|$. For defining f, assume that the concept names in χ are linearly ordered and, for each set Ψ with $\emptyset \subset \Psi \subseteq \chi$, $\min(\Psi)$ denotes the concept in Ψ which is minimal w.r.t. this ordering. Define the function f from $\mathsf{sub}(C)$ to χ as follows:

$$f(E) := \min\{B \in \chi \mid \mathsf{unfold}(B, \mathcal{T}) = E\}.$$

We show that f is well-defined and injective.

- Let k be the number of iterations performed by the **while** loop in the unfolding algorithm (see Figure 2.5) if started on $(A, \mathcal{T})$ and let C_i $(0 \leq i \leq k)$ denote the concept C after the i-th loop, i.e., $C_0 = A$ and $C_k = C$. To prove well-definedness, we establish the following claim:

  Claim: For all $0 \leq i \leq k$, and for all $E \in \mathsf{sub}(C_i)$, there exists a $B \in \chi$ such that $\mathsf{unfold}(E, \mathcal{T}) = \mathsf{unfold}(B, \mathcal{T})$.

  The claim implies well-definedness since, for all concepts $E \in \mathsf{sub}(C_k) = \mathsf{sub}(C)$, we have $\mathsf{unfold}(E, \mathcal{T}) = E$. The proof of the claim is by induction on i. For $i = 0$, the claim trivially holds since $C_0 = A$ and $A \in \chi$. Now for the induction step. Assume that, in the the i-th step, every occurrence of a concept name A' has been replaced by a concept F. Let $E \in \mathsf{sub}(C_{i+1}) \setminus \mathsf{sub}(C_i)$. Then we have one of the following two cases:

  - $E \in \mathsf{sub}(F)$. Since $\mathcal{T}$ is simple, this implies that either $E = F$ or E is a concept name from χ. In the first case, E occurs on the right-hand side of a concept equation in $\mathcal{T}$ and hence there obviously exists a $B \in \chi$ as required. In the second case, we can use E itself as the required B.
  - $E \notin \mathsf{sub}(F)$. Then there exists an E' in $\mathsf{sub}(C_i)$ such that E can be obtained from E' by substituting every occurrence of A' in E' by F. By induction hypothesis, there exists a $B \in \chi$ such that $\mathsf{unfold}(E', \mathcal{T}) = \mathsf{unfold}(B, \mathcal{T})$. Since, obviously, $\mathsf{unfold}(E, \mathcal{T}) = \mathsf{unfold}(E', \mathcal{T})$, we have $\mathsf{unfold}(E, \mathcal{T}) = \mathsf{unfold}(B, \mathcal{T})$.

- Assume that f is not injective, i.e., there exist two concepts $E, E' \in \mathsf{sub}(\mathsf{C})$ with $E \neq E'$ such that $f(E) = f(E') = B$. By definition of f, this implies $\mathsf{unfold}(B, \mathcal{T}) = E$ and $\mathsf{unfold}(B, \mathcal{T}) = E'$. Since $E \neq E'$ and unfolding is deterministic, this is obviously impossible. ❑

We are now ready to prove correctness and analyze the time requirements of the modified completion algorithm.

Proposition 5.57. *The modified completion algorithm is sound and complete. Moreover, if it is started on input $(A, \mathcal{T})$, then it terminates after at most $2^{p(|\mathcal{T}|)}$ rule applications, for some polynomial $p(n)$.*

Proof. Soundness and completeness are an immediate consequence of Lemma 5.55 and the following facts:

1. the original algorithm is sound, complete, and terminating;
2. a concept name A and a simple TBox $\mathcal{T}$ have a model iff the concept $C = \mathsf{unfold}(A, \mathcal{T})$ has a model.

Now for termination. By Lemma 5.55, the maximum number of rule applications made by the modified algorithm if started on $A, \mathcal{T}$ is identical to the maximum number of rule applications made by the original algorithm if started on $D = \mathsf{unfold}(A, \mathcal{T})$. By Proposition 5.49, the latter number is bounded by

$$(|\mathsf{sub}(D)| \cdot k^{\mathsf{maxar}(D)}) + k \cdot (2 \cdot |\mathsf{sub}(D)| + \mathsf{ncr}(D))$$

where

$$k = \Big[|\mathsf{sub}(D)| + \big(|\mathsf{sub}(D)| + \mathsf{ncr}(D) \cdot (|\mathsf{sub}(D)| + 2)\big) \cdot \mathsf{maxar}(D) \cdot (2 \cdot |\mathsf{sub}(D)|)^{\mathsf{rd}(D)}\Big]^{3 \cdot \mathsf{rd}(D)}.$$

Using Lemma 5.56, we can infer the existence of a polynomial $p(n)$ as required. ❑

Finally, the upper bound for $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes can be given.

Theorem 5.58. *Let $\mathcal{D}$ be an admissible concrete domain.*

1. *If $\mathcal{D}$-satisfiability is in* NP*, then $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes is in* NExpTime*.*
2. *If $\mathcal{D}$-satisfiability is in* PSpace*, then $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes is in* ExpSpace*.*
3. *If $\mathcal{D}$-satisfiability is in* NExpTime *(*ExpSpace*), then $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes is in* 2-NExpTime *(*2-ExpSpace*).*

Proof. As above, we only deal with the satisfiability of concept names w.r.t. simple TBoxes since the more general problem referred to in the theorem can be polynomially reduced to this task.

First for Point 1. By Proposition 5.57, the completion algorithm terminates after at most $2^{p(|\mathcal{T}|)}$ rule applications if started on $A, \mathcal{T}$, where $p(n)$ is a polynomial. Since each rule application adds at most a single tuple to $\mathcal{P}$, the length of predicate conjunctions $\zeta_{\mathcal{P}}$ encountered while checking for clashes is also bounded by $2^{p(|\mathcal{T}|)}$. Hence, $\mathcal{D}$-satisfiability being in NP implies that clashes can be checked for in (non-deterministic) time exponential in $|\mathcal{T}|$. Since the number of clash checks is bounded by the number of rule applications, we conclude that the modified completion algorithm can be executed in non-deterministic exponential time.

Now for Point 2. Since the completion algorithm terminates after at most $2^{p(|\mathcal{T}|)}$ rule applications, it is obvious that the number of nodes in the constructed completion systems is bounded by $2^{q(|\mathcal{T}|)}$, where $q(n)$ is a polynomial. Hence, the space required to store completion systems is exponential in $|\mathcal{T}|$. Moreover, as argued above, the length of predicate conjunctions $\zeta_{\mathcal{P}}$ is exponential in $|\mathcal{T}|$ and thus $\mathcal{D}$-satisfiability being in PSPACE implies that clashes can be checked for using at most exponential space. It remains to apply the well-known result that EXPSPACE = NEXPSPACE [Savitch 1970].

Point 3 can be proved similarly. ❑

Together with, e.g., Theorem 5.16, we obtain a tight complexity bound for the case of arithmetic concrete domains.

Corollary 5.59. *If $\mathcal{D}$ is an admissible arithmetic concrete domain and $\mathcal{D}$-satisfiability is in* NP, *then $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes is* NEXPTIME-*complete.*

Since subsumption can be reduced to (un)satisfiability, the above results carry over to $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-concept subsumption with the exception that, in Theorem 5.58, Point 1 and Corollary 5.59, "NEXPTIME" has to be replaced by "co-NEXPTIME", and, in Theorem 5.58, Point 3, "2-NEXPTIME" has to be replaced by "co-2-NEXPTIME".

Can the results just established be extended to ABox consistency? In the literature, there exist two main approaches for obtaining ABox consistency algorithms: either reduce ABox consistency to concept satisfiability obtaining a precompletion-style algorithm as in Section 3.2, or "directly" devise a *completion* algorithm for deciding ABox consistency (instead of for deciding concept satisfiability) as done, e.g., in [Haarslev & Möller 2000b; Horrocks *et al.* 2000b]. In the case of $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$, the precompletion approach does not seem to be feasible. The reason for this is the presence of the generalized concrete domain constructors and of the concrete domain role constructor.

To discuss the problems with precompletion algorithms for $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$ in some more detail, let us consider the precompletion algorithm for $\mathcal{ALCF}(\mathcal{D})$ from Section 3.2. In that algorithm, we had to compute the "closure under feature successors" before constructing the reduction concepts, i.e., we had to generate all feature-successors of ABox objects, all feature successors of these feature successors, etc. If we had not done this, the "concrete parts" of models for different reduction concepts would have interacted, which would have resulted in the failure of the reduction. Intuitively, we exploited the facts that the closure under feature successors is of polynomial size and that concrete domain information cannot "cross" (non-feature) role successor relationships. In the case of the generalized concrete domain constructors, this obviously does not work. Since roles are allowed inside the generalized constructors in place of features, concrete domain information may very well cross role successor relationships. This means that we would have to take the closure under role successors, which is equivalent to taking the "direct" approach to ABox reasoning described above.

The problem with predicate roles is that they have a global flavor. If, for example, during the satisfiability checking of a reduction concept we detect that some domain element has to satisfy the concepts $\neg g\uparrow$, $\forall(\exists(g),(g).P).C$ and $\forall(\exists(g),(g).\overline{P}).C$, then all domain elements in models of *every* reduction concept have to satisfy C if they have a

g-successor. Since precompletion algorithms check the satisfiability of each reduction concept separately, it is not clear how such effects can be taken into account.

The described difficulties make it rather unlikely that an upper complexity bound for $\mathcal{ALCP}^{rp,-,\sqcap}(\mathcal{D})$-ABox consistency can be obtained using a precompletion algorithm. Hence, our only choice to establish such a bound would have been to take the direct approach and devise a completion algorithm for deciding ABox consistency right from the start. We have chosen not to do this since the presentation of the completion algorithm and the proofs of correctness and termination are already quite involved and a further extension of the algorithm to ABoxes would have made its presentation even more complex. Thus, we only conjecture that Theorem 5.58 and Corollary 5.59 extend to ABox consistency w.r.t. acyclic TBoxes.

5.5 Comparison with $\mathcal{ALCF}$

In previous chapters, we have already seen that the concrete domain concept constructor and the feature (dis)agreement constructors are closely related. Apart from their obvious syntactical similarity—i.e., the fact that they both take paths as arguments—we observed in Chapter 3 that $\mathcal{ALCF}$ and $\mathcal{ALC}(\mathcal{D})$ can be treated with similar algorithmic techniques and are both PSPACE-complete (if $\mathcal{D}$-satisfiability is in PSPACE). Moreover, as was shown in Sections 4.2 and 5.3.1, the complexity of both $\mathcal{ALCF}$ and $\mathcal{ALC}(\mathcal{D})$ moves from PSPACE-complete to NEXPTIME-complete if acyclic TBoxes are added. In the following, we further elaborate on this correspondence between the complexity of (extensions of) $\mathcal{ALCF}$ and the complexity of (extensions of) $\mathcal{ALC}(\mathcal{D})$. We concentrate on extensions of $\mathcal{ALC}(\mathcal{D})$ that have been considered in Section 5.3 and are also applicable to $\mathcal{ALCF}$, i.e., on the addition of role conjunction and inverse roles. As we shall see, the combination of feature (dis)agreements with role conjunction behaves similarly to the combination of concrete domains with role conjunction, i.e., reasoning becomes NEXPTIME-complete. In contrast, the addition of inverse roles leads to different complexity in the cases of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$: while, in Sections 5.3.3 and 5.4, we proved that $\mathcal{ALC}^{-}(\mathcal{D})$-concept satisfiability is NEXPTIME-complete if $\mathcal{D}$-satisfiability is in NP, we will see in the following that $\mathcal{ALCF}^{-}$-concept satisfiability is undecidable.

5.5.1 $\mathcal{ALCF}^{\sqcap}$-concept Satisfiability

We show that $\mathcal{ALCF}^{\sqcap}$-concept satisfiability is NEXPTIME-hard. The definition of this logic and the corresponding upper bound can be found in Section 4.3. The basic idea is to modify the reduction of the NEXPTIME-hard variant of the domino problem to $\mathcal{ALCF}$-concept satisfiability w.r.t. acyclic TBoxes described in Section 4.2 such that acyclic TBoxes are replaced by role conjunction. Note that this is very similar to what we did in Section 5.3.2, where the NEXPTIME-hardness proof for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability w.r.t. acyclic TBoxes has been adapted to $\mathcal{ALC}^{\sqcap}(\mathcal{D})$-concept satisfiability.

In Section 4.2, the NEXPTIME-hard domino problem was reduced to $\mathcal{ALCF}$-concept satisfiability w.r.t. TBoxes by defining, for each domino system $\mathcal{D}$ and each

$$
\begin{aligned}
\mathsf{Tree}_0[\alpha,\beta,\gamma] \quad := \quad & \exists(R \sqcap \alpha).\top \sqcap \exists(R \sqcap \beta).\top \\
& \sqcap \beta^{n+1}\gamma \downarrow \alpha^{n+1} \sqcap \alpha\beta^{n}\gamma \downarrow \beta\alpha^{n} \\
& \sqcap \forall R.\big(\exists(R \sqcap \alpha).\top \sqcap \exists(R \sqcap \beta).\top \\
& \qquad \sqcap \alpha\beta^{n-1}\gamma \downarrow \beta\alpha^{n-1}\big) \\
& \qquad \vdots \\
& \sqcap \forall R^{n}.\big(\exists(R \sqcap \alpha).\top \sqcap \exists(R \sqcap \beta).\top \\
& \qquad \sqcap \alpha\gamma \downarrow \beta\big) \\
& \sqcap \forall R^{n+1}.\mathsf{Grid}
\end{aligned}
$$

Figure 5.20: The $\mathcal{ALCF}^{\sqcap}$-reduction concept $C_{\mathcal{D},a}$ with $n = |a|$: tree definition. Substitute α, β, γ with f, g, y and f', g', x.

initial condition $a = a_0, \ldots, a_{n-1}$, a concept (name) $C_{\mathcal{D},a}$ and a TBox $\mathcal{T}_{\mathcal{D},a}$ such that models of $C_{\mathcal{D},a}$ and $\mathcal{T}_{\mathcal{D},a}$ have the form of a $2^{n+1} \times 2^{n+1}$-grid representing a torus of the same size that is properly tiled and satisfies the initial condition a. Acyclic TBoxes serve two purposes in this reduction: firstly, they are used to build up the two binary trees of depth $n+1$ whose leaf nodes are connected by chains of features. These two feature chains are row 0 and column 0 of the torus to be defined. Secondly, TBoxes are used to define the rest of the grid once row 0 and column 0 have been established. Very similarly to what was done in Section 5.3.2, the tree definition from Figure 4.4 can be reformulated without reference to TBoxes by using role conjunction. The resulting reduction concept can be found in Figure 5.20. This figure contains an abbreviation rather than a concept definition from a TBox since TBoxes are not available.

It hence remains to adapt the grid definition from Figure 4.6 to $\mathcal{ALCF}^{\sqcap}$ without TBoxes. In the original grid definition, the TBox is used to propagate the Tile_i-concepts to appropriate positions in the grid. Instead of using a TBox, we can also achieve this by using universal value restriction on the diagonals in the grid $z_0, \ldots, z_n$. However, this yields a reduction concept whose size is no longer polynomial in the size of $\mathcal{D}$ and a since we must use universal value restrictions $\forall z_{i_0}. \cdots .\forall z_{i_k}.\mathsf{Tile}_j$ for all prefixes $i_0, \ldots, i_k$ ($k \leq n$) of permutations $i_0, \ldots, i_n$ of $0, \ldots, n$. This blowup can be avoided by using role conjunction to enforce that there exists a role R such that all diagonals in the grid $z_0, \ldots, z_n$ are additionally labeled with R. We may then replace the exponentially many universal value restrictions by polynomially many of the form $\forall R.\mathsf{Tile}_j, \ldots, \forall R^{n+1}.\mathsf{Tile}_j$. With some additional simplifications, this approach yields the reduction concept in Figure 5.21.

The proof of the following lemma is omitted since it is very similar to the proof of Lemma 4.16.

Lemma 5.60. *$C_{\mathcal{D},a}$ is satisfiable w.r.t. $\mathcal{T}_{\mathcal{D},a}$ iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition a where $n = |a|$.*

Together with Corollary 4.23, we obtain the following result.

Theorem 5.61. *$\mathcal{ALCF}^{\sqcap}$-concept satisfiability is* NExpTime*-complete.*

$$
\begin{array}{rcl}
\mathsf{GridAux} & := & \mathsf{Tile} \sqcap xy \downarrow yx \sqcap xy \downarrow z_0 \sqcap \bigsqcap_{0 \le i \le n} \exists(z_i \sqcap R).\top \sqcap \bigsqcap_{0 \le i < n} z_i z_i \downarrow z_{i+1} \\
\mathsf{Grid} & := & \mathsf{GridAux} \sqcap \forall R.\mathsf{GridAux} \sqcap \cdots \sqcap \forall R^{n+1}.\mathsf{GridAux} \\
\mathsf{Tile} & := & \bigsqcup_{t \in T} A_t \sqcap \bigsqcap_{t \in T} \bigsqcap_{t' \in T \setminus \{t\}} \neg(A_t \sqcap A_{t'}) \\
& & \bigsqcap_{t \in T} (A_t \to \exists x. \bigsqcup_{(t,t') \in H} A_{t'}) \\
& & \bigsqcap_{t \in T} (A_t \to \exists y. \bigsqcup_{(t,t') \in V} A_{t'}) \\
\mathsf{Init} & := & \exists \ell_2^{n+1}.(A_{a_0} \sqcap \exists x.(A_{a_1} \sqcap \cdots \sqcap \exists x.(A_{a_{n-2}} \sqcap \exists x.A_{a_{n-1}}) \ldots)) \\
C_{\mathcal{D},a} & := & \mathsf{Tree}_0[\ell_1, r_1, y] \sqcap \mathsf{Tree}_0[\ell_2, r_2, x] \sqcap \ell_1^{n+1} \downarrow \ell_2^{n+1} \sqcap \mathsf{Init}
\end{array}
$$

Figure 5.21: The $\mathcal{ALCF}$ reduction concept $C_{\mathcal{D},a}$ with $n = |a|$: grid definition and tiling.

It is an immediate consequence of Theorem 5.61 that $\mathcal{ALCF}^{\sqcap}$-concept subsumption is co-NExpTime-complete. Similar to the reduction in Section 5.3.2, the one just described relies on the fact that the role conjunction constructor may be applied to features. If we disallow the use of the role conjunction constructor on features, we obtain the logic $\mathcal{ALCF}^{(\sqcap)}$, which is the fusion of the logics $\mathcal{ALCF}$ and and $\mathcal{ALC}^{\sqcap}$. Since concept satisfiability is in PSpace for both these logics (see Chapter 3 and [Donini *et al.* 1997]), we conjecture that $\mathcal{ALCF}^{(\sqcap)}$-concept satisfiability (and subsumption) is also in PSpace. (c.f. Section 5.6).

We should like to add a general comment concerning the complexity of reasoning with Description Logics that provide the role conjunction constructor. This constructor is frequently believed to be "harmless" w.r.t. complexity, i.e., that it can be added to Description Logics without increasing the complexity of reasoning [Donini *et al.* 1997; Tobies 2001b]. In this thesis, we encountered two logics for which this is not the case: $\mathcal{ALC}(\mathcal{D})$ in Section 5.3.2 and $\mathcal{ALCF}$ in this section. Another case is known from the literature since, as shown in [Lutz & Sattler 2000; 2001], the complexity of the logic $\mathcal{ALC}^{\neg}$—$\mathcal{ALC}$ extended with a negation constructor on roles—moves from ExpTime-complete to NExpTime-complete if role conjunction is added. Hence, there seem to be many cases in which role conjunction is indeed "harmless", but also several cases where it has a considerable impact on the complexity of reasoning.

5.5.2 Undecidability of $\mathcal{ALCF}^{-}$

We consider $\mathcal{ALCF}^{-}$, the extension of $\mathcal{ALCF}$ with inverse roles, and show that, for this Description Logic, concept satisfiability is undecidable. The logic $\mathcal{ALCF}^{-}$ is defined in correspondence with $\mathcal{ALC}^{-}(\mathcal{D})$, the extension of $\mathcal{ALC}(\mathcal{D})$ with inverse roles

$$
\begin{aligned}
\mathsf{Grid} &:= \exists f^{-}.\top \ \sqcap\ \forall f^{-}.xy{\downarrow}yx \ \sqcap\ \forall f^{-}.(f{\downarrow}xf \sqcap f{\downarrow}yf \sqcap f{\downarrow}xyf) \\
\mathsf{Tile} &:= (\bigsqcup_{t\in T} A_t) \sqcap \bigsqcap_{t\in T}\ \bigsqcap_{t'\in T\setminus\{t\}} \neg(A_t \sqcap A_{t'}) \\
&\quad \bigsqcap_{t\in T}(A_t \rightarrow \exists x.\ \bigsqcup_{(t,t')\in H} A_{t'}) \\
&\quad \bigsqcap_{t\in T}(A_t \rightarrow \exists y.\ \bigsqcup_{(t,t')\in V} A_{t'}) \\
C_{\mathcal{D}} &:= \mathsf{Grid} \ \sqcap\ \forall f^{-}.\mathsf{Tile}
\end{aligned}
$$

Figure 5.22: The $\mathcal{ALCF}^{-}$ reduction concept $C_{\mathcal{D}}$.

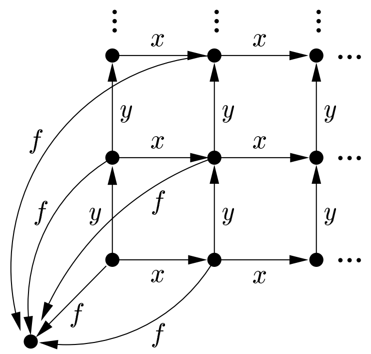

Figure 5.23: Clipping from a model of $C_{\mathcal{D}}$.

introduced in Section 5.3.3. Hence, we may use the inverse constructor both on roles and features but only inside universal and existential value restrictions.

The proof is by a reduction of the general domino problem to $\mathcal{ALCF}^{-}$-concept satisfiability. More precisely, we reduce the domino problem that requires finding a tiling of the first quadrant of the plane. As noted in Section 4.2, this variant is already undecidable [Knuth 1968]. Given a domino system $\mathcal{D} = (T, V, H)$, the reduction concept $C_{\mathcal{D}}$ is defined such that (i) models of $C_{\mathcal{D}}$ have the form of a two-side infinite grid, (ii) every node of the grid is an instance of exactly one of the concept names A_t with $t \in T$ (representing tile types), and (iii) the horizontal and vertical conditions V and H are satisfied. The reduction concept can be found in Figure 5.22 and an example $C_{\mathcal{D}}$ model can be found in Figure 5.23. Again, the figure defines abbreviations which are not to be confused with concept definitions from a TBox. The symbols x, y, and f denote (abstract) features. In the reduction, the Grid concept generates the grid and the Tile concept ensures that the conditions (ii) and (iii) are satisfied.

Lemma 5.62. *$C_{\mathcal{D}}$ is satisfiable iff $\mathcal{D}$ has a solution.*

Proof. Assume that $C_{\mathcal{D}}$ has a model $\mathcal{I}$. We define a solution τ for $\mathcal{D}$. Let $r \in C_{\mathcal{D}}^{\mathcal{I}}$ and $r' \in (f^{-})^{\mathcal{I}}(d)$ (such r and r' exist due to the first conjunct of the Grid concept).

Define the function π from $\mathbb{N}^2$ to $\Delta_\mathcal{I}$ inductively as follows:

1. $\pi(0,0) = r'$,
2. if $\pi(i,j) = d$ and $x^\mathcal{I}(d) = e$, then $\pi(i+1,j) = e$,
3. if $\pi(i,j) = d$ and $y^\mathcal{I}(d) = e$, then $\pi(i,j+1) = e$.

The Grid concept ensures that this function is total and well-defined. Finally, we define $\tau(i,j)$ as the $t \in T$ for which $\pi(i,j) \in A_t^\mathcal{I}$. Note that, due to the first line of Tile, there exists exactly one such t for each $\pi(i,j)$. It is straightforward to check that τ is well-defined and a solution for $\mathcal{D}$.

Conversely, assume that τ is a solution for $\mathcal{D}$. We define a model $\mathcal{I}$ for $C_\mathcal{D}$ as follows:

- $\Delta_\mathcal{I} = \mathbb{N}^2 \cup \{d\}$,
- $x^\mathcal{I}(i,j) = (i+1,j)$ for all $i,j \in \mathbb{N}$,
- $y^\mathcal{I}(i,j) = (i,j+1)$ for all $i,j \in \mathbb{N}$,
- $f^\mathcal{I}(i,j) = d$ for all $i,j \in \mathbb{N}$,
- $A_t^\mathcal{I} = \tau^{-1}(t)$ for all $t \in T$.

It is straightforward to verify that $\mathcal{I}$ is a model of $C_\mathcal{D}$. ❑

The following theorem is an immediate consequence of Lemma 5.62 and the undecidability of the domino problem.

Theorem 5.63. *$\mathcal{ALCF}^-$-concept satisfiability is undecidable.*

Since (un)satisfiability can be reduced to subsumption, $\mathcal{ALCF}^-$-concept subsumption is clearly also undecidable. Again, the reduction is only possible since the inverse constructor may be applied to features. It does not work for $\mathcal{ALCF}^{(-)}$, which is obtained from $\mathcal{ALCF}^-$ by disallowing the use of the inverse constructor on features. In fact, since $\mathcal{ALCF}^{(-)}$ is nothing but the fusion of $\mathcal{ALCF}$ and $\mathcal{ALC}^-$, it follows from the results in [Baader *et al.* 2002a] that $\mathcal{ALCF}^{(-)}$-concept satisfiability is decidable. Moreover, since both $\mathcal{ALCF}$ and $\mathcal{ALC}^-$ are PSPACE-complete (see Chapter 3 and [Horrocks *et al.* 2000a; Spaan 1993b]), we conjecture that $\mathcal{ALCF}^{(-)}$-concept satisfiability is also PSPACE-complete.

5.6 Discussion

We have seen that the PSPACE upper bound for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability from Chapter 3 is not robust in the sense that, for many seemingly simple extensions of $\mathcal{ALC}(\mathcal{D})$ and a large class of concrete domains, reasoning becomes NEXPTIME-complete. However, there also exist several rather expressive concrete domains and extensions of $\mathcal{ALC}(\mathcal{D})$ to which the obtained results are *not* applicable:

1. Some interesting concrete domains, such as the temporal concrete domains P and $\mathtt{I}$ from Section 2.4.3, are not arithmetic. Hence, none of the hardness results obtained in this chapter applies to extensions of $\mathcal{ALC}(\mathsf{P})$ and $\mathcal{ALC}(\mathtt{I})$.

2. We gave no NExpTime-hardness results for extensions of $\mathcal{ALC}(\mathcal{D})$ with some standard DL constructors such as transitive roles and qualifying number restrictions.

Concerning the first point, it seems that many extensions of $\mathcal{ALC}(\mathsf{P})$ and $\mathcal{ALC}(\mathtt{I})$ are indeed not NExpTime-hard. This is illustrated in the following chapter, where general TBoxes are added to $\mathcal{ALC}(\mathsf{P})$, and the resulting logic is proved to be decidable and ExpTime-complete. Using the reduction of $\mathcal{ALC}(\mathtt{I})$-concept satisfiability to $\mathcal{ALC}(\mathsf{P})$-concept satisfiability sketched in Section 2.4.3, one can then show that this upper bound also applies to $\mathcal{ALC}(\mathtt{I})$ with general TBoxes.

Concerning the second point, let us discuss the extension of $\mathcal{ALC}(\mathcal{D})$ with transitive roles and qualifying number restrictions on an informal level. As in previous sections, it is convenient to use the fact that some Description Logics can be viewed as the *fusion* of other Description Logics to obtain decidability results and conjecture complexity results. However, it is out of the scope of this thesis to give a formal definition of the notion "fusion" and we refer the interested reader to, e.g., [Baader *et al.* 2002a; Kracht & Wolter 1991; Spaan 1993a]. Intuitively, the fusion $\mathcal{L}$ of two Description Logics $\mathcal{L}_1$ and $\mathcal{L}_2$ is obtained by combining $\mathcal{L}_1$ and $\mathcal{L}_2$ in a way such that their constructors do not "interact". As an example, consider the logics $\mathcal{ALCF}^-$ and $\mathcal{ALCF}^{(-)}$ introduced in Section 5.5.2, which are both combinations of $\mathcal{ALCF}$ and $\mathcal{ALC}^-$. While $\mathcal{ALCF}^{(-)}$ is the fusion of these two logics, $\mathcal{ALCF}^-$ is "more" than the fusion: feature agreements interact with the inverse role constructor since we allow the inverse constructor to be applied to features. This difference is reflected by the computational properties of the two logics: in contrast to $\mathcal{ALCF}^-$, which was proven undecidable in Section 5.5.2, we can use the general transfer results from [Baader *et al.* 2002a] to deduce that $\mathcal{ALCF}^{(-)}$-concept satisfiability is decidable. More precisely, the transfer results in [Baader *et al.* 2002a] state the following:

- if $\mathcal{L}_1$-concept satisfiability and $\mathcal{L}_2$-concept satisfiability are decidable and $\mathcal{L}$ is the fusion of $\mathcal{L}_1$ and $\mathcal{L}_2$, then $\mathcal{L}$-concept satisfiability is decidable;

- if $\mathcal{L}_1$-concept satisfiability w.r.t. general TBoxes and $\mathcal{L}_2$-concept satisfiability w.r.t. general TBoxes are decidable and $\mathcal{L}$ is the fusion of $\mathcal{L}_1$ and $\mathcal{L}_2$, then $\mathcal{L}$-concept satisfiability w.r.t. general TBoxes is decidable.

Unfortunately, Baader et al. do not provide transfer results for the complexity of reasoning, and, indeed, it seems that the methods used in [Baader *et al.* 2002a] will in many cases not yield a tight upper complexity bound.

The mentioned transfer results can be used to prove decidability of extensions of $\mathcal{ALC}(\mathcal{D})$ with transitive roles and qualifying number restrictions. For example, it is easily seen that $\mathcal{ALCQ}^-_{R^+}(\mathcal{D})$, i.e., the extension of $\mathcal{ALC}(\mathcal{D})$ with qualifying number restrictions, the inverse role constructor, and transitive roles, is the fusion of the Description Logics $\mathcal{ALC}^-(\mathcal{D})$ and $\mathcal{ALCQ}^-_{R^+}$—more precisely, this is only true if the use of features inside qualifying number restrictions is prohibited, which obviously is no restriction w.r.t. expressive power. Since $\mathcal{ALC}^-(\mathcal{D})$-concept satisfiability was proven decidable in Section 5.4 and $\mathcal{ALCQ}^-_{R^+}$-concept satisfiability is also decidable [Horrocks *et al.* 2000a], we obtain the following result:[9]

Theorem 5.64. *If $\mathcal{D}$ is admissible, then $\mathcal{ALCQ}^-_{R^+}(\mathcal{D})$-concept satisfiability is decidable.*

Although this does not follow from the results in [Baader *et al.* 2002a], in many cases the complexity of component logics also transfers to their fusion. In the area of Modal Logics, this has, e.g., been shown by Spaan [1993a]. More precisely, Spaan proves the following result:

- if $\mathcal{L}_1$ and $\mathcal{L}_2$ are Modal Logics, $\mathcal{L}$ is the fusion of $\mathcal{L}_1$ and $\mathcal{L}_2$, C is a complexity class that contains PSPACE, and satisfiability of $\mathcal{L}_i$-formulas under restrictions is in C for $i \in \{1, 2\}$, then satisfiability of $\mathcal{L}$-formulas under restrictions is also in C.

Roughly spoken, "$\mathcal{L}$-formula satisfiability under restrictions" means to decide whether, for a given $\mathcal{L}$-formula φ and a given set of $\mathcal{L}$-formulas $\mathcal{R}$, φ is satisfiable by a model $\mathcal{M}$ such that, for every set Ψ of formulas verified by a world w of $\mathcal{M}$, the intersection of Ψ with a certain set of formulas "relevant at w" is in $\mathcal{R}$. Although Spaan's notion of "Modal Logic" does not capture most of the Description Logics used in this thesis, a close inspection of Spaan's results gives reason to assume that they can be extended to a large class of Description Logics. Moreover, it seems possible to show that, for most Description Logics $\mathcal{L}$ used in this thesis, $\mathcal{L}$-concept satisfiability being in a complexity class C implies that $\mathcal{L}$-concept satisfiability under restrictions is also in C. These observations are the basis for the following conjectures.

For the logic $\mathcal{ALCQ}^-_{R^+}(\mathcal{D})$ from above, we conjecture concept satisfiability to be in NEXPTIME if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in NP. The reason for this is that the component $\mathcal{ALC}^-(\mathcal{D})$ is in NEXPTIME in this case (Theorem 5.52) and the component $\mathcal{ALCQ}^-_{R^+}$ is EXPTIME-complete even if the numbers inside number restrictions are coded binarily [Tobies 2001a]. In search of a maximally expressive PSPACE Description Logic with concrete domains, we give another conjecture. It is not hard to see that $\mathcal{ALCFN}_{R^+}(\mathcal{D})$, the extension of $\mathcal{ALC}(\mathcal{D})$ with feature (dis)agreements, unqualifying number restrictions, and transitive roles, is the fusion of the Description Logics $\mathcal{ALCF}(\mathcal{D})$ and $\mathcal{ALCN}_{R^+}$ (if the use of features inside qualifying number restrictions is prohibited). Since, in Chapter 3, we proved $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability to be PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE, and $\mathcal{ALCN}_{R^+}$ is also PSPACE-complete (with binary coding of numbers and if transitive roles are not used inside number restrictions [Horrocks *et al.* 1998]), we conjecture that $\mathcal{ALCFN}_{R^+}(\mathcal{D})$-concept satisfiability is PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE. However, it seems that the results in [Horrocks *et al.* 1998] can be extended to $\mathcal{ALCQ}_{R^+}$, and, thus, $\mathcal{ALCFQ}_{R^+}(\mathcal{D})$ is another candidate for a very expressive PSPACE Description Logic with concrete domains.

[9] $\mathcal{ALCQ}^-_{R^+}$-concept satisfiability is only known to be decidable if the use of transitive roles inside qualifying number restrictions is prohibited. Hence, the same restriction applies to the logic $\mathcal{ALCQ}^-_{R^+}(\mathcal{D})$.

Chapter 6

Concrete Domains and General TBoxes

In Chapter 5, we obtained tight complexity bounds for reasoning with concrete domains and acyclic TBoxes. However, for many applications it is desirable to have at one's disposal the strictly more powerful general TBoxes introduced in Section 2.2.1. Striving for more expressive Description Logics with concrete domains, in this chapter we investigate the possibility of combining concrete domains and general TBoxes without losing decidability. The first result we obtain is a negative one since it states that, for every arithmetic concrete domain $\mathcal{D}$, $\mathcal{ALC}(\mathcal{D})$-concept satisfiability w.r.t. general TBoxes is undecidable. Retrospectively, this strong undecidability result provides a major justification for our detailed investigation of Description Logics with concrete domains and acyclic TBoxes in Chapter 5. However, as the main result of this chapter will demonstrate, the situation concerning concrete domains and general TBoxes is not hopeless in all cases: we prove that $\mathcal{ALC}(\mathsf{P})$-concept satisfiability w.r.t. general TBoxes and $\mathcal{ALC}(\mathsf{P})$-ABox consistency w.r.t. general TBoxes are decidable in deterministic exponential time, where P is the point-based temporal concrete domain introduced in Section 2.4.3. Using an encoding technique as in the reduction of $\mathcal{ALC}(\mathtt{I})$-concept satisfiability to $\mathcal{ALC}(\mathsf{P})$-concept satisfiability sketched in Section 2.4.3 (where $\mathtt{I}$ is the interval-based temporal concrete domain from Section 2.4.3), it can then be shown that these upper bounds also apply to $\mathcal{ALC}(\mathtt{I})$, or, viewed from a different perspective, that $\mathcal{ALC}(\mathsf{P})$ can be used for combined point-based and interval-based temporal reasoning.

It is important to note that the temporal Description Logic $\mathcal{ALC}(\mathsf{P})$ is not only interesting because it provides an example for the case where a concrete domain can be combined with general TBoxes without losing decidability. The obtained decidability results are interesting in their own right since, to the best of our knowledge, for the first time we provide a combination of interval-based temporal reasoning and reasoning with general TBoxes in a decidable Description Logic. Before proving decidability, we illustrate the usefulness of this combination by describing a general framework for the representation of conceptual temporal knowledge with $\mathcal{ALC}(\mathsf{P})$ and then applying this framework exemplarily in the application domain of process engineering [Sattler

$$
\begin{array}{lcl}
C_P & := & \exists \ell. =_\epsilon \ \sqcap \exists r. =_\epsilon \\
\mathcal{T}_P & := & \{\top \sqsubseteq \displaystyle\bigsqcap_{(\ell_i, r_i) \in P} \exists \ell, f_i \ell.\mathsf{conc}_{\ell_i} \ \sqcap \ \exists r, f_i r.\mathsf{conc}_{r_i} \\
& & \ \ \top \sqsubseteq \exists \ell. =_\epsilon \ \sqcup \neg \exists \ell, r.= \}
\end{array}
$$

Figure 6.1: The $\mathcal{ALC}(\mathsf{W})$ reduction concept C_P and TBox $\mathcal{T}_P$.

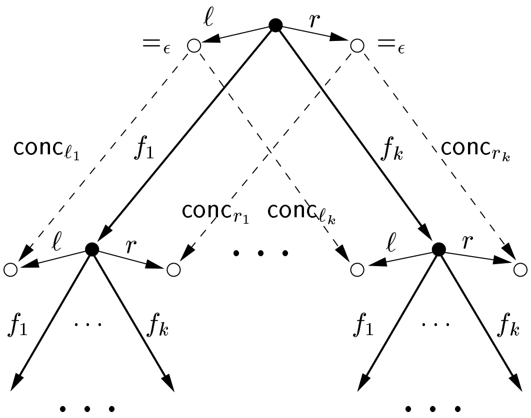

Figure 6.2: An example model of C w.r.t. $\mathcal{T}$

1998; Molitor 2000]. A comparison with existing temporal Description Logics shows that $\mathcal{ALC}(\mathsf{P})$ with general TBoxes significantly extends the expressive power of other interval-based temporal Description Logics known from the literature and thus has the potential to become a valuable tool in many application areas.

6.1 An Undecidability Result

We prove that $\mathcal{ALC}(\mathsf{W})$-concept satisfiability is undecidable, where W is the concrete domain for encoding Post Correspondence Problems introduced in Section 5.2. This is done by reducing the general, undecidable PCP to $\mathcal{ALC}(\mathsf{W})$-concept satisfiability using a technique similar to the ones employed in Section 5.3. As in Section 5.3, W can be replaced by any arithmetic concrete domain.

Given an instance of the Post Correspondence Problem $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$, the idea is to define a concept C_P and a TBox $\mathcal{T}_P$ such that models of C_P and $\mathcal{T}_P$ represent all potential solutions of P, i.e., all corresponding left and right concatenations of words from P induced by sequences of indices of arbitrary length. Moreover, the definition of C_P and $\mathcal{T}_P$ ensures that none of the potential solutions is in fact a solution. Hence, P has a solution iff C_P is unsatisfiable w.r.t. $\mathcal{T}_P$. The reduction concept and TBox can be found in Figure 6.1, where ℓ and r denote concrete features. An example model is displayed in Figure 6.2.

Lemma 6.1. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP. Then P has a solution iff the concept C_P is unsatisfiable w.r.t. the TBox $\mathcal{T}_P$.*

Proof. For both directions, we show the contrapositive. Hence, to show the "if" direction, assume that P has no solution. We show that C_P is satisfiable w.r.t. $\mathcal{T}_P$ by constructing an interpretation $\mathcal{I}$ that is a model of C_P and $\mathcal{T}_P$. If $w = i_1, \ldots, i_n$ is a sequence of indices, we use $\mathsf{leftconc}(w)$ to denote the concatenation of the words $\ell_{i_1}, \ldots, \ell_{i_n}$ and $\mathsf{rightconc}(w)$ to denote the concatenation of the words $r_{i_1}, \ldots, r_{i_n}$. We define

$$
\begin{array}{rcl}
\Delta_{\mathcal{I}} & := & \{i_1 \cdots i_n \mid n \geq 0 \text{ and } 1 \leq i_j \leq k \text{ for } 1 \leq j \leq n\}, \\
f_i(w) & := & wi \text{ for } w \in \Delta_{\mathcal{I}} \text{ and } 1 \leq i \leq k, \\
\ell(w) & := & \mathsf{leftconc}(w) \text{ for } w \in \Delta_{\mathcal{I}}, \\
r(w) & := & \mathsf{rightconc}(w) \text{ for } w \in \Delta_{\mathcal{I}}.
\end{array}
$$

Note that $\Delta_{\mathcal{I}}$ also contains the empty sequence of indices. Since P has no solution, it is readily checked that $\mathcal{I}$ is a model of C_P and $\mathcal{T}_P$.

Now for (the contrapositive of) the "only if" direction. Let $\mathcal{I}$ be a model of C_P and $\mathcal{T}_P$ with $d \in C_P^{\mathcal{I}}$. We must show that P has no solution. Assume to the contrary that $w = i_1, \ldots, i_n$ is a solution for P. By induction on j, it is easy to show that $(f_{i_1} \cdots f_{i_j} \ell)^{\mathcal{I}} = \mathsf{leftconc}(w)$ and $(f_{i_1} \cdots f_{i_j} r)^{\mathcal{I}} = \mathsf{rightconc}(w)$ for $1 \leq j \leq n$. Since $\mathcal{I}$ is a model of $\mathcal{T}_P$ and $n \geq 1$, we clearly have $(f_{i_1} \cdots f_{i_n} \ell)^{\mathcal{I}} \neq (f_{i_1} \cdots f_{i_n} r)^{\mathcal{I}}$ and thus $\mathsf{leftconc}(w) \neq \mathsf{rightconc}(w)$. Hence, w is no solution to P, a contradiction. ❑

The following theorem is an immediate consequence of the described reduction:

Theorem 6.2. *$\mathcal{ALC}(\mathsf{W})$-concept satisfiability w.r.t. general TBoxes is undecidable.*

As already mentioned, W can be replaced by any arithmetic concrete domain (see Section 5.2).

Corollary 6.3. *For every arithmetic concrete domain $\mathcal{D}$, $\mathcal{ALC}(\mathcal{D})$-concept satisfiability w.r.t. general TBoxes is undecidable.*

These undecidability results obviously also apply to concept subsumption and ABox consistency since concept satisfiability can be reduced to these tasks. It should be noted that the above proof is similar to Baader and Hanschke's undecidability proof for $\mathcal{ALC}(\mathcal{D})$ extended with a transitive closure role constructor [Baader & Hanschke 1992]. Moreover, the reduction resembles the one used in [Lutz & Möller 1997] to prove undecidability of unrestricted $\mathcal{ALC}^{rp}(\mathcal{D})$ (see Section 2.3.2). In all three cases, being able to "reach" every domain element in connected models from the root together with the expressive power of the concrete domain is the cause of undecidability.

6.2 $\mathcal{ALC}(\mathsf{P})$ with General TBoxes

Despite the discouraging undecidability result established in the previous section, there exist interesting concrete domains $\mathcal{D}$ such that reasoning with $\mathcal{ALC}(\mathcal{D})$ and general TBoxes is decidable. In this section, we show that the temporal concrete domain P introduced in Section 2.4.3 has this desirable property by proving that $\mathcal{ALC}(\mathsf{P})$-concept

$$
\begin{array}{rcl}
\mathsf{ATemporal} & \doteq & t\uparrow \sqcap \ell\uparrow \sqcap r\uparrow \\
\mathsf{Temporal} & \doteq & \mathsf{Point} \sqcup \mathsf{Interval} \\
\mathsf{Point} & \doteq & \exists t, t.{=} \sqcap \ell\uparrow \sqcap r\uparrow \\
\mathsf{Interval} & \doteq & \exists \ell, r.{<} \sqcap t\uparrow \\
\top & \doteq & (\exists \ell, \ell.{=} \sqcup \exists r, r.{=}) \rightarrow \mathsf{Interval} \\
 & & \sqcap\ \exists t, t.{=} \rightarrow \mathsf{Point}
\end{array}
$$

Figure 6.3: TBox $\mathcal{T}^*$ with basic definitions of the framework.

satisfiability and $\mathcal{ALC}(\mathsf{P})$-ABox consistency can be decided in deterministic exponential time. Before presenting the decision procedures, we illustrate the relevance of our results by motivating $\mathcal{ALC}(\mathsf{P})$ as a powerful tool for conceptual temporal representation and reasoning. More precisely, we first introduce a general framework for the representation of conceptual temporal knowledge with $\mathcal{ALC}(\mathsf{P})$ and then apply this framework in the application area of process engineering.

6.2.1 Temporal Reasoning with $\mathcal{ALC}(\mathsf{P})$

We present a framework for the representation of conceptual temporal knowledge with the Description Logic $\mathcal{ALC}(\mathsf{P})$. The most important aspect addressed by this framework is the fact that $\mathcal{ALC}(\mathsf{P})$ cannot only be used for point-based temporal reasoning but can also be viewed as a full-fledged interval-based temporal Description Logic. Moreover, it is even possible to freely combine point-based and interval-based reasoning. The introduced representation framework once more illustrates the close connection between the point-based concrete domain P and the interval-based concrete domain $\mathtt{I}$ (see Section 2.4.3).

The representation framework consists of several conventions and abbreviations. We assume that each entity of the application domain is either temporal or atemporal. If it is temporal, its temporal extension may be either a time point or an interval. We generally assume that left endpoints of intervals are represented by the concrete feature ℓ, right endpoints of intervals are represented by the concrete feature r, and time-points not related to intervals are represented by the concrete feature t. All this can be expressed by the TBox $\mathcal{T}^*$ displayed in Figure 6.3. The first four concept equations in the TBox define the relevant notions while the fifth equation ensures that all domain elements for which one of the concrete features ℓ, r, or t is defined is either an Interval or a Point. The TBox implies that the concepts $\mathsf{ATemporal}$, Point, and $\mathsf{Interval}$ are mutually disjoint, and that their disjunction is equivalent to $\top$.

Interval-based reasoning with $\mathcal{ALC}(\mathsf{P})$ is based on Allen's relations, which have been introduced in Section 2.4.3. The Allen relations can be defined in terms of their endpoints as in the reduction of $\mathcal{ALC}(\mathtt{I})$-concept satisfiability to $\mathcal{ALC}(\mathsf{P})$-concept satisfiability sketched in Section 2.4.3. To keep concepts readable, we define a suitable abbreviation for each of the 13 basic relations. For example,

$$\exists(p, p').\mathsf{contains} \text{ abbreviates } \exists p\ell, p'\ell.{<} \sqcap \exists p'r, pr.{<}$$

where p and p' are abstract paths. Note that we have

$$\exists(p,p').\mathsf{contains} \quad \sqsubseteq_{\mathcal{T}^*} \quad \exists p.\mathsf{Interval} \sqcap \exists p'.\mathsf{Interval}.$$

Similar abbreviations are introduced for the other basic relations and the defining concepts can be read off from Figure 2.7. We do not introduce explicit abbreviations for generalized Allen relations (c.f. Section 2.4.3), but these relations can clearly be expressed in $\mathcal{ALC}(\mathsf{P})$ by using the disjunction concept constructor. In what follows, we use self to denote the empty abstract path. For example,

$$\exists(p,\mathsf{self}).\mathsf{starts} \text{ abbreviates } \exists p\ell,\ell.= \ \sqcap \ \exists pr,r.<.$$

Intuitively, self refers to the interval associated with the domain element at which the $\exists(p,\mathsf{self}).\mathsf{starts}$ concept is "evaluated".

Since we have intervals *and* points at our disposal, we should be able to talk about the relationship between points and intervals. More precisely, there exist 5 possible relations between a point and an interval and we introduce the following abbreviations for them:

$$\begin{array}{rll}
\exists(p,p').\mathsf{beforep} & \text{for} & \exists pt,p'\ell.< \\
\exists(p,p').\mathsf{startsp} & \text{for} & \exists pt,p'\ell.= \\
\exists(p,p').\mathsf{duringp} & \text{for} & \exists p'\ell,pt.< \ \sqcap \ \exists pt,p'r.< \\
\exists(p,p').\mathsf{finishesp} & \text{for} & \exists pt,p'r.= \\
\exists(p,p').\mathsf{afterp} & \text{for} & \exists p'r,pt.<
\end{array}$$

where p and p' are again abstract paths. We refrain from defining abbreviations for the inverses of these relations since they can easily be expressed by exchanging the arguments in the above abbreviations.

Given the abbreviations introduced in this section and the reductions given in Section 2.4.3, it should be clear that $\mathcal{ALC}(\mathtt{I})$-concept satisfiability w.r.t. general TBoxes can be reduced to $\mathcal{ALC}(\mathsf{P})$-concept satisfiability w.r.t. general TBoxes and $\mathcal{ALC}(\mathtt{I})$-ABox consistency w.r.t. general TBoxes can be reduced to $\mathcal{ALC}(\mathsf{P})$-ABox consistency w.r.t. general TBoxes. Hence, the decidability and complexity results obtained in the following subsections also apply to reasoning with $\mathcal{ALC}(\mathtt{I})$.

The usefulness of the introduced framework is demonstrated in the next subsection.

6.2.2 A Modelling Example

Interval-based temporal Description Logics have been used in various application areas such as reasoning about actions and plans [Artale & Franconi 2001; 1998] and disaster management [Kullmann *et al.* 2000]. We claim that $\mathcal{ALC}(\mathsf{P})$ is a valuable contribution to most of these application areas since, unlike existing interval-based Description Logics, it admits general TBoxes. To substantiate this claim, we motivate $\mathcal{ALC}(\mathsf{P})$ as an appropriate tool for temporal reasoning in the area of process engineering. In [Sattler 1998] and [Molitor 2000], it is described how Description Logics can be used for knowledge representation and reasoning in this application domain. However, in

$$
\begin{aligned}
\mathsf{Week} \doteq\; & \mathsf{Interval} \sqcap \\
& \bigsqcap_{1 \le i \le 7} \exists \mathsf{day}_i.\mathsf{Day} \sqcap \\
& \exists(\mathsf{day}_1, \mathsf{self}).\mathsf{starts} \sqcap \exists(\mathsf{day}_7, \mathsf{self}).\mathsf{finishes} \sqcap \\
& \bigsqcap_{1 \le i < 7} \exists(\mathsf{day}_i, \mathsf{day}_{i+1}).\mathsf{meets} \sqcap \\
& \exists \mathsf{next}.\mathsf{Week} \sqcap \exists(\mathsf{self}, \mathsf{next}).\mathsf{meets} \\
\mathsf{Day} \sqsubseteq\; & \mathsf{Interval}
\end{aligned}
$$

Figure 6.4: Weeks and Days.

Sattler's and Molitor's approach, only static knowledge about process engineering is considered, i.e., there is no explicit representation of temporal relationships. We use the framework presented in the previous section to show how the temporal aspects of this application domain can be represented in $\mathcal{ALC}(\mathsf{P})$, thus refining Sattler's and Molitor's model.

Our goal is to represent information about an automated chemical production process that is carried out by some complex technical device. The device operates each day for some time, depending on the output quantity that is to be produced. It needs complex startup and shutdown phases before and after operation. Moreover, some weekly maintenance is needed to keep the device functional.

Let us first represent the underlying temporal structure consisting of weeks and days. The corresponding TBox can be found in Figure 6.4. In the figure, we use $C \sqsubseteq D$ as an abbreviation for $\top \doteq (C \rightarrow D)$. The first concept equation states that each week consists of seven days, where the i-th day is accessible from the corresponding week via the abstract feature day_i. The temporal relationship between the days are as expected: Monday starts the week, Sunday finishes it, and each day temporally meets the succeeding one. Moreover, each week has a successor week (accessible via the abstract feature next) that it temporally meets. The TBox clearly implies that days 2 to 6 are during the corresponding week although this is not explicitly stated. Note that the TBox is cyclic since Week is defined in terms of itself. Indeed, it is easy to see that such an (infinite) temporal structure cannot be described using acyclic TBoxes.

Figure 6.5 defines the startup, operation, shutdown, and maintenance phases, where start, op, shut, and maint are abstract features and "$\circ$" is used as a separator for the features in (concrete and abstract) paths for better readability. In lines 2 to 5 of the concept equation for Day, we freely combine abbreviations from the framework with predicates from P itself to ensure succinct definitions. Taken together, these lines imply that phases are related to the corresponding day as follows: startup via starts or during, shutdown via during or $\mathsf{finishes}$, and operation via during. Moreover, the startup phase meets the operation phase, which in turn meets the shutdown phase.

Until now, we did not say anything about the temporal relationship of maintenance and operation. This may be inadequate, if, for example, maintenance and operation

$$
\begin{aligned}
\mathsf{Day} \sqsubseteq\ & \exists \mathsf{start}.\mathsf{Startup} \sqcap \exists \mathsf{op}.\mathsf{Operation} \sqcap \exists \mathsf{shut}.\mathsf{Shutdown} \sqcap \\
& \exists \mathsf{start} \circ \ell, \ell.\geq \sqcap \\
& \exists (\mathsf{start}, \mathsf{op}).\mathsf{meets} \sqcap \\
& \exists (\mathsf{op}, \mathsf{shut}).\mathsf{meets} \sqcap \\
& \exists \mathsf{shut} \circ r, r.\leq \\
\mathsf{Week} \sqsubseteq\ & \exists \mathsf{maint}.\mathsf{Maintenance} \sqcap \exists (\mathsf{self}, \mathsf{maint}).\mathsf{contains} \\
\mathsf{Interval} \sqsupseteq\ & \mathsf{Startup} \sqcup \mathsf{Operation} \sqcup \mathsf{Shutdown} \sqcup \mathsf{Maintenance}
\end{aligned}
$$

Figure 6.5: Operation and Maintenance.

are mutually exclusive. We can take this into account by using the additional concept equation

$$
\mathsf{Week} \sqsubseteq \bigsqcap_{1 \leq i \leq 7} \big(\exists (\mathsf{maint}, \mathsf{day}_i \circ \mathsf{op}).\mathsf{before} \sqcup \exists (\mathsf{maint}, \mathsf{day}_i \circ \mathsf{op}).\mathsf{after} \big) \tag{$*$}
$$

which expresses that the weekly maintenance phase must be either before or after the operation phase of every weekday. It is not hard to check that this is the case if and only if the weekly maintenance phase does not overlap the operation phase of any weekday.

This finishes the modelling of the basic properties of our production process. Let us define some more advanced concepts to illustrate reasoning with $\mathcal{ALC}(\mathsf{P})$. For example, we can define a busy week as follows:

$$
\mathsf{BusyWeek} \doteq \mathsf{Week} \sqcap \bigsqcap_{1 \leq i \leq 7} \big(\exists (\mathsf{day}_i \circ \mathsf{start}, \mathsf{day}_i).\mathsf{starts} \sqcap \exists (\mathsf{day}_i \circ \mathsf{shut}, \mathsf{day}_i).\mathsf{finishes} \big)
$$

The concept equation says that on every day of a busy week, the startup phase starts at the beginning of the day and the shutdown finishes at the end of the day. Say now that it is risky to do maintenance during startup and shutdown phases and define

$$
\mathsf{RiskyWeek} \doteq \mathsf{Week} \sqcap \neg \bigsqcap_{1 \leq i \leq 7} \big(\exists (\mathsf{day}_i \circ \mathsf{start}, \mathsf{maint}).\mathsf{before} \sqcup \exists (\mathsf{day}_i \circ \mathsf{shut}, \mathsf{maint}).\mathsf{after} \big)
$$

expressing that, in a risky week, the maintenance phase is not strictly separated from the startup and shutdown phases. An $\mathcal{ALC}(\mathsf{P})$ reasoner could be used to detect that $\mathsf{BusyWeek} \sqsubseteq \mathsf{RiskyWeek}$, i.e., every busy week is a risky week: in a busy week, every day of the week is partitioned into startup, shutdown, and operation phases. Since maintenance may not overlap with operation phases by $(*)$, it must overlap with startup and/or shutdown phases, which means that the week is a risky week.

In order to demonstrate combined reasoning with time points and intervals, we propose a further refinement of our model. Assume that the production process is fully automated except that an operator interaction is necessary to initiate the startup and shutdown phases. This is described by the concept equations in Figure 6.6,

$$\begin{aligned}\mathsf{Day} \sqsubseteq\ & \exists \mathsf{up\text{-}int}.\mathsf{Interaction} \sqcap \exists \mathsf{down\text{-}int}.\mathsf{Interaction} \sqcap \\ & \exists(\mathsf{up\text{-}int} \circ t, \mathsf{start}).\mathsf{startsp} \\ & \exists(\mathsf{down\text{-}int} \circ t, \mathsf{shut}).\mathsf{startsp} \\ \mathsf{Interaction} \sqsubseteq\ & \mathsf{Point}\end{aligned}$$

Figure 6.6: Operator interaction.

where up-int and down-int are abstract features. Note that the operator interaction is represented by a time point instead of a time interval. To illustrate reasoning, assume that, on Friday of calendar week 23, a shutdown interaction was performed by the maintenance team:

$$\mathsf{Week23} \sqsubseteq \mathsf{Week} \sqcap \exists(\mathsf{day}_5 \circ \mathsf{down\text{-}int}, \mathsf{maint}).\mathsf{duringp}.$$

It is not hard to see that this is inconsistent with the description of faultless operation from above, i.e., that Week23 is unsatisfiable: the shutdown interaction finishes the operation phase (since it starts the shutdown phase and the operation phase meets the shutdown phase), which means that the maintenance phase, during which the shutdown interaction was performed, is not strictly separated from the operation phase. This separateness, however, is required by $(*)$ since maintenance and operation are mutually exclusive. Hence, unsatisfiability of Week23 allows us to conclude that something went wrong on the Friday of calendar week 23.

6.2.3 Deciding Concept Satisfiability

In this section, we prove satisfiability of $\mathcal{ALC}(\mathsf{P})$-concepts w.r.t. general TBoxes to be decidable and obtain a tight ExpTime complexity bound for this problem. We use an automata-theoretic approach: first, models are abstracted to so-called Hintikka-trees such that there exists a model for a concept C and a TBox $\mathcal{T}$ iff there exists a Hintikka-tree for C and $\mathcal{T}$. Then we build, for each $\mathcal{ALC}(\mathsf{P})$-concept C and TBox $\mathcal{T}$, a looping tree automaton $\mathcal{A}_{(C,\mathcal{T})}$ (i.e., a Büchi tree automaton without Büchi condition) that accepts exactly the Hintikka-trees for C and $\mathcal{T}$. Hence, $\mathcal{A}_{(C,\mathcal{T})}$ accepts the empty (tree-) language iff C is unsatisfiable w.r.t. $\mathcal{T}$.

Throughout this section, we assume that $\mathcal{ALC}(\mathsf{P})$-concepts and TBoxes contain at most the concrete predicates $<$ and $=$. It is easy to see that this can be done without loss of generality since other predicates can be eliminated by exhaustively applying the following rewrite rules:

$$\begin{aligned}\exists u.\top_{\mathsf{P}} &\rightsquigarrow \exists u, u.= \\ \exists u.\bot_{\mathsf{P}} &\rightsquigarrow \bot \\ \exists u_1, u_2.\leq &\rightsquigarrow \exists u_1, u_2.< \sqcup \exists u_1, u_2.= \\ \exists u_1, u_2.\geq &\rightsquigarrow \exists u_1, u_2.> \sqcup \exists u_1, u_2.= \\ \exists u_1, u_2.\neq &\rightsquigarrow \exists u_1, u_2.> \sqcup \exists u_1, u_2.<\end{aligned}$$

For what follows, it is interesting to note that $\mathcal{ALC}(\mathsf{P})$ with general TBoxes lacks the finite model property since there exist satisfiable TBoxes such as $\top \doteq \exists g, fg.<$ having only infinite models (due to the semantics of the "$<$" predicate). Hence, Hintikka-trees and most other structures used for deciding satisfiability are (potentially) infinite.

Preliminaries

We introduce the basic notions needed for the automata-theoretic satisfiability algorithm like infinite trees, looping automata, and the language they accept. We also introduce constraint graphs which will be needed to take into account concrete domains when defining Hintikka trees.

Definition 6.4. Let M be a set and $k \geq 1$. A *k-ary M-tree* is a mapping $T : \{1, \ldots, k\}^* \to M$ that labels each node $\alpha \in \{1, \ldots, k\}^*$ with $T(\alpha) \in M$. Intuitively, the node αi is the i-th child of α. We use ϵ to denote the empty word (corresponding to the root of the tree).

A *looping automaton* $\mathcal{A} = (Q, M, I, \Delta)$ for k-ary M-trees is defined by a finite set Q of *states*, a finite alphabet M, a subset $I \subseteq Q$ of *initial states*, and a *transition relation* $\Delta \subseteq Q \times M \times Q^k$.
A *run* of $\mathcal{A}$ on an M-tree T is a mapping $r : \{1, \ldots, k\}^* \to Q$ with $r(\epsilon) \in I$ and

$$(r(\alpha), T(\alpha), r(\alpha 1), \ldots, r(\alpha k)) \in \Delta$$

for each $\alpha \in \{1, \ldots, k\}^*$. A looping automaton accepts all those M-trees for which there exists a run, i.e., the language $L(\mathcal{A})$ of M-trees accepted by $\mathcal{A}$ is

$$L(\mathcal{A}) := \{T \mid \text{there is a run of } \mathcal{A} \text{ on } T\}.$$

$\diamond$

Vardi and Wolper [1986] show that the emptiness problem for looping automata, i.e., the problem to decide whether the language $L(\mathcal{A})$ accepted by a given looping automaton $\mathcal{A}$ is empty, is decidable in polynomial time.

A Hintikka-tree T for C and $\mathcal{T}$ corresponds to a canonical model $\mathcal{I}$ of C and $\mathcal{T}$. Apart from representing the abstract domain $\Delta_\mathcal{I}$ together with the interpretation of concepts and roles, T induces a directed graph whose edges are labeled with predicates from $\{<, =\}$. Such constraint graphs describe the "concrete part" of $\mathcal{I}$, i.e., concrete successors of elements of $\Delta_\mathcal{I}$ and their relationship by concrete domain predicates.

Definition 6.5. A *constraint graph* is a pair $G = (V, E)$, where V is a countable set of *nodes* and $E \subseteq V \times V \times \{=, <\}$ a set of *edges*. We generally assume that constraint graphs are *equality closed*, i.e., that $(v_1, v_2, =) \in E$ implies $(v_2, v_1, =) \in E$. We use $\mathsf{cl}_=(E)$ to denote the equality closure of a set of edges E which is defined in the obvious way. A constraint graph $G = (V, E)$ is called *satisfiable over S*—where S is a set equipped with a total ordering $<$—iff there exists a total mapping δ from V to S such that $\delta(v_1)\, P\, \delta(v_2)$ for all $(v_1, v_2, P) \in E$. Such a mapping δ is called a *solution* for G.

A *path* Q in G is a finite non-empty sequence of nodes $v_0, \ldots, v_k \in V$ such that, for all i with $i < k$, we have $(v_i, v_{i+1}, P) \in E$ for some $P \in \{<, =\}$. Such a path is also called a path *from* v_0 to v_k. A *cycle* O in G is a path $v_0, \ldots, v_k$ with $(v_k, v_0, P) \in E$ for some $P \in \{<, =\}$. For $i \leq k$, we use i_O^+ to denote $(i+1) \bmod k+1$, i.e., i_O^+ denotes the index following i in the cycle O. The index $\cdot_O$ is omitted if clear from the context. A path $v_0, \ldots, v_k$ is a *$=$-path* iff $(v_i, v_{i+1}, =) \in E$ for $i < k$. A cycle $O = v_0, \ldots, v_k$ is a *$<$-cycle* iff $(v_i, v_{i^+}, <) \in E$ for some i with $i \leq k$. ◇

Note that constraint graphs are just a different notation for (potentially infinite) conjunctions of concrete domain predicates as used in previous sections. However, since $<$-cycles in constraint graphs play an important role in what follows, it is more convenient to use graphs rather than conjunctions.

The following theorem will be crucial for proving that, for every Hintikka-tree, there exists a corresponding canonical model. More precisely, it will be used to ensure that the constraint graph induced by a Hintikka-tree, which describes the concrete part of the corresponding model, is satisfiable.

Theorem 6.6. *A constraint graph G is satisfiable over S with $S \in \{\mathbb{Q}, \mathbb{R}\}$ iff G does not contain a $<$-cycle.*

Proof. Since the "only if" direction is trivial, we concentrate on the "if" direction. Let G be a constraint graph not containing a $<$-cycle. Let $\sim$ be the relation on V with $v_1 \sim v_2$ iff $v_1 = v_2$ or there exists a $=$-path between v_1 and v_2. Since constraint graphs are assumed to be equality closed, $\sim$ is an equivalence relation. For $v \in V$, we denote the equivalence class of v w.r.t. $\sim$ by $[v]_\sim$. Define a new constraint graph $G' = (V', E')$ as follows:

$$\begin{array}{rcl} V' & := & \{[v]_\sim \mid v \in V\} \\ E' & := & \{([v_1]_\sim, [v_2]_\sim, <) \mid \exists v_1', v_2' \in V \text{ such that} \\ & & \qquad v_1' \in [v_1]_\sim, v_2' \in [v_2]_\sim, \text{ and } (v_1', v_2', <) \in E\} \end{array}$$

Using the fact that G does not contain a $<$-cycle, it is straightforward to prove that G' does not contain a $<$-cycle. Since G' does not contain a $<$-cycle, E' induces a partial order with domain V'. By Szpilrajn's Theorem, every partial order can be extended to a total order (on the same domain) [Szpilrajn 1930]. Let $\prec_{E'}$ be a total order obtained in this way from the partial order induced by E'. In the following, we show that every total order with a countable domain can be embedded into $\mathbb{Q}$ and $\mathbb{R}$ such that the ordering is preserved. This suffices to complete the proof since it implies that that there exists a total mapping δ from V to S such that $v_1 \prec_{E'} v_2$ implies $\delta(v_1) < \delta(v_2)$. It is obvious that δ is a solution for G' and it is straightforward to use δ to construct a solution for G.

Hence, it remains to show that every total order $\prec$ with a countable domain D can be embedded into $\mathbb{Q}$ and $\mathbb{R}$ such that the ordering is preserved. Let $d_0, d_1, \ldots$ be an enumeration of D. We use induction on this enumeration to define a function δ from D to $\mathbb{Q}$ such that $d_1 \prec d_2$ implies $\delta(d_1) < \delta(d_2)$ for all $d_1, d_2 \in D$.

1. For the induction start, set $\delta(d_0)$ to some $r \in \mathbb{Q}$.

2. Assume that $\delta(d_i)$ is defined for all $i < k$. We distinguish three cases:

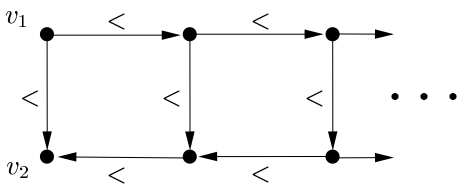

Figure 6.7: A constraint graph containing no $<$-cycle that is unsatisfiable over $\mathbb{N}$.

(a) $d_i \prec d_k$ for all $i < k$. Since $\mathbb{Q}$ has no maximum, there exists an $r \in \mathbb{Q}$ such that $r > \delta(d_i)$ for all $i < k$. Set $\delta(d_k) := r$.

(b) $d_k \prec d_i$ for all $i < k$. Since $\mathbb{Q}$ has no minimum, there exists an $r \in \mathbb{Q}$ such that $r < \delta(d_i)$ for all $i < k$. Set $\delta(d_k) := r$.

(c) Neither of the previous two cases holds. Since $\mathbb{Q}$ is dense, there exists an $r \in \mathbb{Q}$ such that

$$\max\{\delta(d_i) \mid i < k \text{ and } d_i \prec d_k\} < r < \min\{\delta(d_i) \mid i < k \text{ and } d_k \prec d_i\}.$$

Set $\delta(d_k) := r$.

It is readily checked that δ is as required. Since $\mathbb{Q} \subseteq \mathbb{R}$, δ is also an embedding of $\prec$ into $\mathbb{R}$. ❏

Note that Δ_{P} is defined as $\mathbb{R}$ and thus we will use this temporal structure in what follows. However, all obtained results also apply if we choose $\mathbb{Q}$ instead. Note that Theorem 6.6 does *not* hold if satisfiability over non-dense structures such as $\mathbb{N}$ is considered: if there exist two nodes v_1 and v_2 such that the length of $<$-paths (which are defined in analogy to $<$-cycles) between v_1 and v_2 is unbounded, then a constraint graph is unsatisfiable over $\mathbb{N}$ even if it contains no $<$-cycle. Figure 6.7 shows such a constraint graph.

Path Normal Form

Apart from the assumption that only the predicates $<$ and $=$ occur in concepts and TBoxes, we require some more normalization as a prerequisite for the satisfiability algorithm. More specifically, we assume concepts and TBoxes to be in negation normal form and, more importantly, restrict the length of concrete paths, which will turn out to be rather convenient for some constructions like defining Hintikka-trees. We start with describing NNF conversion:

Lemma 6.7 (NNF Conversion). *Exhaustive application of the following rewrite rules translates $\mathcal{ALC}(\mathsf{P})$-concepts to equivalent ones in NNF.*

$$\begin{array}{rclrcl}
\neg\neg C & \rightsquigarrow & C & & & \\
\neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D & \neg(C \sqcup D) & \rightsquigarrow & \neg C \sqcap \neg D \\
\neg(\exists R.C) & \rightsquigarrow & (\forall R.\neg C) & \neg(\forall R.C) & \rightsquigarrow & (\exists R.\neg C) \\
\neg(\exists u_1, u_2.P) & \rightsquigarrow & \exists u_1, u_2.\widetilde{P} \sqcup \exists u_2, u_1.< \sqcup\ u_1\uparrow \sqcup\ u_2\uparrow & \neg(g\uparrow) & \rightsquigarrow & \exists g, g.=
\end{array}$$

where $\tilde{\cdot}$ *denotes the exchange of predicates, i.e.,* $\tilde{<}$ *is* $=$ *and* $\tilde{=}$ *is* $<$*. By* $\mathsf{nnf}(C)$*, we denote the result of converting* C *into NNF using the above rules.*

We now introduce path normal form for $\mathcal{ALC}(\mathsf{P})$-concepts and TBoxes.

Definition 6.8 (Path Normal Form). An $\mathcal{ALC}(\mathsf{P})$-concept C is in *path normal form (PNF)* iff it is in NNF and, for all subconcepts $\exists u_1, u_2.P$ of C, we have either

1. $u_1 = g_1$ and $u_2 = g_2$ for some $g_1, g_2 \in \mathsf{N_{cF}}$,
2. $u_1 = fg_1$ and $u_2 = g_2$ for some $f \in \mathsf{N_{aF}}$ and $g_1, g_2 \in \mathsf{N_{cF}}$, or
3. $u_1 = g_1$ and $u_2 = fg_2$ for some $f \in \mathsf{N_{aF}}$ and $g_1, g_2 \in \mathsf{N_{cF}}$.

An $\mathcal{ALC}(\mathsf{P})$-TBox $\mathcal{T}$ is in path normal form iff it is in NNF and all concepts appearing in $\mathcal{T}$ are in PNF. ◇

The following lemma shows that it is not a restriction to consider only concepts and TBoxes in PNF.

Lemma 6.9. *Satisfiability of* $\mathcal{ALC}(\mathsf{P})$*-concepts w.r.t. general TBoxes can be reduced in polynomial time to satisfiability of* $\mathcal{ALC}(\mathsf{P})$*-concepts in PNF w.r.t. general TBoxes in PNF.*

Proof. Let C be an $\mathcal{ALC}(\mathsf{P})$-concept. For every concrete path $u = f_1 \cdots f_n g$ used in C, we assume that $[g], [f_n g], \ldots, [f_1 \cdots f_n g]$ are concrete features not used in C. We inductively define a mapping λ from concrete paths u in C to concepts as follows:

$$\begin{aligned} \lambda(g) &= \top \\ \lambda(fu) &= (\exists [fu], f[u].=) \sqcap \exists f.\lambda(u) \end{aligned}$$

For every $\mathcal{ALC}(\mathsf{P})$-concept C, a corresponding concept $\rho(C)$ is obtained by replacing all subconcepts $\exists u_1, u_2.P$ of C with $\exists [u_1], [u_2].P \sqcap \lambda(u_1) \sqcap \lambda(u_2)$ and $g\uparrow$ with $[g]\uparrow$. We extend the mapping ρ to TBoxes in the obvious way, i.e., if

$$\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_k \sqsubseteq D_k\},$$

then

$$\rho(\mathcal{T}) = \{\rho(C_1) \sqsubseteq \rho(D_1), \ldots, \rho(C_k) \sqsubseteq \rho(D_k)\}.$$

Now let C be an $\mathcal{ALC}(\mathsf{P})$-concept and $\mathcal{T}$ an $\mathcal{ALC}(\mathsf{P})$-TBox. Using the rewrite rules from Lemma 6.7, we can convert C into an equivalent concept C' in NNF and $\mathcal{T}$ into an equivalent TBox $\mathcal{T}'$ in NNF. It is then easy to check that C' is satisfiable w.r.t. a TBox $\mathcal{T}'$ iff $\rho(C')$ is satisfiable w.r.t. $\rho(\mathcal{T}')$. Moreover, $\rho(C')$ and $\rho(\mathcal{T}')$ are clearly in PNF and the translation can be done in polynomial time. ❏

In what follows, we generally assume that all concepts and TBoxes are in path normal form. Moreover, we will often refer to TBoxes $\mathcal{T}$ in their *concept form* $C_\mathcal{T}$ which is defined as follows:

$$C_\mathcal{T} = \bigsqcap_{C \doteq D \in \mathcal{T}} \mathsf{nnf}(C \leftrightarrow D).$$

Defining Hintikka-trees

In this section, we define Hintikka-trees for $\mathcal{ALC}(\mathsf{P})$-concepts C and TBoxes $\mathcal{T}$ (which are both required to be in PNF) and show that Hintikka-trees are proper abstractions of models, i.e., that there exists a Hintikka-tree for C and $\mathcal{T}$ iff there exists a model of C and $\mathcal{T}$.

Let C be a concept and $\mathcal{T}$ be a TBox. By $\mathsf{cl}(C, \mathcal{T})$, we denote the set of subconcepts of C and $C_\mathcal{T}$. Note that, in contrast to $\mathsf{sub}(C, \mathcal{T})$, $\mathsf{cl}(C, \mathcal{T})$ is defined in terms of the concept form $C_\mathcal{T}$ of $\mathcal{T}$. We assume that existential concepts $\exists R.D$ in $\mathsf{cl}(C, \mathcal{T})$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ are linearly ordered, and that $\mathcal{E}(C, \mathcal{T}, i)$ yields the i-th existential concept in $\mathsf{cl}(C, \mathcal{T})$ (starting with $i = 1$). Furthermore, we assume the abstract features used in $\mathsf{cl}(C, \mathcal{T})$ to be linearly ordered and use $\mathcal{F}(C, \mathcal{T}, i)$ to denote the i-th abstract feature in $\mathsf{cl}(C, \mathcal{T})$ (also starting with $i = 1$). The set of concrete features used in $\mathsf{cl}(C, \mathcal{T})$ is denoted by $\mathcal{G}(C, \mathcal{T})$.

We now define Hintikka-pairs which will be used as labels of nodes in Hintikka-trees.

Definition 6.10 (Hintikka-set, Hintikka-pair). Let C be a concept and $\mathcal{T}$ be a TBox. A set $\Psi \subseteq \mathsf{cl}(C, \mathcal{T})$ is a *Hintikka-set for* $(C, \mathcal{T})$ iff it satisfies the following conditions:

(H1) $C_\mathcal{T} \in \Psi$,

(H2) if $C_1 \sqcap C_2 \in \Psi$, then $\{C_1, C_2\} \subseteq \Psi$,

(H3) if $C_1 \sqcup C_2 \in \Psi$, then $\{C_1, C_2\} \cap \Psi \neq \emptyset$,

(H4) $\{A, \neg A\} \not\subseteq \Psi$ for all concept names $A \in \mathsf{cl}(C, \mathcal{T})$,

(H5) if $g\uparrow \in \Psi$, then $\exists u_1, u_2.P \notin \Psi$ for all concepts $\exists u_1, u_2.P$ with $u_1 = g$ or $u_2 = g$.

We say that $f \in \mathsf{N_{aF}}$ is *enforced* by a Hintikka-set Ψ iff either $\exists f.C \in \Psi$ for some concept C or $\{\exists f g_1, g_2.P, \exists g_1, f g_2.P\} \cap \Psi \neq \emptyset$ for some $g_1, g_2 \in \mathsf{N_{cF}}$ and $P \in \{<, =\}$. A *Hintikka-pair* (Ψ, χ) *for* $(C, \mathcal{T})$ consists of a Hintikka-set Ψ for $(C, \mathcal{T})$ and a set χ of tuples (g_1, g_2, P) with $g_1, g_2 \in \mathcal{G}(C, \mathcal{T})$ such that

(H6) if $(g_1, g_2, P) \in \chi$, then $\{g_1\uparrow, g_2\uparrow\} \cap \Psi = \emptyset$.

A concrete path u is *enforced* by (Ψ, χ) iff either u appears in χ or $\{\exists u, u'.P, \exists u', u.P\} \cap \Psi \neq \emptyset$ for some concrete path u' and $P \in \{<, =\}$. By $\Gamma_{(C,\mathcal{T})}$, we denote the set of all Hintikka-pairs for $(C, \mathcal{T})$. ◇

Observe that, if a concrete path u is enforced by a Hintikka-pair (Ψ, χ), then u has length 1 or 2: if u appears in χ, it has length 1 by definition; moreover, if $\{\exists u, u'.P, \exists u', u.P\} \cap \Psi \neq \emptyset$ for some u' and P, then u has length 1 or 2 since all concepts are in path normal form.

Intuitively, each node α of a (yet to be defined) Hintikka-tree T corresponds to a domain element d of the corresponding canonical model $\mathcal{I}$. The first component Ψ_α of the Hintikka-pair labeling α is the set of concepts from $\mathsf{cl}(C, \mathcal{T})$ satisfied by d. The second component χ_α states relationships between concrete successors of d. If, for example, $(g_1, g_2, <) \in \chi_\alpha$, then d must have g_1- and g_2-successors such that $g_1^{\mathcal{I}}(d) < g_2^{\mathcal{I}}(d)$. Note that the restrictions in χ_α are independent from concepts $\exists g_1, g_2.P \in \Psi_\alpha$. As will become clear when Hintikka-trees are defined, the restrictions in χ_α are used to ensure that the constraint graph induced by the Hintikka-tree T, which describes the concrete part of the model $\mathcal{I}$, does not contain a $<$-cycle, i.e., that it is satisfiable. This induced constraint graph can be thought of as the union of smaller constraint graphs, each one being described by a Hintikka-pair labeling a node in T. These pair-graphs are defined next.

Definition 6.11 (Pair-graph). Let C be a concept, $\mathcal{T}$ a TBox, and $p = (\Psi, \chi)$ a Hintikka-pair for $(C, \mathcal{T})$. The *pair-graph* $G(p) = (V, E)$ of p is a constraint graph defined as follows:

1. V is the set of concrete paths enforced by p
2. $E = \chi \cup \{(u_1, u_2, P) \mid \exists u_1, u_2.P \in \Psi\}$.

An *edge extension* of $G(p)$ is a set $E' \subseteq V \times V \times \{<, =\}$ such that for all $fg_1, fg_2 \in V$, we have $(fg_1, fg_2, <) \in E'$, $(fg_1, fg_2, =) \in E'$, or $(fg_2, fg_1, <) \in E'$. If E' is an edge extension of $G(p)$, then the graph $(V, E \cup E')$ is a *completion* of $G(p)$. $\diamond$

Observe that, since all concepts are in path normal form and since no paths of length greater one may appear in χ, we have $E' \cap E = \emptyset$ for every edge extension E' of pair-graphs (V, E). Like all constraint graphs, we assume pair-graphs to be equality closed.

We briefly comment on the connection of completions and the χ-component of Hintikka-pairs. Let α and β be nodes in a Hintikka-tree T representing domain elements d and e in the corresponding canonical model $\mathcal{I}$. Edges in Hintikka-trees represent role-relationships, i.e., if β is a successor of α in T, then there exists an $R \in \mathsf{N_R}$ such that $(d, e) \in R^{\mathcal{I}}$. Assume β is a successor of α and the edge between α and β represents relationship via the abstract feature f, i.e., we have $f^{\mathcal{I}}(d) = e$. The second component χ_β of the Hintikka-pair labeling β fixes the relationships between all concrete successors of e that "d talks about". For example, if $(\exists fg_1, g_2.=) \in \Psi_\alpha$ and $(\exists fg_3, g_2. <) \in \Psi_\alpha$, where Ψ_α is the first component of the Hintikka-pair labeling α, then "d talks about" the concrete g_1-successor and the concrete g_3-successor of e. Hence, χ_β contains $(g_1, g_3, <)$, $(g_1, g_3, =)$, or $(g_3, g_1, <)$. This is formalized by demanding that the pair-graph $G(T(\alpha))$ of the Hintikka-pair labeling α together with all the edges from the χ-components of the successors of α are a completion of $G(T(\alpha))$.

Moreover, this completion has to be satisfiable, which is necessary to ensure that the constraint graph induced by T does not contain a $<$-cycle. An appropriate way of thinking about the χ-components is as follows: at α, a completion of $G(T(\alpha))$ is "guessed". The additional edges are then "recorded" in the χ-components of the successor-nodes of α. We now define Hintikka-trees formally.

Definition 6.12 (Hintikka-tree). Let C be a concept, $\mathcal{T}$ be a TBox, k the number of existential subconcepts in $\mathsf{cl}(C,\mathcal{T})$, and ℓ be the number of abstract features in $\mathsf{cl}(C,\mathcal{T})$. A $k+\ell+1$-tuple of Hintikka-pairs $(p_0,\ldots,p_{k+\ell})$ with $p_i = (\Psi_i,\chi_i)$ and $G(p_0)=(V,E)$ is called *matching* iff

(H7) if $\exists R.D \in \Psi_0$ and $\mathcal{E}(C,\mathcal{T},i) = \exists R.D$, then $D \in \Psi_i$

(H8) if $\{\exists R.D, \forall R.E\} \subseteq \Psi_0$ and $\mathcal{E}(C,\mathcal{T},i) = \exists R.D$, then $E \in \Psi_i$

(H9) if $\exists f.D \in \Psi_0$ and $\mathcal{F}(C,\mathcal{T},i) = f$, then $D \in \Psi_{k+i}$.

(H10) if f is enforced by Ψ_0, $\mathcal{F}(C,\mathcal{T},i) = f$, and $\forall f.D \in \Psi_0$, then $D \in \Psi_{k+i}$.

(H11) the constraint graph $(V, E \cup \mathsf{cl}_=(E'))$ is a satisfiable completion of $G(p_0)$, where

$$E' = \bigcup_{1\le i\le \ell} \{(fg_1, fg_2, P) \mid \mathcal{F}(C,\mathcal{T},i) = f \text{ and } (g_1,g_2,P) \in \chi_{k+i}\}$$

A $k+\ell$-ary $\Gamma_{(C,\mathcal{T})}$-tree T is a *Hintikka-tree for* $(C,\mathcal{T})$ iff it satisfies the following conditions:

(H12) $C \in \Psi_\epsilon$, where $T(\epsilon) = (\Psi_\epsilon, \chi_\epsilon)$,

(H13) for all $\alpha \in \{1,\ldots,k+\ell\}^*$, the tuple $(T(\alpha), T(\alpha 1), \ldots, T(\alpha j))$ with $j = k+\ell$ is matching.

For a Hintikka-tree T and a node $\alpha \in \{1,\ldots,k+\ell\}^*$ with $T(\alpha) = (\Psi,\chi)$, we use $T_\lhd(\alpha)$ to denote Ψ and $T_\rhd(\alpha)$ to denote χ. Moreover, if $G(\alpha) = (V,E)$, we use $\mathsf{cpl}(T,\alpha)$ to denote the constraint graph $(V, E \cup E')$ as defined in **(H11)**. $\diamond$

Whereas most properties of Hintikka-trees deal with concepts, roles, and abstract features and are hardly surprising, **(H11)** ensures that constraint graphs induced by Hintikka-trees contain no $<$-cycle. By "guessing" a completion as explained above, possible $<$-cycles are anticipated and can be detected locally, i.e., it then suffices to check that the completions $\mathsf{cpl}(T,\alpha)$ are satisfiable as demanded by **(H11)**. Indeed, it is crucial that the cycle detection is done by a *local* condition since we need to define an automaton that accepts exactly Hintikka-trees, and automata work locally. It is worth noting that the localization of cycle detection as expressed by **(H11)** crucially depends on the path normal form.

The following two lemmas show that Hintikka-trees are appropriate abstractions of models. This result is the main step towards devising a decision procedure since, as we shall see later, defining looping automata accepting exactly Hintikka-trees is a straightforward task.

Lemma 6.13. *A concept C is satisfiable w.r.t. a general TBox $\mathcal{T}$ if there exists a Hintikka-tree for $(C,\mathcal{T})$.*

Proof. Let C be a concept, $\mathcal{T}$ a TBox, and k and ℓ as in Definition 6.12. Moreover, let T be a Hintikka-tree for $(C, \mathcal{T})$. We define an interpretation $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ as follows:

$$\begin{array}{rcl} \Delta_\mathcal{I} & = & \{1, \dots, k + \ell\}^* \\ A^\mathcal{I} & = & \{\alpha \mid A \in T_\lhd(\alpha)\} \text{ for all } A \in C_N \\ R^\mathcal{I} & = & \{(\alpha, \beta) \mid \beta = \alpha i \text{ and } \mathcal{E}(C, \mathcal{T}, i) = \exists R.E \in T_\lhd(\alpha)\} \text{ for all } R \in \mathsf{N_R} \setminus \mathsf{N_{aF}} \\ f^\mathcal{I} & = & \{(\alpha, \beta) \mid \beta = \alpha i,\ \mathcal{F}(C, \mathcal{T}, i - k) = f, \text{ and } f \text{ is enforced by } T_\lhd(\alpha)\} \\ & & \text{for all } f \in \mathsf{N_{aF}} \end{array}$$

It remains to define the interpretation of concrete features, which is done as follows: we define an (infinite) constraint graph $G(T)$ induced by T, show that $G(T)$ is satisfiable, and define the interpretation of concrete features from a solution of $G(T)$. The nodes of $G(T)$ have the form $\alpha|u$, where α is a node in T and u is a concrete path in C or $\mathcal{T}$. More precisely, $G(T)$ is defined as $(V, \mathsf{cl}_=(E))$, where

1. $V = \{\alpha|u \mid \alpha \in \{1, \dots, k + \ell\}^*, u \text{ appears in } C \text{ or } \mathcal{T}\}$
2. $E = \bigcup_{\alpha \in \{1, \dots, k+\ell\}^*} \{(\alpha|u, \alpha|u', P) \mid (u, u', P) \in \mathsf{cpl}(T, \alpha)\}$
   $\cup \{(\alpha|fg), \alpha i|g, =) \mid \mathcal{F}(C, \mathcal{T}, i - k) = f,\ fg \text{ is a node in } \mathsf{cpl}(T, \alpha)\}$

It is not hard to see that $G(T)$ really is a constraint graph, i.e., the node set of $G(T)$ is countable. Next, we show the following claim:

Claim 1: $G(T)$ is satisfiable over $\mathbb{R}$.

By Theorem 6.6, it suffices to show that $G(T)$ contains no $<$-cycle. Assume to the contrary that $G(T)$ contains a $<$-cycle and that $O = \alpha_0|u_0, \dots, \alpha_n|u_n$ is the $<$-cycle in $G(T)$ with minimal length. Fix a $t \leq n$ such that

$$\text{for each } i \text{ with } i \leq n \text{ and each } \beta \in \{1, \dots, k + \ell\}^+, \text{ we have } \alpha_i \neq \alpha_t \beta, \qquad (*)$$

i.e., there exist no α_i in O such that α_t is a true prefix of α_i (such a t exists since O is of finite length). Since O is a $<$-cycle, there exists an $s \leq n$ such that we have $(\alpha_s|u_s, \alpha_{s^+}|u_{s^+}, <) \in E$. We make a case distinction and derive a contradiction in either case.

- $\alpha_s \neq \alpha_t$. Define a sequence of nodes O' from O by deleting all nodes $\alpha_i|u_i$ with $\alpha_i = \alpha_t$. O' is non-empty since $\alpha_s \neq \alpha_t$. We show that O' is a $<$-cycle in $G(T)$, which is a contradiction to the minimality of O. Let $O' = \alpha'_0|u'_0, \dots, \alpha'_m|u'_m$. By definition of $G(T)$, the fact that $(\alpha_s|u_s, \alpha_{s^+}|u_{s^+}, <) \in E$ implies $\alpha_{s^+} = \alpha_s$. Since $\alpha_s \neq \alpha_t$, $\alpha_s|u_s$ and $\alpha_{s^+}|u_{s^+}$ are in O' and it remains to show that O' is a cycle in $G(T)$, i.e., for all $i \leq m$, we have $(\alpha'_i|u'_i, \alpha'_{i^+}|u'_{i^+}, P) \in E$ for some $P \in \{<, =\}$.

  Let $\alpha'_i|u'_i$ and $\alpha'_{i^+}|u'_{i^+}$ be nodes in O'. If these two nodes are already neighbor nodes in O, we are obviously done. Hence, assume that this is not the case. By construction of O', this implies the existence of a path

  $$\alpha'_i|u'_i, \alpha_t|u^*_1, \dots, \alpha_t|u^*_x, \alpha'_{i^+}|u'_{i^+}$$

in $G(T)$, which is at most as long as O. Since $\alpha'_i \neq \alpha_t$ and $\alpha'_{i+} \neq \alpha_t$, by construction of $G(T)$ and by $(*)$, this implies that

1. there exists a $\beta \in \{1, \ldots, k+\ell\}^*$ such that $\alpha'_i = \alpha'_{i+} = \beta$,
2. there exists an $f \in N_{aF}$ such that $\alpha_t = \beta j$ where $\mathcal{F}(C, \mathcal{T}, j-k) = f$,
3. $u'_i = fg$, $u^*_1 = g$, $u^*_x = g'$, and $u'_{i+} = fg'$ for some $g, g' \in \mathcal{G}(C, \mathcal{T})$, and
4. $(\beta|fg, \beta j|g, =) \in E$ and $(\beta|fg', \beta j|g', =) \in E$.

By definition of $G(T)$ and by Point 4, both fg and fg' are nodes in $\mathsf{cpl}(T, \beta) = (V', E')$. By definition of cpl, this implies that either

(a) $(fg', fg, <) \in E'$ or
(b) $(fg, fg', P) \in E'$ for some $P \in \{<, =\}$.

Together with Point 1 and 3 and the definition of $G(T)$, (b) obviously implies $(\alpha'_i|u'_i, \alpha'_{i+}|u'_{i+}, P) \in E$, and we are done. Moreover, in the following we show that case (a) cannot occur.

Let $\mathsf{cpl}(T, \beta j) = (V'', E'')$. In case (a), we have $(g', g, <) \in E''$: Let $G(\beta) = (V'_*, E'_*)$; by definition of pair-graphs and since all concepts are in path normal form, $(fg', fg, <) \in E'$ implies $(fg', fg, <) \in E' \setminus E'_*$; by definition of cpl and by Point 2, this means that $(g', g, <) \in T_{\triangleright}(\beta)$. Hence, $(g', g, <) \in E''$. By definition of $G(T)$ and Point 1 and 3, $(g', g, <) \in E''$ implies that $(\alpha_t|u^*_x, \alpha_t|u^*_1, <) \in E$. Hence, the path $\alpha_t|u^*_1, \ldots, \alpha_t|u^*_x$ is a $<$-cycle in $G(T)$, which contradicts the minimality of O.

- $\alpha_s = \alpha_t$. We first show that there exists a node $\alpha_z|u_z$ in O such that $\alpha_z \neq \alpha_t$. For suppose that no such node exists. Then, by definition of $G(T)$, $u_0, \ldots, u_n$ is a $<$-cycle in $\mathsf{cpl}(T, \alpha_t)$. This is clearly a contradiction to the fact that T is a Hintikka-tree. Hence, we may conclude the existence of an α_z as above. Define a sequence of nodes O' from O by deleting all nodes $\alpha_i|u_i$ with $\alpha_i \neq \alpha_t$. O' is non-empty since $\alpha_s = \alpha_t$. Moreover, O' is shorter than O due to the existence of α_z. We show that O' is a $<$-cycle in $G(T)$, which is a contradiction to the minimality of O. Let $O' = \alpha_t|u'_0, \ldots, \alpha_t|u'_m$. By definition of $G(T)$, the fact that $(\alpha_s|u_s, \alpha_{s+}|u_{s+}, <) \in E$ implies $\alpha_{s+} = \alpha_s = \alpha_t$. Hence, it remains to show that O' is a cycle in $G(T)$, i.e., that, for all $i \leq m$, we have $(\alpha_t|u'_i, \alpha_t|u'_{i+}, P) \in E$ for some $P \in \{<, =\}$.

  Let $\alpha_t|u'_i$ and $\alpha_t|u'_{i+}$ be nodes in O'. If these two nodes are already neighbor nodes in O, we are obviously done. Hence, assume that this is not the case. By construction of O', this implies the existence of a path

  $$\alpha_t|u'_i, \alpha^*_1|u^*_1, \ldots, \alpha^*_x|u^*_x, \alpha_t|u'_{i+}$$

  in $G(T)$, which is at most as long as O, such that $\alpha^*_i \neq \alpha_t$ for all i with $1 \leq i \leq x$. By construction of $G(T)$ and by $(*)$, this implies that

  1. there exists a $\beta \in \{1, \ldots, k+\ell\}^*$ such that $\alpha^*_1 = \alpha^*_x = \beta$,

2. there exists an $f \in \mathsf{N_{aF}}$ such that $\alpha_t = \beta j$ where $\mathcal{F}(C, \mathcal{T}, j-k) = f$,
3. $u'_i = g$, $u^*_1 = fg$, $u^*_x = fg'$, and $u'_{i+} = g'$ for some $g, g' \in \mathcal{G}(C, \mathcal{T})$, and
4. $(\beta|fg, \beta j|g, =) \in E$ and $(\beta|fg', \beta j|g', =) \in E$.

By definition of $G(T)$ and by Point 4, both fg and fg' are nodes in $\mathsf{cpl}(T, \beta) = (V', E')$. By definition of cpl, this implies that either

(a) $(fg', fg, <) \in E'$ or

(b) $(fg, fg', P) \in E'$ for some $P \in \{<, =\}$.

Case (a) is impossible: together with Point 1 and 3 and the definition of $G(T)$, (a) obviously implies $(\alpha^*_x|u^*_x, \alpha^*_1|u^*_1, <) \in E$. Hence, the path $\alpha^*_1|u^*_1, \ldots, \alpha^*_x|u^*_x$ is a $<$-cycle in $G(T)$ which contradicts the minimality of O.

Hence, let us assume that (b) holds. Moreover, let $\mathsf{cpl}(T, \beta j) = (V'', E'')$. We have $(g, g', P) \in E''$, which can be seen as follows: let $G(\beta) = (V'_*, E'_*)$; by definition of pair-graphs and since all concepts are in path normal form, $(fg, fg', P) \in E'$ implies $(fg, fg', P) \in E' \setminus E'_*$; by definition of cpl and by Point 2, this means that $(g, g', P) \in T_{\triangleright}(\beta)$. Hence, $(g, g', P) \in E''$. By definition of $G(T)$ and Point 1 and 3, $(g, g', P) \in E''$ implies that we have $(\alpha_t|u'_i, \alpha_t|u'_{i+}, P) \in E$, as was to be shown.

This finishes the proof of Claim 1. We may now define the interpretation of concrete features. Let δ be a solution for $G(T)$. We set

$$g^{\mathcal{I}} \;=\; \{(\alpha, x) \mid g \text{ is enforced by } T(\alpha) \text{ and } \delta(\alpha|g) = x\} \text{ for all } g \in \mathsf{N_{cF}}.$$

To show that there exists a $d \in \Delta_{\mathcal{I}}$ such that $d \in C^{\mathcal{I}}$, we prove the following claim:

Claim 2: $D \in T_{\triangleleft}(\alpha)$ implies $\alpha \in D^{\mathcal{I}}$ for all $\alpha \in \Delta_{\mathcal{I}}$ and $D \in \mathsf{cl}(C, \mathcal{T})$.

Proof: The claim is proved by induction on the structure of D. First for the induction start, which splits into several subcases:

- D is a concept name. Immediate by definition of $\mathcal{I}$.
- $D = \neg E$. Since C is in NNF and by definition of $\mathsf{cl}()$, D is in NNF. Hence, E is a concept name. By definition of $\mathcal{I}$ and since $T(\alpha)$ is a Hintikka-set and thus satisfies **(H4)**, we have $\alpha \in (\neg E)^{\mathcal{I}}$.
- $D = \exists u_1, u_2.P$. Let $G(T) = (V, E)$ and $\mathsf{cpl}(T, \alpha) = (V', E')$. By definition of pair-graphs and $\mathsf{cpl}()$, we have $(u_1, u_2, P) \in E'$. Moreover, by definition of $G(T)$, we have $(\alpha|u_1, \alpha|u_2, P) \in E$. It thus remains to show that $u_1^{\mathcal{I}}(\alpha) = \delta(\alpha|u_1)$, and $u_2^{\mathcal{I}}(\alpha) = \delta(\alpha|u_2)$: since δ is a solution for $G(T)$, this clearly implies $u_1^{\mathcal{I}}(\alpha) P u_2^{\mathcal{I}}(\alpha)$.

  First, assume $u_i = g$ for some $g \in \mathsf{N_{cF}}$. By definition of $g^{\mathcal{I}}$ and since g is enforced by $T(\alpha)$, we have $u_i^{\mathcal{I}}(\alpha) = \delta(\alpha|u_i)$ as required. Now let $u_i = fg$ with $\mathcal{F}(C, \mathcal{T}, j-k) = f$. Since fg is a node in $\mathsf{cpl}(T, \alpha)$, we have $(\alpha|fg, \alpha j|g, =) \in E$. Hence, $\delta(\alpha j|g) = \delta(\alpha|fg)$. By definition of $f^{\mathcal{I}}$ and since f is obviously enforced by $T_{\triangleleft}(\alpha)$, we have $f^{\mathcal{I}}(\alpha) = \alpha j$. By definition of cpl and of pair-graphs, $fg \in V'$

implies that g appears in $T_\triangleright(\alpha j)$: since $\mathsf{cpl}(T,\alpha)$ is both a completion of $G(\alpha)$ and satisfiable, $fg \in V'$ implies $(fg, fg, =) \in E'$; due to the definition of pair graphs and since all concepts are in path normal form, $(fg, fg, =)$ is not an edge of $G(\alpha)$; hence, by definition of cpl and since $\mathcal{F}(C,\mathcal{T},j-k) = f$, we must have $(g,g,=) \in T_\triangleright(\alpha j)$, i.e., g appears in $T_\triangleright(\alpha j)$. Since g appears in $T_\triangleright(\alpha j)$ and is thus enforced by $T(\alpha j)$, we have $g^\mathcal{I}(\alpha j) = \delta(\alpha j|g)$ by definition of $g^\mathcal{I}$. Summing up, we obtain $(fg)^\mathcal{I}(\alpha) = \delta(\alpha j|g) = \delta(\alpha|fg)$.

- $D = g\uparrow$. If $g^\mathcal{I}(\alpha)$ is defined, then g is enforced by $T(\alpha)$. We show that this implies $g\uparrow \notin T_\triangleleft(\alpha)$. If g is enforced by $T(\alpha)$, then either (i) g appears in $T_\triangleright(\alpha)$ or (ii) $\{\exists g, u'.P, \exists u', g.P\} \cap T_\triangleleft(\alpha) \neq \emptyset$ for some concrete path u' and $P \in \{<, =\}$. In case (i), **(H6)** yields $g\uparrow \notin T_\triangleleft(\alpha)$. In case (ii), **(H5)** yields the same result.

For the induction step, we make a case distinction according to the topmost operator in D. Assume $D \in T_\triangleleft(\alpha)$.

- $D = C_1 \sqcap C_2$ or $D = C_1 \sqcup C_2$. Straightforward by **(H2)** and **(H3)** of Hintikka-sets and by induction hypothesis.

- $D = \exists R.E$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. By definition of $R^\mathcal{I}$, we have $(\alpha,\beta) \in R^\mathcal{I}$ for $\beta = \alpha i$ and $\mathcal{E}(C,\mathcal{T},i) = \exists R.E$. By **(H7)**, we have $E \in T_\triangleleft(\beta)$, and, by induction, $\beta \in E^\mathcal{I}$.

- $D = \exists f.E$ with $f \in \mathsf{N_{aF}}$. Hence, f is enforced by $T_\triangleleft(\alpha)$. By definition of $f^\mathcal{I}$, we have $f^\mathcal{I}(\alpha) = \beta$ for $\beta = \alpha i$ and $\mathcal{F}(C,\mathcal{T},i-k) = f$. By **(H9)**, we have $E \in T_\triangleleft(\beta)$, and, by induction, $\beta \in E^\mathcal{I}$.

- $D = \forall R.E$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. Let $(\alpha,\beta) \in R^\mathcal{I}$. By definition of $R^\mathcal{I}$, there exists an i such that $\mathcal{E}(C,\mathcal{T},i) = \exists R.D \in T_\triangleleft(\alpha)$ and $\beta = \alpha i$. By **(H8)**, we have $E \in T_\triangleleft(\beta)$, and, by induction, $\beta \in E^\mathcal{I}$. Since this holds independently of the choice of β, we have $\alpha \in (\forall R.E)^\mathcal{I}$.

- $D = \forall f.E$ with $f \in \mathsf{N_{aF}}$. Let $f^\mathcal{I}(\alpha) = \beta$. By definition of $f^\mathcal{I}$, we have $\beta = \alpha i$, $\mathcal{F}(C,\mathcal{T},i-k) = f$, and f is enforced by $T_\triangleleft(\alpha)$. By **(H10)**, we have $E \in T_\triangleleft(\beta)$, and, by induction, $\beta \in E^\mathcal{I}$.

This completes the proof of the claim. Since $C \in T_\triangleleft(\epsilon)$ by **(H12)** and, for all $\alpha \in \Delta_\mathcal{I}$, we have $C_\mathcal{T} \in T_\triangleleft(\alpha)$ by **(H1)**, it is an immediate consequence of the semantics of TBoxes and Claim 2 that $\mathcal{I}$ is a model of C w.r.t. $\mathcal{T}$. ❑

Lemma 6.14. *A concept C is satisfiable w.r.t. a general TBox $\mathcal{T}$ only if there exists a Hintikka-tree for $(C,\mathcal{T})$.*

Proof. Let C be a concept, $\mathcal{T}$ a TBox, and k and ℓ as in Definition 6.12. Moreover, let $\mathcal{I}$ be a model of C w.r.t. $\mathcal{T}$, i.e., there exists a $d_0 \in \Delta_\mathcal{I}$ such that $d_0 \in C^\mathcal{I}$ and $D^\mathcal{I} = E^\mathcal{I}$ for all $D \doteq E \in \mathcal{T}$. We inductively define a Hintikka-tree T for $(C, \mathcal{T})$, i.e., a $k+\ell$-ary $\Gamma_{(C,\mathcal{T})}$-tree that satisfies **(H12)** and **(H13)**. Along with T, we define a mapping τ from $\{1, \ldots, k+\ell\}^*$ to $\Delta_\mathcal{I}$ in such a way that

$$T_\triangleleft(\alpha) = \{D \in \mathsf{cl}(C, \mathcal{T}) \mid \tau(\alpha) \in D^\mathcal{I}\} \qquad (*)$$

For the induction start, set

$$\tau(\epsilon) := d_0, \;\; T_\triangleleft(\epsilon) := \{D \in \mathsf{cl}(C, \mathcal{T}) \mid d_0 \in D^\mathcal{I}\}, \;\; \text{and} \;\; T_\triangleright(\epsilon) := \emptyset.$$

Obviously, $(*)$ is satisfied. Now for the induction step. Let $\alpha \in \{1, \ldots, k+\ell\}^*$ be a word of minimal length such that $\tau(\alpha)$ is defined and $\tau(\alpha i)$ is undefined for some $i \in \{1, \ldots, k+\ell\}$. We make a case distinction as follows:

1. $\mathcal{E}(C, \mathcal{T}, i) = \exists R.D \in T_\triangleleft(\alpha)$. By $(*)$, we have $\tau(\alpha) \in (\exists R.D)^\mathcal{I}$. Thus, there exists some $e \in \Delta_\mathcal{I}$ such that $(\tau(\alpha), e) \in R^\mathcal{I}$ and $e \in D^\mathcal{I}$. Set $\tau(\alpha i) := e$, $T_\triangleleft(\alpha i) := \{E \in \mathsf{cl}(C, \mathcal{T}) \mid e \in E^\mathcal{I}\}$, and $T_\triangleright(\alpha i) := \emptyset$.

2. $\mathcal{F}(C, \mathcal{T}, i-k) = f$, and f is enforced by $\tau(\alpha)$. By $(*)$ and the definition of "enforced", there exists an $e \in \Delta_\mathcal{I}$ such that $f^\mathcal{I}(\tau(\alpha)) = e$. Set $\tau(\alpha i) := e$, $T_\triangleleft(\alpha i) := \{E \in \mathsf{cl}(C, \mathcal{T}) \mid e \in E^\mathcal{I}\}$, and

$$T_\triangleright(\alpha i) := \{(g_1, g_2, P) \mid fg_1 \text{ and } fg_2 \text{ are enforced by } T(\alpha) \text{ and } g_1^\mathcal{I}(e) P g_2^\mathcal{I}(e)\}$$

3. α, i do not match the above cases. Then set $\tau(\alpha i) := \tau(\epsilon)$ and $T(\alpha i) := T(\epsilon)$.

Clearly, $(*)$ is satisfied after each induction step, and hence T is well-defined. Intuitively, Case 3 applies if the i-th successor of α is not needed to satisfy the Properties of Hintikka-trees. In this case, the choice of $\tau(\alpha i)$ is arbitrary: we could have defined $\tau(\alpha i)$ as any element of $\Delta_\mathcal{I}$ (instead of choosing $\tau(\epsilon)$).

We must show that T is a Hintikka-tree for $(C, \mathcal{T})$. From $(*)$ together with the semantics of concepts and TBoxes, it is clear that $T_\triangleleft(\alpha)$ is a Hintikka-set for $(C, \mathcal{T})$ for each $\alpha \in \{1, \ldots, k+\ell\}^*$. Let us show exemplarily that **(H1)** holds. Assume to the contrary that there exists an $\alpha \in \{1, \ldots, k+\ell\}^*$ such that $C_\mathcal{T} \notin T_\triangleleft(\alpha)$. Since $C_\mathcal{T} \in \mathsf{cl}(C, \mathcal{T})$ and by $(*)$, we have $\tau(\alpha) \notin C_\mathcal{T}^\mathcal{I}$ and thus $\tau(\alpha) \in (\neg C_\mathcal{T})^\mathcal{I}$. By definition of $C_\mathcal{T}$, this implies the existence of $D \doteq E \in \mathcal{T}$ such that $\tau(\alpha) \in (\neg \mathsf{nnf}(D \leftrightarrow E))^\mathcal{I}$, i.e., $\tau(\alpha) \in D^\mathcal{I} \setminus E^\mathcal{I}$ or $\tau(\alpha) \in E^\mathcal{I} \setminus D^\mathcal{I}$. Hence, we do not have $D^\mathcal{I} = E^\mathcal{I}$ and obtain a contradiction to the fact that $\mathcal{I}$ is a model of $\mathcal{T}$.

Now we show that $T(\alpha)$ is a Hintikka-pair for each node α, i.e., that **(H6)** is satisfied. The proof is by contradiction. Assume that there exists an $\alpha \in \{1, \ldots, k+\ell\}^*$ such that $(g_1, g_2, P) \in T_\triangleright(\alpha)$ and $g_j\!\uparrow \in T_\triangleleft(\alpha)$ for $j \in \{1, 2\}$. By definition of $T_\triangleright$, $(g_1, g_2, P) \in T_\triangleright(\alpha)$ implies that $g_j^\mathcal{I}(\tau(\alpha))$ is defined. But from $g_j\!\uparrow \in T_\triangleleft(\alpha)$ and $(*)$, we obtain that $g_j^\mathcal{I}(\tau(\alpha))$ is undefined: contradiction.

It remains to show that T satisfies **(H12)** and **(H13)**, where the latter amounts to showing that, for each $\alpha \in \{1, \ldots, k+\ell\}^*$, the tuple $(T(\alpha), T(\alpha 1), \ldots, T(\alpha j))$ with $j = k+\ell$ satisfies **(H7)** to **(H11)**.

(H7) Let $\exists R.D \in T_{\lhd}(\alpha)$ and $\mathcal{E}(C,\mathcal{T},i) = \exists R.D$. By definition of τ (Case 1), we have $\tau(\alpha i) = e$ for some $e \in \Delta_{\mathcal{I}}$ with $(\tau(\alpha), e) \in R^{\mathcal{I}}$ and $e \in D^{\mathcal{I}}$. By $(*)$, we thus have $D \in T_{\lhd}(\alpha i)$.

(H8) Let $\{\exists R.D, \forall R.E\} \subseteq T_{\lhd}(\alpha)$ and $\mathcal{E}(C,\mathcal{T},i) = \exists R.D$. By definition of τ (Case 1), we have $\tau(\alpha i) = e$ for some $e \in \Delta_{\mathcal{I}}$ with $(\tau(\alpha), e) \in R^{\mathcal{I}}$. By $(*)$, we have $\tau(\alpha) \in (\forall R.E)^{\mathcal{I}}$ which implies $e \in E^{\mathcal{I}}$. By $(*)$, we thus have $E \in T_{\lhd}(\alpha i)$.

(H9) Let $\exists f.D \in T_{\lhd}(\alpha)$ and $\mathcal{F}(C,\mathcal{T},i) = f$. Hence, f is enforced by $T(\alpha)$. By definition of τ (Case 2), we have $\tau(\alpha j) = e$ for $e = f^{\mathcal{I}}(\tau(\alpha))$ and $j = k + i$. From $\exists f.D \in T_{\lhd}(\alpha)$ and $(*)$, we obtain $\tau(\alpha) \in (\exists f.D)^{\mathcal{I}}$ and thus $e \in D^{\mathcal{I}}$. Again by $(*)$, we get $D \in T_{\lhd}(\alpha j)$.

(H10) Let f be enforced by $T(\alpha)$, $\mathcal{F}(C,\mathcal{T},i) = f$, and $\forall f.D \in T_{\lhd}(\alpha)$. By definition of τ (Case 2), we have $\tau(\alpha j) = e$ for $e = f^{\mathcal{I}}(\tau(\alpha))$ and $j = k + i$. From $\forall f.D \in T_{\lhd}(\alpha)$ and $(*)$, we obtain $\tau(\alpha) \in (\forall f.D)^{\mathcal{I}}$ and thus $e \in D^{\mathcal{I}}$. Again by $(*)$, we get $D \in T_{\lhd}(\alpha j)$.

(H11) Let $G(T(\alpha)) = (V, E)$ and E' be defined as in **(H11)**. To prove that **(H11)** is satisfied, we show that

1. E' is an edge extension of $G(T(\alpha))$, which implies that $(V, E \cup E')$ is a completion of $G(T(\alpha))$ and
2. $(V, E \cup E')$ is satisfiable.

We first prove Point 1. It needs to be shown that, for each $fg_1, fg_2 \in V$, $\{(fg_1, fg_2, <), (fg_1, fg_2, =), (fg_2, fg_1, <)\} \cap E' \neq \emptyset$. By definition of $G(T(\alpha))$, fg_1 and fg_2 are enforced by $T(\alpha)$. Since $T_{\rhd}(\alpha)$ may only contain concrete paths of length 1, we have $\{\exists fg_1, u.P', \exists u, fg_1.P'\} \cap T_{\lhd}(\alpha) \neq \emptyset$ for some concrete path u and $P' \in \{<, =\}$ and similarly for fg_2. By $(*)$, this implies that $f^{\mathcal{I}}(g_1^{\mathcal{I}}(\tau(\alpha)))$ and $f^{\mathcal{I}}(g_2^{\mathcal{I}}(\tau(\alpha)))$ are defined. By definition of T (Case 2) and since f is obviously enforced by $T_{\lhd}(\alpha)$, we have $f^{\mathcal{I}}(\tau(\alpha)) = \tau(\alpha i)$ with $\mathcal{F}(C,\mathcal{T}, i-k) = f$. Hence, $g_1^{\mathcal{I}}(\tau(\alpha i))$ and $g_2^{\mathcal{I}}(\tau(\alpha i))$ are defined. By the semantics, we have $g_1^{\mathcal{I}}(\tau(\alpha i)) < g_2^{\mathcal{I}}(\tau(\alpha i))$, $g_1^{\mathcal{I}}(\tau(\alpha i)) = g_2^{\mathcal{I}}(\tau(\alpha i))$, or $g_2^{\mathcal{I}}(\tau(\alpha i)) < g_1^{\mathcal{I}}(\tau(\alpha i))$. By definition of $T_{\rhd}$, this implies $\{(g_1, g_2, <), (g_1, g_2, =), (g_2, g_1, <)\} \cap T_{\rhd}(\alpha i) \neq \emptyset$. Hence, by definition of E', we have $\{(fg_1, fg_2, <), (fg_1, fg_2, =), (fg_2, fg_1, <)\} \cap E' \neq \emptyset$.

We now prove Point 2 from above. Define a mapping δ from V to $\mathbb{R}$ as follows: $\delta(u) := u^{\mathcal{I}}(\tau(\alpha))$. This mapping is well-defined, which can be seen as follows. Fix a $u \in V$. Since u is enforced by $T(\alpha)$, either

(i) u occurs in $T_{\rhd}(\alpha)$ or

(ii) $\{\exists u, u'.P, \exists u', u.P\} \cap T_{\lhd}(\alpha) \neq \emptyset$ for some concrete path u' and $P \in \{<, =\}$.

In Case (i), we have $u = g$ for some $g \in \mathsf{N_{cF}}$. By definition of T, there exists a predecessor β of α in T such that $\alpha = \beta i$, $\mathcal{F}(C,\mathcal{T}, i-k) = f$ for some $f \in \mathsf{N_{aF}}$, and fg is enforced by $T(\beta)$. Since $T_{\rhd}(\beta)$ contains only concrete paths of length 1, we have $\{\exists fg, u.P, \exists u, fg.P\} \cap T_{\lhd}(\beta) \neq \emptyset$ for some concrete path u

and $P \in \{<,=\}$. By $(*)$, $g^{\mathcal{I}}(f^{\mathcal{I}}(\tau(\beta)))$ is defined. Since, by definition of T, we have $f^{\mathcal{I}}(\tau(\beta))) = \tau(\alpha)$, $g^{\mathcal{I}}(\tau(\alpha))$ is defined. In Case (ii), it follows from $(*)$ that $u^{\mathcal{I}}(\tau(\alpha))$ is defined.

We show that δ is a solution for $(V, E \cup E')$ by distinguishing the following cases:

1. $(u_1, u_2, P) \in E$ and $(u_1, u_2, P) \in T_{\triangleright}(\alpha)$. Then there exist $g_1, g_2 \in \mathsf{N_{cF}}$ such that $u_1 = g_1$ and $u_2 = g_2$. By definition of $T_{\triangleright}$, we have $g_1^{\mathcal{I}}(\tau(\alpha))Pg_2^{\mathcal{I}}(\tau(\alpha))$, and thus, by definition of δ, $\delta(g_1)P\delta(g_2)$.
2. $(u_1, u_2, P) \in E$ and $\exists u_1, u_2.P \in T_{\triangleleft}(\alpha)$. By $(*)$, we have $\tau(\alpha) \in (\exists u_1, u_2.P)^{\mathcal{I}}$. Hence, $u_1^{\mathcal{I}}(\tau(\alpha))Pu_2^{\mathcal{I}}(\tau(\alpha))$. By definition of δ, we thus obtain $\delta(u_1)P\delta(u_2)$.
3. $(u_1, u_2, P) \in E'$. By definition of E', we have $u_1 = fg_1$, $u_2 = fg_2$, and $(g_1, g_2, P) \in T_{\triangleright}(\alpha i)$ where $g_1, g_2 \in \mathsf{N_{cF}}$ and $\mathcal{F}(C, \mathcal{T}, k - i) = f$. By definition of $T_{\triangleright}$, this implies that fg_1 and fg_2 are enforced by $T(\alpha)$ and that $g_1^{\mathcal{I}}(\tau(\alpha i))Pg_2^{\mathcal{I}}(\tau(\alpha i))$. From this and the definition of T (Case 2), it follows that $f^{\mathcal{I}}(\tau(\alpha)) = \tau(\alpha i)$. We conclude $\delta(u_1)P\delta(u_2)$.

To sum up, we have shown that **(H13)** holds. **(H12)** is satisfied by definition of T (induction start) and since $d_0 \in C^{\mathcal{I}}$. ❏

Defining Looping Automata

To prove decidability, it remains to define a looping automaton $\mathcal{A}_{(C,\mathcal{T})}$ for each concept C and TBox $\mathcal{T}$ such that $\mathcal{A}_{(C,\mathcal{T})}$ accepts exactly the Hintikka-trees for $(C, \mathcal{T})$. Using the notion of matching tuples of Hintikka-pairs from Definition 6.12, this is rather straightforward.

Definition 6.15. Let C be a concept, $\mathcal{T}$ be a TBox, k the number of existential subconcepts in $\mathsf{cl}(C, \mathcal{T})$, and ℓ be the number of abstract features in $\mathsf{cl}(C, \mathcal{T})$. The looping automaton $\mathcal{A}_{(C,\mathcal{T})} = (Q, M, \Delta, I)$ is defined as follows:

- $Q := M := \Gamma_{(C,\mathcal{T})}$
- $I := \{(\Psi, \chi) \in Q \mid C \in \Psi\}$.
- $((\Psi, \chi), (\Psi', \chi'), (\Psi_1, \chi_1), \ldots, (\Psi_{k+\ell}, \chi_{k+\ell})) \in \Delta$ iff

  $(\Psi, \chi) = (\Psi', \chi')$ and

  $((\Psi, \chi), (\Psi_1, \chi_1), \ldots, (\Psi_{k+\ell}, \chi_{k+\ell}))$ is matching.

◇

As a consequence of the following lemma and Lemmas 6.13 and 6.14, we can reduce satisfiability of concepts w.r.t. general TBoxes (both in PNF) to the emptiness of the language accepted by looping automata.

Lemma 6.16. *T is a Hintikka-tree for $(C, \mathcal{T})$ iff $T \in L(\mathcal{A}_{(C,\mathcal{T})})$.*

Proof. Let C be a concept, $\mathcal{T}$ a TBox, and k, ℓ, and $\mathcal{A}_{(C,\mathcal{T})}$ as in Definition 6.15.

For the "if" direction, let r be a run of $\mathcal{A}_{(C,\mathcal{T})}$ on T. By definition of runs and of Δ, we have

$$r(\alpha) = T(\alpha) \text{ for all } \alpha \in \{1, \dots, k + \ell\}^*.$$

Hence, it remains to be shown that r is a Hintikka-tree for $(C, \mathcal{T})$, which is straightforward: (i) by definition of Q, r is a $\Gamma_{(C,\mathcal{T})}$-tree; (ii) since, by definition of runs, $r(\epsilon) \in I$, **(H12)** is satisfied; and (iii) by definition of runs and of Δ, **(H13)** is satisfied.

Now for the "only if" direction. It is straightforward to check that the function r defined by $r(\alpha) := T(\alpha)$ is a run of $\mathcal{A}_{C,\mathcal{T}}$ on T: (i) by definition of Hintikka-trees and $\mathcal{A}_{C,\mathcal{T}}$, $r(\alpha) \in Q$ for all $\alpha \in \{1, \dots, k + \ell\}^*$; (ii) by **(H12)** and definition of I, we have $r(\epsilon) \in I$; (iii) by **(H13)** and by definition of r and of Δ, we have $(r(\alpha), T(\alpha), r(\alpha_1), \dots, r(\alpha_k)) \in \Delta$ for all $\alpha \in \{1, \dots, k + \ell\}^*$. ❏

It is an immediate consequence of Lemmas 6.9, 6.13, 6.14, and 6.16 and the decidability of the emptiness problem of looping automata [Vardi & Wolper 1986] that satisfiability of $\mathcal{ALC}(\mathsf{P})$-concepts w.r.t. general TBoxes is decidable. However, the presented automata-based algorithm has the nice property of additionally providing us with a tight complexity bound.

Theorem 6.17. *Satisfiability of $\mathcal{ALC}(\mathsf{P})$-concepts w.r.t. general TBoxes is* ExpTime-*complete.*

Proof. The lower bound is an immediate consequence of the fact that $\mathcal{ALC}$ with general TBoxes is ExpTime-hard (Theorem 2.6). For the upper bound, we need to show that the size of $\mathcal{A}_{(C,\mathcal{T})}$ is exponential in $|C| + |\mathcal{T}|$, which clearly implies that $\mathcal{A}_{(C,\mathcal{T})}$ can be computed in exponential time. Indeed, if this is established, we can use Lemmas 6.9, 6.13, 6.14, and 6.16 together with the fact that the emptiness problem for looping automata $\mathcal{A}_{(C,\mathcal{T})}$ is in PTime [Vardi & Wolper 1986] to conclude that satisfiability of $\mathcal{ALC}(\mathsf{P})$-concepts w.r.t. TBoxes can be decided in deterministic exponential time. Hence, let us investigate the size of $\mathcal{A}_{(C,\mathcal{T})} = (Q, M, \Delta, I)$. Obviously, the cardinality of $\mathsf{cl}(C, \mathcal{T})$ is linear in $|C| + |\mathcal{T}|$. Hence, by definition of $\mathcal{A}_{(C,\mathcal{T})}$ and Hintikka-pairs, the cardinality of Q and M are exponential in $|C| + |\mathcal{T}|$. Again by definition of $\mathcal{A}_{(C,\mathcal{T})}$, this implies that the cardinalities of I and Δ are also exponential in $|C| + |\mathcal{T}|$. Hence, the size of $\mathcal{A}_{(C,\mathcal{T})}$ is exponential in $|C| + |\mathcal{T}|$. ❏

Since subsumption can be reduced to (un)satisfiability, $\mathcal{ALC}(\mathsf{P})$-concept subsumption is also ExpTime-complete. It is interesting to note that the result just established does not contradict Corollary 5.18 since the concrete domain P is not arithmetic. Using the translation of $\mathcal{ALC}(\mathtt{I})$-concepts to $\mathcal{ALC}(\mathsf{P})$-concepts from Section 2.4.3 and extending it to TBoxes in the natural way, we obtain the following corollary.

Corollary 6.18. *Satisfiability of $\mathcal{ALC}(\mathtt{I})$-concepts w.r.t. general TBoxes is* ExpTime-*complete.*

6.2.4 Deciding ABox Consistency

In this section, we extend the ExpTime upper bound just obtained to $\mathcal{ALC}(\mathsf{P})$-ABox consistency w.r.t. general TBoxes. The extended upper bound is established using a precompletion algorithm similar to the ones devised in Sections 3.2 and 4.1.3.

As in the previous subsection, we assume w.l.o.g. that all concepts (also inside TBoxes and ABoxes) contain only the predicates $<$ and $=$. Moreover, we require TBoxes and ABoxes to be in path normal form, where an ABox $\mathcal{A}$ is in PNF iff every concept occurring in $\mathcal{A}$ is in PNF. The next lemma shows that this assumption does not sacrifice generality.

Lemma 6.19. *Consistency of $\mathcal{ALC}(\mathsf{P})$-ABoxes w.r.t. general TBoxes can be reduced to consistency of $\mathcal{ALC}(\mathsf{P})$-ABoxes in PNF w.r.t. general TBoxes in PNF.*

Proof. Let $\mathcal{A}$ be an ABox and $\mathcal{T}$ a TBox, and let k be the length of the longest concrete path occurring in $\mathcal{A}$ or $\mathcal{T}$. For every concrete path $u = f_1 \cdots f_n g$ used in $\mathcal{A}$ or $\mathcal{T}$, we assume that $[g], [f_n g], \dots, [f_1 \cdots f_n g]$ are concrete features not appearing in $\mathcal{A}$ or $\mathcal{T}$. Let ρ be the mapping from concepts to concepts in PNF and from TBoxes to TBoxes in PNF introduced in the proof of Lemma 6.9. Construct an ABox $\rho(\mathcal{A})$ from $\mathcal{A}$ by performing the following steps:

1. Replace every concept C in $\mathcal{A}$ with $\rho(C)$;

2. Replace every assertion $(a, x) : g \in \mathcal{A}$ with $(a, x) : [g]$;

3. For $i = 1, \dots, k-1$ do the following: for every pair of assertions

$$(a, b) : f,\ (b, x) : [u] \in \mathcal{A}$$

   where the length of u is i and fu is a postfix of a concrete path occurring in $\mathcal{A}$ or $\mathcal{T}$, add $(a, x) : [fu]$ to $\mathcal{A}$.

It is straightforward to prove that $\mathcal{A}$ is satisfiable w.r.t. $\mathcal{T}$ iff $\rho(\mathcal{A})$ is satisfiable w.r.t. $\rho(\mathcal{T})$. Moreover, the size of $\rho(\mathcal{A})$ and $\rho(\mathcal{T})$ is polynomial in $n = |\mathcal{A}| + |\mathcal{T}|$ and $\rho(\mathcal{A})$ and $\rho(\mathcal{T})$ can be constructed in polynomial time. For $\rho(\mathcal{A})$, this can be seen as follows. Since the number of postfixes of concrete paths occurring in $\mathcal{A}$ and $\mathcal{T}$ is clearly bounded by n, the number of abstract and concrete objects in $\mathcal{A}$ is also bounded by n, and no new abstract objects are introduced, the number of assertions of the form $(a, x) : [u]$ generated in Step 3 is bounded by n^3. Since the number of assertions $(a, b) : f$ is obviously bounded by n, the number of pairs to be considered in each step of the "for" loop in Step 3 is thus bounded by n^4. Since we clearly have $k \leq n$, $\rho(\mathcal{A})$ can be computed in time n^5. In addition, $\rho(\mathcal{T})$ was treated in the proof of Lemma 6.9. ❑

R$\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{A}(a)$ then $\mathcal{A} := \mathcal{A} \cup \{a : C_1, a : C_2\}$
R$\sqcup$	if $C_1 \sqcup C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \cap \mathcal{A}(a) = \emptyset$ then $\mathcal{A}_1 := \mathcal{A} \cup \{a : C_1\}$ and $\mathcal{A}_2 := \mathcal{A} \cup \{a : C_2\}$
R$\exists$f	if $\exists f.C \in \mathcal{A}(a)$, b is an f-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$
R$\forall$	if $\forall R.C \in \mathcal{A}(a)$, b is an R-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$
Rc1	if $\exists g_1, g_2.P \in \mathcal{A}(a)$, x_i is a g_i-successor of a in $\mathcal{A}$ for $i \in \{1, 2\}$, and $(x_1, x_2) : P \notin \mathcal{A}$ then set $\mathcal{A} := \mathcal{A} \cup \{(x_1, x_2) : P\}$
Rc2	if $\exists f g_1, g_2.P \in \mathcal{A}(a)$, b is an f-successor of a in $\mathcal{A}$, and there are no $x_1, x_2 \in \mathsf{O_c}$ such that - x_1 is a g_1-successor of a, - x_2 is a g_2-successor of b, and - $(x_1, x_2) : P \in \mathcal{A}$ then set $\mathcal{A} := \mathcal{A} \cup \{(a, x_1) : g_1, (b, x_2) : g_2, (x_1, x_2) : P\}$ where x_1 and x_2 are fresh in $\mathcal{A}$
Rc3	Symmetric to Rc2 but for concepts $\exists g_1, f g_2.P \in \mathcal{A}(a)$
Rch	if x_i is a g_i-successor of a in $\mathcal{A}$ for $i \in \{1, 2\}$ and $\{\exists g_1, g_2.<,\ \exists g_1, g_2.=,\ \exists g_2, g_1.<\} \cap \mathcal{A} = \emptyset$ then set $\mathcal{A}_1 := \mathcal{A} \cup \{a : \exists g_1, g_2.<\}$, $\mathcal{A}_2 := \mathcal{A} \cup \{a : \exists g_1, g_2.=\}$, and $\mathcal{A}_3 := \mathcal{A} \cup \{a : \exists g_2, g_1.<\}$.
R$\doteq$	if $C_\mathcal{T} \notin \mathcal{A}(a)$ then set $\mathcal{A} := \mathcal{A} \cup \{a : C_\mathcal{T}\}$
Rfe	if $\{(a, b) : f, (a, c) : f\} \subseteq \mathcal{A}$ and $b \neq c$ (resp. $\{(a, x) : g, (a, y) : g\} \subseteq \mathcal{A}$ and $x \neq y$) then replace b by c in $\mathcal{A}$ (resp. x by y)

Figure 6.8: Precompletion rules for $\mathcal{ALC}(\mathsf{P})$.

The precompletion algorithm repeatedly applies precompletion rules in search of an ABox to which no more rules are applicable and which does not contain obvious contradictions. If such an ABox is found, the precompletion algorithm constructs a number of reduction concepts that are passed to the algorithm from the previous section for satisfiability checking. A major difference to the precompletion algorithms devised in previous sections is that, here, we are heading for a deterministic time bound. Hence, in the case of branching rules (i.e., rules with more than one possible outcome), we cannot just non-deterministically "guess" an outcome, but we must (deterministically) consider all possible outcomes. Intuitively, this means that the precompletion algorithm computes *all* precompletions of the input ABox instead of guessing a single one.

The precompletion rules can be found in Figure 6.8. In the formulation of the rules, we use the notion "successor" introduced in Definition 3.2. As in previous sections, we call an object $x \in \mathsf{O_c}$ *fresh* in an ABox $\mathcal{A}$ iff x does not occur in $\mathcal{A}$. The determinism of branching rules is reflected in the definition of the rules $\mathsf{R}\sqcup$ and Rch, whose application results in the construction of more than one ABox. Let us comment on the rules in more detail:

- Apart from being deterministic, the rules $\mathsf{R}\sqcap$, $\mathsf{R}\sqcup$, $\mathsf{R}\forall$, and Rfe are well-known from, e.g., Figures 3.2 and 4.1.
- The $\mathsf{R}\exists\mathsf{f}$ rule resembles its counterpart from Figure 3.2, but differs in that it does not generate new objects.
- The $\mathsf{Rc1}$, $\mathsf{Rc2}$, and $\mathsf{Rc3}$ rules are all specializations of the Rc rule in Figure 3.2. There exists one rule for each syntactic form allowed for $\exists u_1, u_2.P$ concepts in PNF. The main difference to the Rc rule from Figure 3.2 is that no new abstract objects are generated. Note that $\mathsf{Rc1}$ does not generate new concrete objects while $\mathsf{Rc2}$ and $\mathsf{Rc3}$ do. Intuitively, if (i) $\exists g_1, g_2.P \in \mathcal{A}(a)$ and a has both g_1- and g_2-successor or (ii) $\{\exists fg_1, g_2.P, \exists g_1, fg_2.P\} \cap \mathcal{A}(a) \neq \emptyset$ and a has a b-successor, then we must "treat" the mentioned concepts in the precompletion phase of the algorithm to avoid losing information. In all other cases (e.g. $\exists g_1, g_2.P \in \mathcal{A}(a)$ and a has no g_1-successor) it suffices to "treat" these concepts as part of the reduction concept constructed for a.
- The Rch rule has the character of a "choose rule" (c.f. Section 5.4) and is needed to ensure that the relation between any two concrete successors of an abstract object a is recorded as a concept of the form $\exists g_1, g_2.P$ in the node label of a. This is necessary since the relation between such concrete successors must be passed to the satisfiability algorithm as part of the reduction concept.
- The $\mathsf{R}\doteq$ rule deals with general TBoxes, where the concept form $C_\mathcal{T}$ of a TBox $\mathcal{T}$ is defined as in Section 6.2.3.

If an ABox $\mathcal{A}'$ can be obtained from an ABox $\mathcal{A}$ by exhaustive rule application using a TBox $\mathcal{T}$, then $\mathcal{A}'$ is called *precomplete* and a *precompletion* of $\mathcal{A}$ w.r.t. $\mathcal{T}$. Contradictory ABoxes are formalized as follows:

Definition 6.20 (Clash). Let $\mathcal{A}$ be an ABox. We use $\zeta_\mathcal{A}$ to denote the predicate conjunction associated with $\mathcal{A}$ as introduced in Definition 3.4. $\mathcal{A}$ is called *concrete domain satisfiable* iff $\zeta_\mathcal{A}$ is satisfiable. $\mathcal{A}$ is said to contain a *clash* iff one of the following conditions applies:

1. $\{A, \neg A\} \subseteq \mathcal{A}(a)$ for a concept name A and object $a \in \mathsf{O_a}$,
2. $g{\uparrow} \in \mathcal{A}(a)$ for some $a \in \mathsf{O_a}$ and there exists a g-successor x of a, or

define procedure $\mathsf{cons}(\mathcal{A},\mathcal{T})$
  while a completion rule $\mathsf{R} \in \{\mathsf{R}\sqcap, \mathsf{R}\exists\mathsf{f}, \mathsf{R}\forall, \mathsf{Rc1}, \mathsf{Rc2}, \mathsf{Rc3}, \mathsf{R}\doteq, \mathsf{Rfe}\}$
        is applicable to $\mathcal{A}$ **do**
    apply R to $\mathcal{A}$
  if a completion rule $\mathsf{R} \in \{\mathsf{R}\sqcup, \mathsf{Rch}\}$ is applicable to $\mathcal{A}$ **then**
    apply R to $\mathcal{A}$ yielding $\mathcal{A}_1,\dots,\mathcal{A}_k$ ($k \in \{2,3\}$)
    if $\mathsf{cons}(\mathcal{A}_i,\mathcal{T}) = \mathsf{consistent}$ for some $i \in \{1,\dots,k\}$ **then**
      return consistent
    return inconsistent
  if $\mathcal{A}$ contains a clash **then**
    return inconsistent
  if $\mathsf{sat}(\bigsqcap_{C\in\mathcal{A}(a)} C, \mathcal{T}) = \mathsf{satisfiable}$ for every $a \in \mathsf{O_a}$ in $\mathcal{A}$ **then**
    return consistent
  return inconsistent

Figure 6.9: The $\mathcal{ALC}(\mathsf{P})$ precompletion algorithm.

3. $\mathcal{A}$ is not concrete domain satisfiable.

If $\mathcal{A}$ does not contain a clash, then $\mathcal{A}$ is *clash-free*. ◇

The precompletion algorithm itself is given in Figure 6.9. In the formulation of the algorithm, we use $\mathsf{sat}(C,\mathcal{T})$ to denote the result of applying the satisfiability algorithm from the previous section to the concept C and TBox $\mathcal{T}$. Since the order of rule application is obviously not important, it is easily seen that the precompletion algorithm computes all precompletions of the input ABox w.r.t. the input TBox. Moreover, for every abstract object a in each such precompletion $\mathcal{A}$, the algorithm calls the sat algorithm to decide the satisfiability of the reduction concept

$$\mathsf{con}(\mathcal{A},a) := \bigsqcap_{C\in\mathcal{A}(a)} C.$$

Note that there exists a fundamental difference between the $\mathcal{ALC}(\mathsf{P})$-precompletion algorithm and the $\mathcal{ALCF}(\mathcal{D})$-precompletion algorithm presented in Section 3.2: in the $\mathcal{ALCF}(\mathcal{D})$ case, we expanded the input ABox such that the "concrete part" of models for the resulting precompletion is independent of the concrete parts of models for the reduction concepts. More precisely, we generated all feature successors of objects in the ABox, all feature successors of these successors, and so on. We then generated a reduction concept for each R-successor of objects in the precompleted ABox with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, and concrete information could not pass this "R-boundary" since only features were admitted inside concrete paths. This does not work in the case of $\mathcal{ALC}(\mathsf{P})$ due to general TBoxes, in whose presence the closure under feature successors is no longer polynomial, it is indeed not even finite. Hence, we cannot separate the concrete parts of the ABox and of the reduction concepts. Instead, we ensure that the concrete part of models for the reduction concepts can be "plugged into" solutions for the predicate conjunction $\zeta_{\mathcal{A}}$ induced by the precompleted ABox $\mathcal{A}$. This is the purpose of the rules Rc1, Rc2, Rc3, and Rch (see also Lemma 6.24 below).

We now prove termination and investigate the time requirements of the algorithm. First, we establish an upper bound for the number of rules that may be applied to a given ABox (independently of the algorithm).

Lemma 6.21. *Let $\mathcal{A}$ be an ABox, $\mathcal{T}$ a TBox, and $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k$ with $\mathcal{A}_0 = \mathcal{A}$ a sequence of ABoxes obtained by repeated rule application. Then $k \leq p(|\mathcal{A}| + |\mathcal{T}|)$ for some polynomial $p(n)$.*

Proof. We abbreviate $|\mathcal{A}| + |\mathcal{T}|$ by n. Each of the rules R$\sqcap$, R$\sqcup$, R$\exists$f, R$\forall$, Rch, and R$\doteq$ adds a new concept to the label of an abstract object. Since all added concepts are from the set

$$\chi := \mathsf{cl}(\mathcal{A}, \mathcal{T}) \cup \{\exists g_1, g_2.P \mid P \in \{<,=\} \text{ and } g_1, g_2 \text{ used in } \mathsf{cl}(\mathcal{A}, \mathcal{T})\}$$

and $|\chi| \leq 2n^2 + n$, the number of applications of the above rules per abstract object is also bounded by $2n^2 + n$ and their overall number of applications is bounded by $2n^3 + n^2$. There are four remaining rules:

- Rc1, Rc2, Rc3. These rules may be applied at most once per concept $\exists u_1, u_2.P \in \chi$ and abstract object a in $\mathcal{A}$. Since no new abstract objects are introduced, there are at most $2n^3 + n^2$ applications of Rc1, Rc2 and Rc3. Moreover, since each rule application introduces at most 2 new concrete objects and Rc2 and Rc3 are the only rules to introduce new concrete objects, it also follows that the number of newly introduced concrete objects is bounded by $4n^3 + 2n^2$.

- Rfe. The rule is applied at most once per (concrete or abstract) object. The initial ABox contains at most n abstract or concrete objects, no new abstract objects are generated, and at most $4n^3 + 2n^2$ new concrete objects are generated. Hence, the number of applications of Rfe is bounded by $4n^3 + 2n^2 + n$.

Taking together these observations, it is obvious that there exists a polynomial $p(n)$ as required.

❑

We can now prove termination.

Proposition 6.22 (Termination). *If started on an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$, the precompletion algorithm terminates after time exponential in $|\mathcal{A}| + |\mathcal{T}|$.*

Proof. Assume that the precompletion algorithm is started on an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$. The precompletion algorithm is a recursive procedure. In every recursion step, either several recursion calls or several calls to the `sat` algorithm are made. Obviously, a run of the algorithm induces a recursion tree, where nodes in the tree are recursion steps and edges are recursion calls. These recursion trees have the following properties:

1. Since at most three recursion calls are made per recursion step, the outdegree is three.

2. Every path of the recursion tree induces a sequence of ABoxes $\mathcal{A}_0, \mathcal{A}_1, \ldots$ with $\mathcal{A}_0 = \mathcal{A}$ that can be obtained by repeated rule application. By Lemma 6.21, the length of this sequence is bounded by $p(|\mathcal{A}| + |\mathcal{T}|)$, and, thus, the depth of recursion trees is also bounded by $p(|\mathcal{A}| + |\mathcal{T}|)$.

This implies that the total number of recursion steps made by the algorithm is bounded by $3^{p(|\mathcal{A}|+|\mathcal{T}|)}$. Since none of the rules introduces new abstract objects, the number of sat calls per recursion step is bounded by $|\mathcal{A}|$ and the total number of calls to sat by $3^{p(|\mathcal{A}|+|\mathcal{T}|)} \cdot |\mathcal{A}|$. Together with Theorem 6.17, we obtain termination and the exponential time bound. ❏

We now prove a series of lemmas that will finally allow to establish soundness and completeness of the precompletion algorithm. We start with showing that the construction of precompletions preserves (un)satisfiability.

Lemma 6.23. *Let $\mathcal{A}$ be an ABox and $\mathcal{T}$ be a TBox. Then $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ iff there exists a precompletion $\mathcal{A}'$ of $\mathcal{A}$ w.r.t. $\mathcal{T}$ such that $\mathcal{A}'$ is consistent w.r.t. $\mathcal{T}$.*

Proof. Recall that $\mathcal{A}'$ is a precompletion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ if $\mathcal{A}'$ can be obtained from $\mathcal{A}$ by exhaustive rule application (using the TBox $\mathcal{T}$). By Lemma 6.21, exhaustive rule application always terminates. Hence, we only need to show that, if a precompletion rule R is applicable to an ABox $\mathcal{A}$, then $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ iff R can be applied to $\mathcal{A}$ such that an ABox $\mathcal{A}'$ is obtained which is consistent w.r.t. $\mathcal{T}$.

We make a case distinction according to the type of R. The rules $\mathsf{R}\sqcap$, $\mathsf{R}\sqcup$, $\mathsf{R}\exists\mathsf{f}$, $\mathsf{R}\forall$, $\mathsf{Rc1}$, $\mathsf{Rc2}$, $\mathsf{Rc3}$, and Rfe can be treated similarly to the corresponding rules in the proof of Part 1 of Lemma 3.9. Hence, we concentrate on the Rch and $\mathsf{R}\doteq$ rules. In both cases, the "if" direction is trivial since $\mathcal{A} \subseteq \mathcal{A}'$ if $\mathcal{A}'$ is obtained from $\mathcal{A}$ by rule application. Hence, every model of $\mathcal{A}'$ and $\mathcal{T}$ is clearly also a model of $\mathcal{A}$ and $\mathcal{T}$. It remains to prove the "only if" direction.

- $\mathsf{R} = \mathsf{Rch}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$ and $\mathcal{T}$ and assume that the Rch rule is applied to an abstract object a, its concrete g_1-successor x_1, and its concrete g_2-successor x_2. The rule application adds one of the assertions $a : \exists g_1, g_2.<$, $a : \exists g_1, g_2.=$, and $a : \exists g_2, g_1.<$. Since $\mathcal{I}$ is a model of $\mathcal{A}$, there exist $r_1, r_2 \in \mathbb{R}$ such that $x_1^{\mathcal{I}} = r_1$ and $x_2^{\mathcal{I}} = r_2$. Trivially, we have either $r_1 < r_2$, $r_1 = r_2$, or $r_2 < r_1$. Hence, the Rch rule can be applied such that $\mathcal{I}$ is a model of the resulting ABox $\mathcal{A}'$.
- $\mathsf{R} = \mathsf{R}\doteq$. Let $\mathcal{I}$ be a model of $\mathcal{A}$ and $\mathcal{T}$. The rule application adds $a : C_{\mathcal{T}}$ for some $a \in \mathsf{O_a}$. By definition of $C_{\mathcal{T}}$, we have $d \in C_{\mathcal{T}}^{\mathcal{I}}$ for every $d \in \Delta_{\mathcal{I}}$. Hence, $\mathcal{I}$ is clearly also a model of the resulting ABox $\mathcal{A}'$.

❏

Our next aim is to show that every clash-free precomplete ABox, for which all reduction concepts are satisfiable, is consistent. We start with a technical lemma that states that, intuitively, for every precomplete ABox $\mathcal{A}$ with satisfiable reduction concepts, we can find models for the reduction concepts such that their concrete parts can be "plugged into" solutions for the predicate conjunction $\zeta_{\mathcal{A}}$ induced by $\mathcal{A}$. Recall that we use $\mathsf{con}(\mathcal{A}, a)$ to denote the reduction concept for $a \in \mathsf{O_a}$ in $\mathcal{A}$.

Lemma 6.24. *Let $\mathcal{A}$ be a precomplete ABox, δ a solution for $\zeta_\mathcal{A}$, and $a \in \mathsf{O_a}$ used in $\mathcal{A}$. If $\mathsf{con}(\mathcal{A}, a)$ is satisfiable w.r.t. $\mathcal{T}$, then there exists a model $\mathcal{I}$ of $\mathsf{con}(\mathcal{A}, a)$ and $\mathcal{T}$ and a $d_a \in \mathsf{con}(\mathcal{A}, a)^\mathcal{I}$ such that, for all $(a, x) : g \in \mathcal{A}$, we have $g^\mathcal{I}(d_a) = \delta(x)$.*

Proof. Let $\mathcal{A}$, $\zeta_\mathcal{A}$, and δ be as in the lemma, and let $\mathcal{I}$ be a model of $\mathsf{con}(\mathcal{A}, a)$ and $\mathcal{T}$. Moreover, let d_a be an arbitrary element of $\mathsf{con}(\mathcal{A}, a)^\mathcal{I}$. We show that $\mathcal{I}$ can be transformed into a model $\mathcal{J}$ such that $\mathcal{J}$ and d_a are as required.

In the following, we assume that there exists a well-founded linear ordering on the set $\Delta_\mathcal{I} \times \mathsf{N_{cF}}$. This can be done w.l.o.g. since it is a byproduct of the proof of Lemma 6.13 that, if a concept C is satisfiable w.r.t. a TBox $\mathcal{T}$, then there exist a model of C and $\mathcal{T}$ (the one constructed in the proof) for which such an ordering exists. We construct the model $\mathcal{J}$ from $\mathcal{I}$ by modifying the interpretations of concrete features in an appropriate way. To do this, we successively "mark" pairs in $\Delta_\mathcal{I} \times \mathsf{N_{cF}}$ such that a pair (d, g) is marked iff $g^\mathcal{J}(d)$ has already been determined. During the construction of $\mathcal{J}$, the following invariant will always hold:

$$\begin{array}{l}\text{if } (d_1, g_1), (d_2, g_2) \in \Delta_\mathcal{I} \times \mathsf{N_{cF}} \text{ are marked, then} \\ \quad g_1^\mathcal{I}(d_1)\, P\, g_2^\mathcal{I}(d_2) \text{ with } P \in \{<, =, >\} \text{ implies } g_1^\mathcal{J}(d_1)\, P\, g_2^\mathcal{J}(d_2)\end{array} \qquad (*)$$

Initially, each pair in $\Delta_\mathcal{I} \times \mathsf{N_{cF}}$ is unmarked. The construction of $\mathcal{J}$ consists of an initial step and a looping step.

1. Initial step. For all $(a, x) : g \in \mathcal{A}$, set $g^\mathcal{J}(e) := \delta(x)$ and mark the pair (e, g).

   We need to show that $(*)$ is satisfied. Hence, fix two marked pairs (e, g_1) and (e, g_2) from $\Delta_\mathcal{I} \times \mathsf{N_{cF}}$. Since both pairs are marked, we have $\{(a, x_1) : g_1, (a, x_2) : g_2\} \subseteq \mathcal{A}$ for some $x_1, x_2 \in \mathsf{O_c}$. Since neither the Rch nor the $\mathsf{Rc1}$ rule is applicable, we have either (i) $\exists g_1, g_2.< \in \mathcal{A}(a)$ and $(x_1, x_2) : < \in \mathcal{A}$, (ii) $\exists g_1, g_2.= \in \mathcal{A}(a)$ and $(x_1, x_2) : = \in \mathcal{A}$, or (iii) $\exists g_2, g_1.< \in \mathcal{A}(a)$ and $(x_2, x_1) : < \in \mathcal{A}$. We only treat case (i) exemplarily. By definition of $\mathsf{con}(\mathcal{A}, a)$ and since $e \in \mathsf{con}(\mathcal{A}, a)^\mathcal{I}$, we have $e \in (\exists g_1, g_2.<)^\mathcal{I}$ and thus $g_1^\mathcal{I}(e) < g_2^\mathcal{I}(e)$. From $(x_1, x_2) : < \in \mathcal{A}$ and the definition of $\zeta_\mathcal{A}$, it follows that $\delta(x_1) < \delta(x_2)$ and hence $g_1^\mathcal{J}(e) < g_2^\mathcal{J}(e)$. Cases (ii) and (iii) are analogous.

2. Looping step. Choose the least unmarked pair (d, g) from $\Delta_\mathcal{I} \times \mathsf{N_{cF}}$ (w.r.t. the assumed ordering) for which $g^\mathcal{I}(d)$ is defined. For $P \in \{<, =, >\}$, let Ψ_P be the set of marked pairs $(d_1, g_1) \in \Delta_\mathcal{I} \times \mathsf{N_{cF}}$ for which $g_1^\mathcal{I}(d_1)\, P\, g^\mathcal{I}(d)$. By $(*)$, we have

   - $g_1^\mathcal{J}(d_1) = g_2^\mathcal{J}(d_2)$ for all $(d_1, g_1), (d_2, g_2) \in \Psi_=$,
   - $g_1^\mathcal{J}(d_1) < g_2^\mathcal{J}(d_2)$ for all $(d_1, g_1) \in \Psi_<$ and $(d_2, g_2) \in \Psi_= \cup \Psi_>$, and
   - $g_1^\mathcal{J}(d_1) < g_2^\mathcal{J}(d_2)$ for all $(d_1, g_1) \in \Psi_=$ and $(d_2, g_2) \in \Psi_>$.

   Hence, due to the density of $\mathbb{R}$, there exists an $r \in \mathbb{R}$ such that

   - $r > \max\{g_1^\mathcal{J}(d_1) \mid (d_1, g_1) \in \Psi_<\}$,
   - $r = g_1^\mathcal{J}(d_1)$ for all $(d_1, g_1) \in \Psi_=$, and

- $r < \min\{g_1^{\mathcal{J}}(d_1) \mid (d_1, g_1) \in \Psi_>\}$.

Set $g^{\mathcal{J}}(d) := r$. Obviously, $(*)$ is satisfied.

It is straightforward to show by structural induction that $d \in C^{\mathcal{I}}$ iff $d \in C^{\mathcal{J}}$ for all $d \in \Delta_{\mathcal{I}}$ and all $\mathcal{ALC}(\mathsf{P})$-concepts C. Hence, $\mathcal{J}$ is a model of $\mathsf{con}(\mathcal{A}, a)$ and $\mathcal{T}$. By the initial step of its construction, $\mathcal{J}$ is as required. ❑

The following lemma is central for proving soundness and completeness.

Lemma 6.25. *Let $\mathcal{A}$ be a clash-free precompletion of an ABox $\mathcal{A}'$ w.r.t. a TBox $\mathcal{T}$. $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ iff $\mathsf{con}(\mathcal{A}, a)$ is satisfiable w.r.t. $\mathcal{T}$ for every $a \in \mathsf{O_a}$ used in $\mathcal{A}$.*

Proof. The "only if" direction is straightforward. Suppose that $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$, let $\mathcal{I}$ be a model of $\mathcal{A}$ and $\mathcal{T}$, and let $a \in \mathsf{O_a}$ be used in $\mathcal{A}$. Since $\mathcal{I}$ is a model of $\mathcal{A}$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all $C \in \mathcal{A}(a)$. By the semantics of the conjunction constructor, this clearly implies $a^{\mathcal{I}} \in \mathsf{con}(\mathcal{A}, a)^{\mathcal{I}}$. Hence, $\mathcal{I}$ is a model of $\mathsf{con}(\mathcal{A}, a)$ and $\mathcal{T}$.

Now for the "if" direction. Let $\mathfrak{A}$ denote the set of abstract objects $a \in \mathsf{O_a}$ appearing in $\mathcal{A}$. Since $\mathcal{A}$ is clash-free, there exists a solution δ for $\zeta_{\mathcal{A}}$. For every $a \in \mathfrak{A}$, fix a model $\mathcal{I}_a$ of $\mathsf{con}(\mathcal{A}, a)$ and $\mathcal{T}$ and a domain element $d_a \in \Delta_{\mathcal{I}_a}$ such that $d_a \in \mathsf{con}(\mathcal{A}, a)^{\mathcal{I}_a}$. By Lemma 6.24, we may assume w.l.o.g. that, for all $a \in \mathfrak{A}$,

$$(a, x) : g \in \mathcal{A} \text{ implies } g^{\mathcal{I}_a}(d_a) = \delta(x). \qquad (*)$$

Moreover, we assume that (i) $a \neq b$ implies $\Delta_{\mathcal{I}_a} \cap \Delta_{\mathcal{I}_b} = \emptyset$ and (ii) none of the d_a has incoming edges, i.e., $(d, d_a) \notin R^{\mathcal{I}_a}$ for all $d \in \Delta_{\mathcal{I}_a}$ and $R \in \mathsf{N_R}$. It is straightforward to prove that none of these assumptions restricts generality: for example, take for each $a \in \mathfrak{A}$ the canonical model constructed from a Hintikka-tree for $\mathsf{con}(\mathcal{A}, a)$ and $\mathcal{T}$ as in the proof of Lemma 6.13. Then apply the modification from the proof of Lemma 6.24 and finally make all domains $\mathcal{I}_a$ disjoint by renaming. Clearly, $(*)$, (i), and (ii) are satisfied for the resulting set of models. In the following, we define an interpretation $\mathcal{I}$ by taking the "union" of the models $\mathcal{I}_a$ with $a \in \mathfrak{A}$ and the relational structure defined by the ABox. However, we have to be careful not to obtain too many abstract feature successors and prefer successors from the ABox over successors from the models.

1. $\Delta_{\mathcal{I}} := \bigcup_{a \in \mathfrak{A}} \Delta_{\mathcal{I}_a}$,
2. $A^{\mathcal{I}} := \bigcup_{a \in \mathfrak{A}} A^{\mathcal{I}_a}$ for all $A \in \mathsf{N_C}$,
3. $R^{\mathcal{I}} := \{(d_a, d_b) \mid (a, b) : R \in \mathcal{A}\} \cup \bigcup_{a \in \mathfrak{A}} R^{\mathcal{I}_a}$ for all $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$,
4. $f^{\mathcal{I}} := \{(d_a, d_b) \mid (a, b) : f \in \mathcal{A}\} \cup \bigcup_{a \in \mathfrak{A}} \{(d, e) \in f^{\mathcal{I}_a} \mid d \neq d_a$ or a has no f-successor in $\mathcal{A}\}$ for all $f \in \mathsf{N_{aF}}$,
5. $g^{\mathcal{I}} := \bigcup_{a \in \mathfrak{A}} g^{\mathcal{I}_a}$ for all $g \in \mathsf{N_{cF}}$,
6. $a^{\mathcal{I}} := d_a$ for all $a \in \mathfrak{A}$, and
7. $x^{\mathcal{I}} := \delta(x)$ for all $x \in \mathsf{O_c}$ appearing in $\mathcal{A}$.

Note that, for all $f \in \mathsf{N_{aF}}$, $f^{\mathcal{I}}$ is functional since the Rfe rule is not applicable to $\mathcal{A}$. Since none of the d_a has incoming edges, the following claim can be proved straightforwardly by structural induction:

Claim 1: For all objects $a \in \mathcal{A}$, domain elements $d \in \Delta_{\mathcal{I}_a}$ with $d \neq d_a$, and $\mathcal{ALC}(\mathsf{P})$-concepts C, we have $d \in C^{\mathcal{I}_a}$ iff $d \in C^{\mathcal{I}}$.

However, we still need to deal with the elements d_a themselves.

Claim 2: For all objects $a \in \mathcal{A}$, $C \in \mathcal{A}(a)$ implies $d_a \in C^{\mathcal{I}}$.

The proof is by induction on the structure of C. The induction start consists of three cases:

- $C \in \mathsf{N_C}$. Straightforward by definition of $\mathsf{con}(\mathcal{A}, a)$, the choice of $\mathcal{I}_a$ and d_a, and the construction of $\mathcal{I}$.

- $C = \exists u_1, u_2.P$. By definition of $\mathsf{con}(\mathcal{A}, a)$ and choice of $\mathcal{I}_a$ and d_a, $C \in \mathcal{A}(a)$ implies $d_a \in C^{\mathcal{I}_a}$. We make a case distinction according to the form of u_1 and u_2 (recall that all concepts are assumed to be in path normal form).

  1. $u_1 = g_1$ and $u_2 = g_2$. Since $d_a \in (\exists g_1, g_2.P)^{\mathcal{I}_a}$, there exist $r_1, r_2 \in \mathbb{R}$ such that $g_1^{\mathcal{I}_a}(d_a) = r_1$, $g_2^{\mathcal{I}_a}(d_a) = r_2$, and $r_1 P r_2$. By definition of $\mathcal{I}$, this implies $g_1^{\mathcal{I}}(d_a) = r_1$, $g_2^{\mathcal{I}}(d_a) = r_2$ and thus $d_a \in (\exists g_1, g_2.P)^{\mathcal{I}}$.
  2. $u_1 = fg_1$ and $u_2 = g_2$. We have to distinguish two subcases. First assume that a has an f-successor b in $\mathcal{A}$. Since the Rc2 rule is not applicable, there exist $x_1, x_2 \in \mathsf{O_c}$ such that $\{(a, x_1) : g_1, (b, x_2) : g_2, (x_1, x_2) : P\} \in \mathcal{A}$. Since δ is a solution for $\zeta_{\mathcal{A}}$, there clearly exist $r_1, r_2 \in \mathbb{R}$ such that $r_1 = \delta(x_1)$, $r_2 = \delta(x_2)$, and $r_1 P r_2$. Since $\mathcal{I}_a$ and $\mathcal{I}_b$ satisfy $(*)$, we have $g_1^{\mathcal{I}_a}(d_a) = r_1$ and $g_2^{\mathcal{I}_b}(d_b) = r_2$. By construction of $\mathcal{I}$, we have $f^{\mathcal{I}}(d_a) = d_b$, $g_1^{\mathcal{I}}(d_a) = g_1^{\mathcal{I}_a}(d_a)$, and $g_2^{\mathcal{I}}(d_b) = g_2^{\mathcal{I}_b}(d_b)$. Hence, $g_1^{\mathcal{I}}(d_a) P g_2^{\mathcal{I}}(d_b)$ and $d_a \in (\exists fg_1, g_2.P)^{\mathcal{I}}$.

     Now assume that a has no f-successor b in $\mathcal{A}$. From $d_a \in (\exists fg_1, g_2.P)^{\mathcal{I}_a}$ and the construction of $\mathcal{I}$, it follows straightforwardly (similar to Case 1) that $d_a \in (\exists fg_1, g_2.P)^{\mathcal{I}}$.
  3. $u_1 = g_1$ and $u_2 = fg_2$. Analogous to the previous case using Rc3 instead of Rc2.

- $C = g\uparrow$. As in the previous case, $C \in \mathcal{A}(a)$ implies $d_a \in C^{\mathcal{I}_a}$. Hence, $g^{\mathcal{I}_a}(d_a)$ is undefined. By definition of $\mathcal{I}$, $g^{\mathcal{I}}(d_a)$ is also undefined and thus $d_a \in (g\uparrow)^{\mathcal{I}}$.

For the induction step, we make a case distinction according to the topmost constructor in C:

- $C = C_1 \sqcap C_2$. Since the R$\sqcap$ rule is not applicable to $\mathcal{A}$ and $C \in \mathcal{A}(a)$, we have $\{C_1, C_2\} \subseteq \mathcal{A}(a)$. The induction hypothesis yields $d_a \in C_1^{\mathcal{I}}$ and $d_a \in C_2^{\mathcal{I}}$. By the semantics, we obtain $d_a \in C^{\mathcal{I}}$.

- $C = C_1 \sqcup C_2$. Similar to the previous case.

- $C = \exists R.D$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. By definition of $\mathsf{con}(\mathcal{A}, a)$ and choice of $\mathcal{I}_a$ and d_a, $C \in \mathcal{A}(a)$ implies $d_a \in C^{\mathcal{I}_a}$. Hence, there exists an $e \in \Delta_{\mathcal{I}_a}$ such that $(d_a, e) \in R^{\mathcal{I}_a}$ and $e \in D^{\mathcal{I}_a}$. Since d_a has no incoming edges in $\mathcal{I}_a$ (see above), we have $d_a \neq e$. Hence, by Claim 1, $e \in D^{\mathcal{I}_a}$ implies $e \in D^{\mathcal{I}}$. By construction of $\mathcal{I}$, we additionally have $(d_a, e) \in R^{\mathcal{I}}$ and thus $d_a \in (\exists R.D)^{\mathcal{I}}$.

- $C = \exists f.D$. If there is no $b \in \mathsf{O_a}$ such that $(a, b) : f \in \mathcal{A}$, then we can argue as in the previous case. Hence assume that such a b exists. Since the R$\exists$f rule is not applicable, we have $b : D \in \mathcal{A}$. By induction, we have $d_b \in D^{\mathcal{I}}$. Since we have $f^{\mathcal{I}}(d_a) = d_b$ by construction of $\mathcal{I}$, we obtain $d_a \in (\exists f.D)^{\mathcal{I}}$.

- $C = \forall R.D$. Fix a pair $(d_a, e) \in R^{\mathcal{I}}$. By definition of $\mathcal{I}$, we have either $(d_a, e) \in R^{\mathcal{I}_a}$ or $e = d_b$ and $(a, b) : R \in \mathcal{A}$. In the first case, we have $e \neq d_a$ since d_a has no incoming edges in $\mathcal{I}_a$ and $e \in D^{\mathcal{I}}$ by the semantics and Claim 1. In the second case, we have $D \in \mathcal{A}(b)$ since the R$\forall$ rule is not applicable to $\mathcal{A}$. Hence, by induction, $e \in D^{\mathcal{I}}$ and thus $d_a \in (\forall R.D)^{\mathcal{I}}$.

This finishes the proof of Claim 2.

Using the two claims, it is easy to show that $\mathcal{I}$ is a model of $\mathcal{A}$ and $\mathcal{T}$. We first show that $\mathcal{I}$ satisfies every assertion in $\mathcal{A}$. For assertions of the form $a : C$, we have $a^{\mathcal{I}} = d_a \in C^{\mathcal{I}}$ by Claim 2. Assertions $(a, b) : R$ are obviously satisfied by definition of $\mathcal{I}$. Assertions $(a, x) : g$ are satisfied by construction of $\mathcal{I}$ and since the models $\mathcal{I}_b$ (for $b \in \mathfrak{A}$) satisfy $(*)$. Finally, assertions $(x_1, x_2) : P$ are satisfied since δ is a solution for $\zeta_{\mathcal{A}}$.

It remains to show that $\mathcal{I}$ is a model of $\mathcal{T}$. Fix a concept equation $C \doteq D \in \mathcal{T}$ and a $d \in \Delta_{\mathcal{I}}$. First assume that $d \neq d_a$ for all $a \in \mathfrak{A}$. Let $d \in \Delta_{\mathcal{I}_a}$. Then $d \in C^{\mathcal{I}}$ iff $d \in D^{\mathcal{I}}$ by Claim 1 and since $\mathcal{I}_a$ is a model of $\mathcal{T}$. Now assume $d = d_a$. Since the R$\doteq$ rule is not applicable to $\mathcal{A}$, we have $a : C_{\mathcal{T}} \in \mathcal{A}$. Hence, by Claim 2, $d_a \in C_{\mathcal{T}}^{\mathcal{I}}$. By definition of $C_{\mathcal{T}}$, this clearly implies $d_a \in C^{\mathcal{I}}$ iff $d_a \in D^{\mathcal{I}}$. ❏

Finally, we prove soundness and completeness.

Proposition 6.26 (Soundness and Completeness). *If the precompletion algorithm is started on an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$, then it returns* consistent *if $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ and* inconsistent *otherwise.*

Proof. Let $\mathcal{A}$ and $\mathcal{T}$ be an input to the precompletion algorithm. It is easily seen that the algorithm computes all precompletions of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and, for each clash-free precompletion $\mathcal{A}'$, checks whether the reduction concept $\mathsf{con}(\mathcal{A}, a)$ is satisfiable for all $a \in \mathsf{O_a}$ occurring in $\mathcal{A}'$. It returns consistent if it finds a precompletion for which all reduction concepts are satisfiable and, by Proposition 6.22, inconsistent otherwise. Soundness and completeness are now an immediate consequence of Lemmas 6.23 and 6.25. ❏

Taking together Propositions 6.22 and 6.26, we obtain an ExpTime upper bound for $\mathcal{ALC}$(P)-ABox consistency w.r.t. general TBoxes. Together with the lower bound from Theorem 6.17, we obtain the following result.

Theorem 6.27. *$\mathcal{ALC}(\mathsf{P})$-ABox consistency w.r.t. general TBoxes is* ExpTime*-complete.*

Extending the translation of $\mathcal{ALC}(\mathtt{I})$-concepts to $\mathcal{ALC}(\mathsf{P})$-concepts from Section 2.4.3 to ABoxes and TBoxes in the obvious way, we obtain the following corollary.

Corollary 6.28. *$\mathcal{ALC}(\mathtt{I})$-ABox consistency w.r.t. general TBoxes is* ExpTime*-complete.*

6.3 Related Work and Discussion

During the last years, a variety of temporal Description Logics have been proposed that differ widely w.r.t. their expressive power and computational properties, see [Artale & Franconi 2001] for a (slightly outdated) overview. One of the most fundamental decisions to be made when defining a temporal DL is whether time points or time intervals should be the atomic temporal entity. In Modal Logics, the point-based approach seems to be the most popular one [Goldblatt 1974; Gabbay *et al.* 1994; van Benthem 1996] while in Artificial Intelligence interval-based formalisms are prevalent [Ginsberg 1993; Stock 1997]. In Description Logic research, which is situated at the intersection of these two areas, both point-based and interval-based temporal DLs have attracted considerable attention. Since we view $\mathcal{ALC}(\mathsf{P})$ primarily as an interval-based logic, we will not discuss point-based DLs in detail and refer the interested reader to, e.g., [Schild 1993; Wolter & Zakharyaschev 1999; Lutz *et al.* 2001c; 2001d].

The ancestor of most interval-based temporal DLs is the Modal Logic of time intervals defined by Halpern and Shoham [1992] (in the following called HS). On an informal level, the logic HS can be described as follows. The semantics is defined in terms of Kripke structures whose set of worlds is the set of all time intervals over some temporal structure. The logic offers thirteen modal operators—one for each Allen relation—which, intuitively, "quantify over" the Allen relations. Unfortunately, Halpern and Shoham were able to show that HS-formula satisfiability is undecidable over most interesting temporal structures. The Modal Logic HS can straightforwardly be converted into a Description Logic by associating each modal world with a Description Logic interpretation instead of with a set of propositional variables in the style of [Lutz *et al.* 2001d]. Of course, undecidability of HS is inherited by the resulting multi-dimensional logic (let us call it $\mathsf{HS}_{\mathcal{ALC}}$). Based on these observations, researchers tried either to live with undecidability [Bettini 1997] or to find fragments of $\mathsf{HS}_{\mathcal{ALC}}$ that are decidable yet sufficiently expressive [Artale & Franconi 1998].

Since it is nowadays generally accepted that Description Logics should be decidable, let us focus on Artale and Franconi's approach. Basically, their "fragment" $\mathcal{TL}_{\mathcal{ALCF}}$ of $\mathsf{HS}_{\mathcal{ALC}}$ is obtained by restricting negation to atomic concepts and disallowing the use of the box modalities on temporal relations [Artale & Franconi 1998]. Some new flavor is added to the language by admitting a much more general existential modality that allows to describe "relation networks" (hence, $\mathcal{TL}_{\mathcal{ALCF}}$ is not really a fragment of $\mathsf{HS}_{\mathcal{ALC}}$). As shown in [Artale & Lutz 1999], there exists a rather close correspondence between $\mathcal{TL}_{\mathcal{ALCF}}$ and $\mathcal{ALC}(\mathsf{P})$ without general TBoxes since $\mathcal{TL}_{\mathcal{ALCF}}$-concept satisfiability can be rephrased as $\mathcal{ALC}(\mathsf{P})$-concept satisfiability in a natural way. However, the expressive power of logics like $\mathcal{TL}_{\mathcal{ALCF}}$ is seriously limited since universal statements cannot be made at all. For example, if we want to model the behavior of a robot using a temporal Description Logic, it is surely desirable to make universal statements such as "in every robot state, there exists a sane successor state". It is not hard to see that statements of this type are needed in many temporal reasoning applications. One of the distinguishing aspects of the temporal DL $\mathcal{ALC}(\mathsf{P})$ is that it combines interval-based temporal reasoning with general TBoxes that allow to make universal statements of the above form. To the best of our knowledge, $\mathcal{ALC}(\mathsf{P})$ is the first decidable interval-based temporal Description Logic with this important property.

Let us now discuss the results presented in this chapter in some more detail. One important limitation is that the obtained results are only valid if a dense strict linear order is assumed as the underlying temporal structure. For example, the concept $\top$ is satisfiable w.r.t. the TBox

$$\mathcal{T} = \{\top \sqsubseteq \exists g_1, g_2.< \sqcap \exists g_1, fg_1.< \sqcap \exists fg_2, g_2.<\}$$

over the temporal structures $\mathbb{Q}$ and $\mathbb{R}$ (with the natural orderings) but not over $\mathbb{N}$. To see this, note that $\mathcal{T}$ induces a constraint graph as in Figure 6.7. Hence, it would be interesting to investigate how the presented algorithm has to be modified for reasoning with the temporal structure $\mathbb{N}$. It is not hard to see that a constraint graph G is satisfiable over $\mathbb{N}$ iff there exists an upper bound on the length of $<$-paths between any two nodes in G (which also implies that G contains no $<$-cycle). It is, however, not clear how Hintikka-trees and automata can be modified to account for this stronger condition.

There exist several other interesting directions in which the presented results could be extended. We briefly discuss some of them:

- In its current form, the logic $\mathcal{ALC}(\mathsf{P})$ can be used for qualitative temporal reasoning only. In other words, there exist no predicates for referring to specific time intervals or for specifying a precise distance between two intervals. An extension of the logic allowing for this kind of quantitative temporal reasoning could be rather interesting for many application domains.

- It would be interesting to extend $\mathcal{ALC}(\mathsf{P})$ to make it suitable for reasoning about entity relationship (ER) diagrams with temporal integrity constraints. As demonstrated by Calvanese et al. in [Calvanese 1996b; Calvanese *et al.* 1998b], Description Logics can be used for reasoning about ER diagrams with integrity constraints and thus are a valuable tool for database design. Artale and Franconi propose a temporalization of Calvanese's approach that can be used for reasoning about temporal ER diagrams [Artale & Franconi 1999]. They use a point-based logic and focus on temporal databases, i.e., they admit reference to previous database states in the ER model. By using an appropriate extension of $\mathcal{ALC}(\mathsf{P})$, one should be able to capture a different kind of temporal reasoning with ER diagrams, namely reasoning over ER diagrams with integrity constraints for databases that store temporal data. Such an extension would allow to formulate *temporal* integrity constraints, i.e., integrity constraints that take into account the temporal semantics of the data in the database. For example, a temporal integrity constraint could state that employees birthdays should be before their employment date. But what is an appropriate extension of $\mathcal{ALC}(\mathsf{P})$ for reasoning in this domain? Given the results in [Calvanese 1996b], it is clear that we need at least (unqualifying) number restrictions and inverse roles. An extension of the results presented in this section to the more complex logic appears to be possible, but this still needs to be investigated.

- As already mentioned in Section 2.4.3, there exists a set of eight spatial relations called RCC-8 [Randell *et al.* 1992; Bennett 1997] which in many aspects resembles the set of Allen relations. It would be interesting to replace P by a concrete domain whose predicates describe the RCC-8 relations. Our guess is that again a decidable formalism is obtained but many proof techniques would clearly have to be reworked (for example, the RCC-8 relations cannot be broken down to the predicates $\{<, =\}$).

Chapter 7

Summary and Outlook

In this thesis, we have investigated the complexity of reasoning with Description Logics that provide for concrete domains or the closely related feature (dis)agreement constructors. In Chapter 3, we have established the fundamental result stating that satisfiability and subsumption of $\mathcal{ALCF}(\mathcal{D})$-concepts and consistency of $\mathcal{ALCF}(\mathcal{D})$-ABoxes are PSPACE-complete provided that $\mathcal{D}$-satisfiability is in PSPACE (Theorems 3.13 and 3.18). It immediately follows that the Description Logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ also have PSPACE-complete reasoning problems. Starting from these results, we then extended both $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ with standard means of expressivity and found that

1. the PSPACE upper bounds for reasoning with $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ cannot be considered robust since, for many seemingly harmless extensions of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$, the complexity of reasoning jumps from PSPACE-completeness to NEXPTIME-completeness or even to undecidability. With "seemingly harmless" we mean that, for the vast majority of Description Logics that can be found in the literature, the considered extensions do not change the complexity of reasoning (or at least not in such a dramatic way).

2. apart from the obvious syntactic similarity of their constructors, extensions of $\mathcal{ALC}(\mathcal{D})$ and extensions of $\mathcal{ALCF}$ behave very similarly w.r.t. the complexity of reasoning. The only case where the complexity diverges is the extension with an inverse role constructor: in Sections 5.3.3 and 5.4, we showed that $\mathcal{ALC}^{-}(\mathcal{D})$-concept satisfiability is NEXPTIME-complete if $\mathcal{D}$-satisfiability is in NP, whereas, in Section 5.5.2, we proved $\mathcal{ALCF}^{-}$-concept satisfiability to be undecidable.

These two statements are based on the complexity results obtained in Chapters 4 to 6, in which we proved tight complexity bounds for most standard extensions of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$. In the following, we describe the obtained results in more detail. We will not explicitly mention concept subsumption since, in all logics considered, concept satisfiability being complete for some complexity class C implies that concept subsumption is complete for the complement of C.

Benign Extensions

The PSpace upper bounds proved in Chapter 3 show that adding concrete domains and feature (dis)agreements to the Description Logic $\mathcal{ALC}$ does not increase the complexity of reasoning, i.e., concept satisfiability and ABox consistency remain PSpace-complete. As we argued above, these upper bounds are rather fragile and, indeed, most extensions of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ that still have PSpace-complete reasoning problems seem to be of a "non-interacting" type: the logic obtained through extension can be viewed as the fusion of $\mathcal{ALC}(\mathcal{D})$ (resp. $\mathcal{ALCF}$) and an extension of $\mathcal{ALC}$ for which reasoning is in PSpace. For example, this is the case for the extension of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ with qualifying number restrictions or with transitive roles. In Section 5.6, we used this fact to conjecture PSpace-completeness for several extensions of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$.

In Chapter 4, we investigated the extension of Description Logics with acyclic TBoxes. We presented a technique for modifying completion algorithms that use tracing to take into account acyclic TBoxes. This technique was employed to establish PSpace upper bounds for $\mathcal{ALC}$-concept satisfiability w.r.t. acyclic TBoxes and $\mathcal{ALC}$-ABox consistency w.r.t. acyclic TBoxes (Theorems 4.8 and 4.12). Moreover, we generalized these results into rules of thumb which characterize a class of DLs that contain $\mathcal{ALC}$ as a fragment and for which the addition of acyclic TBoxes has no impact on the complexity of concept satisfiability and ABox consistency (pages 75 and 78). It is easily seen that the identified class of DLs contains many "standard" Description Logics considered in the literature. The motivation for establishing these results is to demonstrate that adding acyclic TBoxes is "usually harmless" w.r.t. the complexity of reasoning. If contrasted with the results described next, this yields an indication for the non-robustness of the $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ PSpace upper bounds.

Malicious Extensions

In Chapters 4 and 5, we considered extensions of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ that are not well-behaved in the sense that reasoning with the extended logics is considerably harder than reasoning with $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ themselves. More precisely, we investigated the extensions of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ with

- acyclic TBoxes,
- the role conjunction constructor,
- the inverse role constructor,
- generalized concrete domain constructors, and
- the concrete domain role constructor.

Obviously, the last two extensions only make sense for $\mathcal{ALC}(\mathcal{D})$ but not for $\mathcal{ALCF}$. Let us summarize the obtained complexity results. The central theorems state that, for each of the above five extensions of $\mathcal{ALC}(\mathcal{D})$, concept satisfiability is NExpTime-hard if $\mathcal{D}$ is an arithmetic concrete domain (Theorems 5.16, 5.20, 5.24, 5.28, and 5.32).

These results are rather surprising since, as already mentioned, most of the above means of expressivity are usually considered harmless w.r.t. the complexity of reasoning. To illustrate the relevance of the obtained results, we should like to stress that it is a relatively weak requirement for a concrete domain to be arithmetic. Indeed, most concrete domains considered in the literature are arithmetic. As a corresponding upper bound, we showed that reasoning with all the above extensions of $\mathcal{ALC}(\mathcal{D})$ put together is in NExpTime if $\mathcal{D}$-satisfiability is in NP (Theorem 5.58).

Concerning $\mathcal{ALCF}$, we have proved NExpTime-completeness for the extension of this logic with acyclic TBoxes, role conjunction, and both (Theorems 4.17, 4.23, and 5.61). As already mentioned above, the extension of $\mathcal{ALCF}$ with the inverse role constructor yields an undecidable logic (Theorem 5.63). Note that both $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ are counterexamples to the observation from Chapter 4 that, usually, the addition of acyclic TBoxes does not increase the complexity of reasoning. For all the above results, we considered concept satisfiability but not ABox consistency (which, however, we conjecture to have the same complexity in all cases).

General TBoxes

Completing our investigation of Description Logics with concrete domains and (various types of) TBoxes, in Chapter 6 we performed an in-depth analysis of the extension of $\mathcal{ALC}(\mathcal{D})$ with general TBoxes. The fundamental observation is a negative one, namely that $\mathcal{ALC}(\mathcal{D})$-concept satisfiability w.r.t. general TBoxes is undecidable for every arithmetic concrete domain $\mathcal{D}$ (Theorem 6.2). However, the situation is not completely hopeless since there exist interesting concrete domains that are *not* arithmetic: we defined the temporal concrete domain P and proved that satisfiability of $\mathcal{ALC}(\mathsf{P})$-concepts w.r.t. general TBoxes is decidable and ExpTime-complete (Theorem 6.17). To demonstrate the usefulness of this decidability result, we motivated $\mathcal{ALC}(\mathsf{P})$ as a powerful tool for mixed point-based and interval-based temporal reasoning. In Section 6.2.4, we then extended the upper bound to $\mathcal{ALC}(\mathsf{P})$-ABox consistency thus proving this reasoning task to be also ExpTime-complete (Theorem 6.27). We did not consider the extension of $\mathcal{ALCF}$ with general TBoxes since it is well-known to be undecidable [Baader *et al.* 1993].

After summarizing the obtained results, let us highlight some promising future research topics. Although we determined the complexity of most standard extensions of $\mathcal{ALC}(\mathcal{D})$ in this thesis, there remain some natural extensions which we did not address. Among these are several for which tight complexity bounds should be easy to obtain and also some for which the complexity is not immediately clear. An example for the former group is the extension of $\mathcal{ALC}(\mathcal{D})$ with a role disjunction constructor. For this logic, it should be straightforward to prove PSpace-completeness of the standard reasoning problems by "coding out" disjunction as done, e.g., in [Lutz & Sattler 2001]. The latter group of extensions includes, for example, $\mathcal{ALCO}(\mathcal{D})$, which is $\mathcal{ALC}(\mathcal{D})$ extended with nominals [Areces & de Rijke 2001; Tobies 2001a]. The complexity of this logic is easily seen to be between PSpace and NExpTime, but the exact complexity is as of now unknown (note that the tracing-style completion algorithm from Chapter 3 can *not* easily be extended to nominals).

The greatest potential for future work seems to lie in the extension of Description Logics with concrete domains and general TBoxes. As discussed in more detail in Section 6.3, there are several promising directions for further research: one could extend the logic $\mathcal{ALC}(\mathsf{P})$ by additional concept and role constructors to make it a suitable tool for reasoning about entity relationship diagrams with temporal integrity constraints; or one could extend $\mathcal{ALC}(\mathsf{P})$, which is restricted to qualitative temporal reasoning, by quantitative predicates obtaining more expressive power for the representation of temporal knowledge; finally, it would be interesting to exchange the temporal concrete domain P by a spatial concrete domain based on the set of RCC-8 relations, thus obtaining a powerful spatial Description Logic.

Bibliography

[Abecker *et al.* 1991] A. Abecker, D. Drollinger, and P. Hanschke. Taxon: A concept language with concrete domains. In *Proceedings of the International Workshop on Processing Declarative Knowledge (PDK'91)*, Kaiserslautern, Germany, 1991.

[Allen 1983] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 1983.

[Andréka *et al.* 1998] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[Areces *et al.* 1999] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, number 1683 in Lecture Notes in Computer Science, pages 307–321. Springer-Verlag, 1999.

[Areces & de Rijke 1998] C. Areces and M. de Rijke. Expressiveness revisited. In E. Franconi, G. De Giacomo, R. MacGregor, W. Nutt, and C. Welty, editors, *Proceedings of the 1998 International Workshop on Description Logics (DL'98)*, number 11 in CEUR-WS (http://ceur-ws.org/), pages 35–43, Trento, Italy, 1998.

[Areces & de Rijke 2001] C. Areces and M. de Rijke. From description logics to hybrid logics, and back. In F. Wolter, H. Wansing, M. de Rijke, and M. Zakharyaschev, editors, *Advances in Modal Logics Volume 3*. CSLI Publications, Stanford, CA, USA, 2001.

[Artale & Franconi 1998] A. Artale and E. Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research (JAIR)*, 9:463–506, 1998.

[Artale & Franconi 1999] A. Artale and E. Franconi. Temporal ER modeling with description logics. In *Proceedings of the International Conference on Conceptual Modeling (ER'99)*, vol. 1728 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[Artale & Franconi 2001] A. Artale and E. Franconi. Temporal description logics. In D. Gabbay, M. Fisher, and L. Vila, editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. MIT Press, 2001. To appear.

[Artale & Lutz 1999] A. Artale and C. Lutz. A correspondence between temporal description logics. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, number 22 in CEUR-WS (http://ceur-ws.org/), pages 145–149, 1999.

[Baader *et al.* 1993] F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *Journal of Logic, Language and Information*, 2:1–18, 1993.

[Baader *et al.* 1996] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[Baader *et al.* 1999] F. Baader, R. Molitor, and S. Tobies. Tractable and decidable fragments of conceptual graphs. In W. Cyre and W. Tepfenhart, editors, *Proceedings of the Seventh International Conference on Conceptual Structures (ICCS'99)*, number 1640 in Lecture Notes in Computer Science, pages 480–493. Springer-Verlag, 1999.

[Baader *et al.* 2002a] F. Baader, C. Lutz, H. Sturm, and F. Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research (JAIR)*, 2002. To appear.

[Baader *et al.* 2002b] F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications.* Cambridge University Press, 2002. To appear.

[Baader 1990a] F. Baader. A formal definition for expressive power of knowledge representation languages. In *Proceedings of the Ninth European Conference on Artificial Intelligence (ECAI-90)*, pages 53–58, Stockholm, Sweden, 1990.

[Baader 1990b] F. Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 621–626, Boston, MA, USA, 1990.

[Baader 1991] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 446–451, Sydney, Australia, 1991.

[Baader 1999] F. Baader. Logic-based knowledge representation. In M. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments*, number 1600 in Lecture Notes in Computer Science, pages 13–41. Springer-Verlag, 1999.

[Baader & Hanschke 1991a] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, Australia, 1991.

[Baader & Hanschke 1991b] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. DFKI Research Report RR-91-10, German Research Center for Artificial Intelligence (DFKI), 1991.

[Baader & Hanschke 1992] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proceedings of the 16th German AI-Conference (GWAI-92)*, vol. 671 of *Lecture Notes in Computer Science*, pages 132–143. Springer-Verlag, 1992.

[Baader & Hollunder 1991a] F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

[Baader & Hollunder 1991b] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proceedings of the Workshop on Processing Declarative Knowledge (PDK-91)*, vol. 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86. Springer-Verlag, 1991.

[Baader & Sattler 1998] F. Baader and U. Sattler. Description logics with concrete domains and aggregation. In H. Prade, editor, *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI'98)*, pages 336–340. John Wiley & Sons, 1998.

[Baader & Sattler 2000] F. Baader and U. Sattler. Tableau algorithms for description logics. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*, vol. 1847 of *Lecture Notes in Artificial Intelligence*, pages 1–18. Springer-Verlag, 2000.

[Bennett 1997] B. Bennett. Modal logics for qualitative spatial reasoning. *Journal of the Interest Group in Pure and Applied Logic*, 4(1), 1997.

[Berger 1966] R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66, 1966.

[Bettini 1997] C. Bettini. Time-dependent concepts: representation and reasoning using temporal description logics. *Data & Knowledge Engineering*, 22:1–38, 1997.

[Blackburn *et al.* 2001] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[Börger *et al.* 1997] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, 1997.

[Borgida 1996] A. Borgida. On the relative expressivness of description logics and predicate logics. *Artificial Intelligence*, 82(1 - 2):353–367, 1996.

[Borgida & Patel-Schneider 1994] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, pages 277–308, 1994.

[Brachman *et al.* 1991] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, and A. Borgida. Living with classic: When and how to use a KL-ONE-like language. In J. F. Sowa, editor, *Principles of Semantic Networks – Explorations in the Representation of Knowledge*, chapter 14, pages 401–456. Morgan Kaufmann, 1991.

[Brachman 1978] R. Brachman. Structured inheritance networks. In W. Woods and R. Brachman, editors, *Research in Natural Language Understanding*, Quarterly Progress Report No. 1, BBN Report No. 3742, pages 36–78. Bolt, Beranek and Newman Inc., Cambridge, Mass., 1978.

[Brachman & Schmolze 1985] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[Buchheit *et al.* 1998] M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. A refined architecture for terminological systems: Terminology = schema + views. *Artificial Intelligence*, 99(2):209–260, 1998.

[Calvanese *et al.* 1998a] D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers (North-Holland), Amsterdam, 1998.

[Calvanese *et al.* 1998b] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.

[Calvanese *et al.* 1999] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In D. Thomas, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 84–89. Morgan Kaufmann, 1999.

[Calvanese *et al.* 2000a] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proceedings of the Seventh International Conference on the Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 176–185, 2000.

[Calvanese *et al.* 2000b] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[Calvanese 1996a] D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*, pages 303–307, 1996.

[Calvanese 1996b] D. Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.* Dottorato di ricerca in informatica, Università degli Studi di Roma "La Sapienza", Italia, 1996.

[Chagrov & Zakharyaschev 1996] A. Chagrov and M. Zakharyaschev. *Modal Logic.* Oxford University Press, 1996.

[D'Agostino *et al.* 1999] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors. *Handbook of Tableau Methods.* Kluwer, Dordrecht, 1999.

[Davis 1973] M. Davis. Hilbert's Tenth Problem is unsolvable. *The American Mathematical Monthly*, 80(3):233–269, 1973.

[Donini *et al.* 1997] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 1997.

[Donini & Massacci 2000] F. M. Donini and F. Massacci. ExpTime tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.

[Edelmann & Owsnicki 1986] J. Edelmann and B. Owsnicki. Data models in knowledge representation systems: A case study. In C.-R. Rollinger and W. Horn, editors, *Proceedings of the Tenth German Workshop on Artificial Intelligence (GWAI'86) and the Second Austrian Symposium on Artificial Intelligence (ÖGAI'86)*, vol. 124 of *Informatik-Fachberichte*, pages 69–74. Springer Verlag, 1986.

[Fensel *et al.* 2000] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In *Proceedings of the European Knowledge Acquisition Conference (EKAW 2000)*, vol. 1937 of *Lecture Notes In Artificial Intelligence*. Springer-Verlag, 2000.

[Fine 1972] K. Fine. In so many possible worlds. *Notre Dame Journal of Formal Logic*, 13:516–520, 1972.

[Fischer & Ladner 1979] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[Gabbay *et al.* 1994] D. M. Gabbay, I. M. Hodkinson, and M. A. Reynolds, editors. *Temporal Logic: Mathematical Foundations and Computational Aspects, Volume 1.* Oxford University Press, Logic Guides 28, 1994.

[Gabbay *et al.* 2001] D. Gabbay, M. Fisher, and L. Vila, editors. *Handbook of Time and Temporal Reasoning in Artificial Intelligence.* MIT Press, 2001. To appear.

[Garey & Johnson 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, USA, 1979.

[Gargov *et al.* 1987] G. Gargov, S. Passy, and T. Tinchev. Modal environment for Boolean speculations. In D. Skordev, editor, *Mathematical Logic and Applications*, pages 253–263, New York, USA, 1987. Plenum Press.

[Giacomo & Lenzerini 1994] G. D. Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94). Volume 1*, pages 205–212. AAAI Press, 1994.

[Ginsberg 1993] M. L. Ginsberg. *Essentials of Artificial Intelligence.* Morgan Kaufmann, 1993.

[Goldblatt 1974] R. Goldblatt. Semantic analysis of orthologic. *Journal of Philosophical Logic*, 3:19–35, 1974.

[Goldblatt 1987] R. Goldblatt. *Logics of Time and Computation, Second Edition, Revised and Expanded*, vol. 7 of *CSLI Lecture Notes.* CSLI, Stanford, CA, USA, 1987. Distributed by University of Chicago Press.

[Goranko & Passy 1992] V. Goranko and S. Passy. Using the universal modality: Gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992.

[Goré 1999] R. Goré. Tableau algorithms for modal and temporal logic. In D'Agostino et al. [1999].

[Grädel *et al.* 1997] E. Grädel, P. Kolaitis, and M. Vardi. On the Decision Problem for Two-Variable First-Order Logic. *Bulletin of Symbolic Logic*, 3:53–69, 1997.

[Grädel 1999] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.

[Grädel & Walukiewicz 1999] E. Grädel and I. Walukiewicz. Guarded Fixed Point Logic. In *Proceedings of Fourteenth IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 45–54, 1999.

[Haarslev *et al.* 1999] V. Haarslev, C. Lutz, and R. Möller. A description logic with concrete domains and role-forming predicates. *Journal of Logic and Computation*, 9(3):351–384, 1999.

[Haarslev *et al.* 2001] V. Haarslev, R. Möller, and M. Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning IJCAR'01*, number 2083 in Lecture Notes in Artifical Intelligence, pages 29–44. Springer-Verlag, 2001.

[Haarslev & Möller 2000a] V. Haarslev and R. Möller. Consistency testing: The RACE experience. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*, vol. 1847 of *Lecture Notes in Artificial Intelligence*, pages 57–61. Springer-Verlag, 2000.

[Haarslev & Möller 2000b] V. Haarslev and R. Möller. Expressive ABox reasoning with number restrictions, role hierarchies, and transitively closed roles. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the Seventh International*

Conference on Principles of Knowledge Representation and Reasoning (KR2000), pages 273–284. Morgan Kaufman, 2000.

[Halpern & Moses 1992] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–380, 1992.

[Halpern & Shoham 1991] J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of ACM*, 38(4):935–962, 1991.

[Hanschke 1992] P. Hanschke. Specifying role interaction in concept languages. In W. Nebel, Bernhard; Rich, Charles; Swartout, editor, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 318–329. Morgan Kaufmann, 1992.

[Harel 1984] D. Harel. Dynamic logic. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume II*, pages 496–604. D. Reidel Publishers, 1984.

[Hemaspaandra 1996] E. Hemaspaandra. The price of universality. *Notre Dame Journal of Formal Logic*, 37(2):174–203, 1996.

[Hollunder 1996] B. Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18:133–157, 1996.

[Hollunder & Baader 1991] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, Boston, MA, USA, 1991.

[Hollunder & Nutt 1990] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-04, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany, 1990.

[Hopcroft & Ullman 1979] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Horrocks *et al.* 1998] I. Horrocks, U. Sattler, and S. Tobies. A PSPACE-algorithm for deciding $\mathcal{ALCNI}_{R^+}$-satisfiability. LTCS-Report 98-08, LuFG Theoretical Computer Science, RWTH Aachen, 1998.

[Horrocks *et al.* 2000a] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264, 2000.

[Horrocks *et al.* 2000b] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In D. MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, number 1831 in Lecture Notes in Computer Science. Springer-Verlag, 2000.

[Horrocks 1997] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* Phd thesis, University of Manchester, 1997.

[Horrocks 1999] I. Horrocks. FaCT and iFaCT. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, number 22 in CEUR-WS (http://ceur-ws.org/), pages 133–135, 1999.

[Horrocks & Patel-Schneider 1999] I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[Horrocks & Sattler 2001] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 199–204. Morgan-Kaufmann, 2001.

[Hustadt & Schmidt 2000] U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caterra and G. Salzer, editors, *Automated Deduction in classical and non-classical logic*, vol. 1761 of *Lecture Notes in Artificial Intelligence*, pages 191–205. Springer-Verlag, 2000.

[Kamp & Wache 1996] G. Kamp and H. Wache. CTL - a description logic with expressive concrete domains. Technical Report LKI-M-96/01, Laboratory for Artificial Intelligence (LKI), Universitity of Hamburg, Germany, 1996.

[Knuth 1968] D. Knuth. *The Art of Computer Programming*, vol. 1. Addison-Wesley, 1968.

[Kozen 1982] D. Kozen. Results on the propositional μ-calculus. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and Programming, 9th Colloquium*, vol. 140 of *Lecture Notes in Computer Science*, pages 348–359. Springer-Verlag, 1982.

[Kracht & Wolter 1991] M. Kracht and F. Wolter. Properties of independently axiomatizable bimodal logics. *The Journal of Symbolic Logic*, 56(4):1469–1485, 1991.

[Kullmann *et al.* 2000] M. Kullmann, F. de Bertrand de Beuvron, and F. Rousselot. A description logic model for reacting in a dynamic environment. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS (http://ceur-ws.org/), pages 203–212, 2000.

[Küsters 1998] R. Küsters. Characterizing the Semantics of Terminological Cycles in $\mathcal{ALN}$ using Finite Automata. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 499–510. Morgan Kaufmann, 1998.

[Küsters 2001] R. Küsters. *Non-Standard Inferences in Description Logics*, vol. 2100 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[Ladkin & Maddux 1994] P. B. Ladkin and R. D. Maddux. On binary constraint problems. *Journal of the ACM*, 41(3):435–469, 1994.

[Ladner 1977] R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

[Lutz *et al.* 1997] C. Lutz, V. Haarslev, and R. Möller. A concept language with role-forming predicate restrictions. Technical Report FBI-HH-M-276/97, University of Hamburg, Computer Science Department, Hamburg, 1997.

[Lutz *et al.* 2001a] C. Lutz, U. Sattler, and F. Wolter. Description logics and the two-variable fragment. In D. McGuiness, P. Pater-Schneider, C. Goble, and R. Möller, editors, *Proceedings of the 2001 International Workshop in Description Logics (DL'01)*, number 49 in CEUR-WS (http://ceur-ws.org/), pages 66–75, 2001.

[Lutz *et al.* 2001b] C. Lutz, U. Sattler, and F. Wolter. Modal logic and the two-variable fragment. In L. Fribourg, editor, *Computer Science Logic*, number 2142 in Lecture Notes in Computer Science, pages 247–261. Springer-Verlag, 2001.

[Lutz *et al.* 2001c] C. Lutz, H. Sturm, F. Wolter, and M. Zakharyaschev. A tableau decision algorithm for modalized $\mathcal{ALC}$ with constant domains. *Studia Logica*, 2001. To appear.

[Lutz *et al.* 2001d] C. Lutz, H. Sturm, F. Wolter, and M. Zakharyaschev. Tableaux for temporal description logic with constant domain. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 121–136. Springer-Verlag, 2001.

[Lutz 1999a] C. Lutz. Complexity of terminological reasoning revisited. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 181–200. Springer-Verlag, 1999.

[Lutz 1999b] C. Lutz. Reasoning with concrete domains. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 90–95. Morgan Kaufmann, 1999.

[Lutz 2000] C. Lutz. NExpTime-complete description logics with concrete domains. In C. Pilière, editor, *Proceedings of the ESSLLI-2000 Student Session*, University of Birmingham, UK, 2000.

[Lutz 2001a] C. Lutz. Interval-based temporal reasoning with general TBoxes. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 89–94. Morgan-Kaufmann, 2001.

[Lutz 2001b] C. Lutz. NExpTime-complete description logics with concrete domains. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 45–60. Springer-Verlag, 2001.

[Lutz & Möller 1997] C. Lutz and R. Möller. Defined topological relations in description logics. In M.-C. Rousset, R. Brachman, F. Donini, E. Franconi, I. Horrocks, and A. Levy, editors, *Proceedings of the International Workshop on Description Logics (DL'97)*, pages 15–19, Gif sur Yvette (Paris), France, 1997. Université Paris-Sud, Centre d'Orsay.

[Lutz & Sattler 2000] C. Lutz and U. Sattler. Mary likes all cats. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS (http://ceur-ws.org/), pages 213–226, 2000.

[Lutz & Sattler 2001] C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logics. In F. Wolter, H. Wansing, M. de Rijke, and M. Zakharyaschev, editors, *Advances in Modal Logics Volume 3.* CSLI Publications, Stanford, CA, USA, 2001.

[Mayr & Meyer 1982] E. W. Mayr and A. R. Meyer. The complexity of the word problem for commutative semigroups and polynomial ideals. *Advanced Mathematics*, 46:305–329, 1982.

[Minsky 1975] M. Minsky. A framework for representating knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York, USA, 1975.

[Molitor 2000] R. Molitor. *Unterstützung der Modellierung verfahrenstechnischer Prozesse durch Nicht-Standardinferenzen in Beschreibungslogiken.* PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2000.

[Nebel 1990] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Nebel 1991] B. Nebel. Terminological cycles: Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann, 1991.

[Nebel & Bürckert 1995] B. Nebel and H.-J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[Papadimitriou 1994] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Patel-Schneider 1999] P. Patel-Schneider. DLP. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International*

Workshop on Description Logics (DL'99), number 22 in CEUR-WS (http://ceur-ws.org/), pages 140–141, 1999.

[Post 1946] E. M. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[Pratt 1979] V. R. Pratt. Models of program logics. In *Proceedings of the Twentieth Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979.

[Quillian 1968] M. R. Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, Cambridge, MA, USA, 1968.

[Randell *et al.* 1992] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176. Morgan Kaufman, 1992.

[Sattler 1996] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, vol. 1137 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1996.

[Sattler 1998] U. Sattler. *Terminological knowledge representation systems in a process engineering application.* PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 1998.

[Sattler 2000] U. Sattler. Description logics for the representation of aggregated objects. In W.Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI'00)*. IOS Press, Amsterdam, 2000.

[Sattler & Vardi 2001] U. Sattler and M. Vardi. The hybrid μ-calculus. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR 2001)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 76–91. Springer-Verlag, 2001.

[Savitch 1970] W. J. Savitch. Relationsship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[Schaerf 1993] A. Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993.

[Schild 1991] K. D. Schild. A correspondence theory for terminological logics: Preliminary report. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 466–471. Morgan Kaufmann, 1991.

[Schild 1993] K. D. Schild. Combining terminological logics with tense logic. In M. Filgueiras and L. Damas, editors, *Progress in Artificial Intelligence – 6th Portuguese Conference on Artificial Intelligence, EPIA'93*, vol. 727 of *Lecture Notes in Artificial Intelligence*, pages 105–120. Springer-Verlag, 1993.

[Schild 1994] K. Schild. Terminological cycles and the propositional μ-calculus. In P. T. Jon Doyle, Erik Sandewall, editor, *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 509–520. Morgan Kaufmann, 1994.

[Schmidt-Schauß 1989] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In A. Cohn, L. Schubert, and S.C.Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 421–431, Trento, Italy, 1989.

[Schmidt-Schauß & Smolka 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Spaan 1993a] E. Spaan. *Complexity of Modal Logics.* PhD thesis, Department of Mathematics and Computer Science, University of Amsterdam, 1993.

[Spaan 1993b] E. Spaan. The complexity of propositional tense logics. In M. de Rijke, editor, *Diamonds and Defaults*, pages 287–307. Kluwer Academic Publishers, 1993.

[Stock 1997] O. Stock, editor. *Spatial and Temporal Reasoning.* Kluwer Academic Publishers, Dordrecht, Holland, 1997.

[Stockmeyer & Meyer 1973] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *ACM Symposium on Theory of Computing (STOC '73)*, pages 1–9, New York, USA, 1973. ACM Press.

[Szpilrajn 1930] E. Szpilrajn. Sur l'extension de l'ordre partiel. *Fundamenta Mathematica*, 16:386–389, 1930.

[Tarski 1951] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, Berkeley, CA, USA, 1951.

[Tobies 2001a] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation.* PhD thesis, RWTH Aachen, 2001.

[Tobies 2001b] S. Tobies. PSpace reasoning for graded modal logics. *Journal of Logic and Computation*, 11(1):85–106, 2001.

[van Beek & Cohen 1990] P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:133–44, 1990.

[van Benthem 1983] J. F. A. K. van Benthem. *Modal Logic and Classical Logic.* Bibliopolis, Naples, Italy, 1983.

[van Benthem 1996] J. van Benthem. Temporal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4*, pages 241–350. Oxford Scientific Publishers, 1996.

[Vardi 1982] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC'82)*, pages 137–146, San Francisco, CA, USA, 1982.

[Vardi 1997] M. Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, vol. 31 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1997.

[Vardi & Wolper 1986] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logic of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.

[Vilain *et al.* 1990] M. Vilain, H. Kautz, and P. Van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufmann, 1990.

[Wolter & Zakharyaschev 1999] F. Wolter and M. Zakharyaschev. Temporalizing description logic. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems*, pages 379 – 402. Studies Press/Wiley, 1999.

Index

Curriculum Vitae

Name:	Carsten Lutz
Birthdate/location:	22 September 1971 in Hamburg
Nationality:	German
Marital status:	unmarried, one child

Education:

1977 - 1981	Grundschule in Rellingen
1981 - 1990	Johannes Brahms Gymnasium, Pinneberg, finished with Abitur
1990 - 1998	Student of computer science at the University of Hamburg.
1998	Diploma in computer science. Diploma thesis supervised by Dr. Ralf Möller: "Representation of Topological Information in Description Logics".
1998 - 2002	PhD student at the Teaching and Research Area for Theoretical Computer Science at RWTH Aachen.

Professional Activities:

1993 - 1994	System administration and software development for the German Aerospace Airbus AG (DASA), Hamburg.
1994 - 1998	Consultant work in Internet-related projects for High-Performance GmbH, Hamburg.
1998 - 1999	Researcher in the multinational ESPRIT project "Foundations of Data Warehouse Quality".
1999 - 2002	Researcher in the DFG project "Combinations of Modal and Description Logics" of Franz Baader and Frank Wolter.